Thèse n° 10 128

Regularization Techniques for Low-Resource Machine Translation

Présentée le 18 octobre 2023

Faculté des sciences et techniques de l'ingénieur
Laboratoire de l'IDIAP
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Alejandro RAMÍREZ ATRIO

Acceptée sur proposition du jury

Prof. J.-Ph. Thiran, président du jury
Dr J.-M. Odobez, Prof. A. Popescu-Belis, directeurs de thèse
Prof. A. Toral, rapporteur
Dr F. Yvon, rapporteur
Dr J. Henderson, rapporteur

École
polytechnique
fédérale
de Lausanne

2023

Abstract

Neural machine translation (MT) and text generation have recently reached very high levels of quality. However, both areas share a problem: in order to reach these levels, they require massive amounts of data. When this is not present, they lack generalization abilities. This is the main problem we address in our thesis: how can we increase the generalization abilities of these models when they are trained in low-resource settings? We propose various regularization techniques to address this problem.

In Part I of the thesis we study the impact of training the weights of a model so that they set in flatter regions of the parameter space, by indirectly guiding them with a more regularized training regime than would be normal in better-resourced settings. We pursue an empirical approach to this. Firstly, without directly measuring the landscape of the loss function in the parameter space, we show that in low-resource settings NMT systems benefit from more aggressive regularization, which can be achieved by modifying several hyper-parameters, and show that a combination of these factors improves scores more than any single factor. Our explanation is that a less precise optimizer – due to increased regularization – is more likely to fall into flatter regions of the loss landscape, leading to more robust systems. We test this hypothesis on a series of low-resource datasets and observe an improvement of quality of 3–6 BLEU points. Secondly, we propose a cost-effective method to directly estimate the flatness of the neighborhood of a point in the parameter space (a model checkpoint) using random perturbations and interpolation. We propose several metrics and compare them. Thirdly, we propose a method to directly train a system into flatter regions by looking ahead for variations of the loss function before performing gradient descent.

In Part II we show that the use of auxiliary and synthetic data for neural MT, which is another way to perform regularization, also improves quality in low-resource settings. Firstly, we simplify a state-of-the-art complex pipeline for low-resource translation with no loss in performance. Secondly, we propose a fixed-schedule multitask training regime, with improvements of 1–3 BLEU points. Thirdly, we demonstrate the design of a novel self-paced learning algorithm that balances languages on a multilingual many-to-one regime, by measuring model weight variation throughout training. Fourthly, we show that many-to-many systems improve over well-optimized unidirectional systems.

In Part III we present an approach to text generation for a low-resource domain, poetry. Firstly, we train a LM and design a rule-based algorithm that generates various structures and rhymes based on user's specifications. Secondly, we show that synthetic poetry generated by this system helps to fine-tune a LM so that it learns to generate appropriate poems without rule-based constraints. In other words, the use of synthetic data helps to generalize a LM trained on a small in-domain data set.

Overall, this thesis offers a unified perspective that improves our understanding of the following regularization techniques: hyper-parameter tuning, loss flatness measuring, multitasking, and usage of auxiliary data. The thesis also shows that these are effective and efficient strategies for improving low-resource neural machine translation and text generation.

Keywords: machine translation, text generation, low-resource training, regularization, machine learning.

Résumé

La traduction automatique neuronale (TA) et la génération de texte ont récemment atteint des niveaux de qualité très élevés. Cependant, les deux domaines partagent un même problème : pour atteindre ces niveaux, les modèles ont besoin de très grandes quantités de données. Lorsque de telles quantités ne sont pas disponibles, les capacités de généralisation des modèles sont très réduites. Le principal problème que nous abordons dans notre thèse est donc le suivant : comment augmenter les capacités de généralisation de ces modèles lorsqu'ils sont entraînés avec peu de ressources ? Nous proposons diverses techniques de régularisation pour résoudre ce problème.

Dans la première partie de la thèse, nous étudions l'entraînement des modèles de TA afin que leurs poids se situent dans des régions plus plates de l'espace des paramètres. Nous guidons indirectement les modèles avec un régime d'entraînement plus régularisé que lorsque de grands volumes de données sont disponibles. Nous poursuivons l'approche empirique suivante. Premièrement, sans observer directement l'aspect de la fonction de coût dans l'espace des paramètres, nous montrons que lorsque peu de données sont disponibles, les systèmes bénéficient d'une régularisation plus agressive, qui peut être obtenue en modifiant plusieurs hyperparamètres. De plus, une combinaison de ces paramètres améliore les scores plus que n'importe quel facteur utilisé seul. Notre explication est qu'un optimiseur moins précis -– en raison d'une régularisation accrue -– est plus susceptible de tomber dans des régions plus plates de la fonction de coût, conduisant à des systèmes plus robustes. Nous testons cette hypothèse sur diverses données de volume réduit observons une amélioration entre 3 et 6 points BLEU. Deuxièmement, nous proposons une méthode pour estimer directement la planéité du voisinage d'un point dans l'espace des paramètres (un état du modèle) en utilisant des perturbations aléatoires et l'interpolation. Nous proposons plusieurs métriques et les comparons. Troisièmement, nous proposons une méthode pour entraîner directement un système dans des régions plus plates en anticipant les variations de la fonction de coût avant d'effectuer une descente de gradient.

Dans la deuxième partie de la thèse, nous montrons que l'utilisation de données auxiliaires et synthétiques pour la TA neuronale, qui est un autre moyen d'effectuer une régularisation, améliore également la qualité lorsque peu de données sont disponibles. Premièrement, nous simplifions sans perte de performances une chaîne de traitement récente pour la TA. Deuxièmement, nous proposons un régime d'entraînement multitâche à programme fixe, avec des améliorations entre 1 et 3 points BLEU. Troisièmement, nous présentons un nouvel algorithme d'apprentissage multitâche qui équilibre les langues en mesurant la variation de poids du modèle tout au long de la formation. Quatrièmement, nous montrons que les systèmes de TA traduisant plusieurs langues sources et cibles sont meilleurs que des systèmes unidirectionnels, même optimisés.

Dans la troisième partie de la thèse, nous présentons une approche de la génération de texte pour un domaine à faibles ressources, la poésie. Premièrement, nous entraînons un modèle de langage et concevons des règles qui génèrent diverses structures et rimes en fonction des spécifications de l'utilisateur. Deuxièmement, nous montrons que les poèmes générés par ce système, utilisés comme données d'entraînement, permettent à un modèle de langage de générer des poèmes qui riment, sans contraintes basées sur des règles. Autrement dit, l'utilisation de données synthétiques permet de généraliser un modèle entraîné sur un petit ensemble de données du domaine.

Dans l'ensemble, cette thèse offre une perspective unifiée qui améliore notre compréhension des principales techniques de régularisation : réglage des hyperparamètres, aspect de la fonction de coût, entraînement multitâche, et utilisation de données auxiliaires. La thèse montre également que celles-ci sont des stratégies efficaces et efficientes pour améliorer la traduction automatique neuronale et la génération de texte à faibles ressources.

Mots clés : traduction automatique, génération de texte, entraînement à faibles ressources, régularisation, apprentissage automatique.

Acknowledgements

I am deeply grateful to my supervisors, Andrei Popescu-Belis and Jean-Marc Odobez for giving me the opportunity to work on the topics presented on this thesis, which I have enjoyed greatly. Particularly as someone coming at these issues from an interdisciplinary background, Andrei has always been extremely helpful and illuminating: I have learned a great deal from him. I am convinced that with another supervisor, I may not have finished my thesis; with him, this four-year project has been a pleasure.

I am also grateful to my thesis committee, Jean-Philippe Thiran (EPFL), James Henderson (IDIAP and EPFL), Antonio Toral (University of Groningen), and François Yvon (Sorbonne Université, UPMC/ISIR). They reviewed my thesis in great detail, produced a very interesting discussion of my work, and provided great feedback for its final version.

I am thankful too to my colleagues from the Institute of ICT at HEIG-VD, specially to those with whom I have had the pleasure of working with (and learning from) in various projects during these four years: Gabriel Luthier, Valentin Minder, Axel Fahy, Alexis Allemann, and Giorgos Vernikos.

I also want to acknowledge the Swiss National Science Foundation, who funded my thesis through the DOMAT project, as well as other projects that my lab was involved with, and in which I participated: Agora project "Digital Lyric", and projects "FamilyMT" and "Unisub" with Armasuisse.

Lastly, on a personal note, I am grateful to my close friends and family, who not only helped me during my PhD, but also since much earlier, making it possible for me to even be able to consider embarking on a project like this in the first place.

Yverdon-les-Bains, September 2023 À. R. A.

Contents

List of Figures

List of Tables

1 Introduction

In this thesis we propose several solutions based on regularization to address the generalization problems that arise from training in low-resource settings in two areas of Natural Language Processing (NLP): Neural Machine Translation (NMT) and Text Generation with Language Models (LMs). These solutions are: adaptation of hyper-parameters, training into flatter regions of the weight space, usage of multitask and multilingual training, and training on synthetic data.

In this chapter we will introduce some general (Subsection 1.1.1) and specific (Subsection 1.1.2) concepts in order to present the main research questions of the thesis (Subsection 1.2), and finally we will specifically present our contributions and the general outline of this thesis (Subsection 1.3).

1.1 Context of the Thesis

In recent years, the quality of neural machine translation has sharply grown, even leading to some reports of human parity (Hassan et al., 2018), although these have been challenged (Läubli et al., 2018; Toral, 2020; Läubli et al., 2020). Translation systems are turning more and more useful, becoming an everyday tool for some, particularly after the re-development of Google Translate into a neural system in late 2016 and other neural competitors such as DeepL in 2017. Similarly, and even more recently, text generation has gone through a comparable process in its numerous applications, with some of the more prominent ones being question and answering. Few things could make this clearer than the immediate popularity of ChatGPT by OpenAI after its introduction in late 2022. However, this increase in quality is very closely tied to an increase in training data (and therefore in computational cost and training time). When training NMT and LM systems without enough training data, problems arise when testing on new, unseen data.

The difference between a model's scores on a train set and on a test set is called the *generalization* gap. A model's generalization ability represents how well its training allows it to perform on unseen data. In this sense, test error can also be called generalization error. Successfully training a neural network means lowering its generalization error as much as possible, given the available training data.

Current NMT systems usually consist of a Transformer-based network trained on parallel texts, i.e. datasets of sentence-aligned (parallel data), one of them being the human translation of the other. The model is given one of the datasets as input, and is trained to generate the other as output (the source and target, respectively). The amount of parallel data that is available determines whether we are in a low-resource scenario.

1.1.1 Generalization Gap and Regularization

Sometimes we constrain our model during training in order to limit its ability to identify patterns in the training data, expecting that the more generalizable patterns will be learned, and the more local ones will not. This means that when we apply these constraints, we expect the model to perform better on test sets, although it may perform worse (or simply not improve) on the training set. Such constraints are called *regularization techniques*.

Regularization is one of the most common ways to lower generalization error. This is achieved by a range of techniques that attempt to lower generalization error without necessarily reducing training error. Early stopping or dropout, for instance, are unlikely to lower training error but they may lower test error. On the contrary, increasing the training data or the amount of epochs during which the model is trained should reduce training error along with test error – as such, neither is a regularization technique.

Now that we have introduced the notion of regularization, let us also introduce three related concepts to better illustrate it: *overfitting*, *capacity*, and *bias versus variance trade-off*. Indeed, any regularization technique can be understood as constraining the model in order to reduce overfitting and lower effective capacity, which we define as follows. *Overfitting* occurs when a model decreases or stabilizes its training error, but increases its test or validation error. Usually it is characterized by an *over-training* curve during the training schedule, where the error on some validation set lowers in an initial phase, but after a certain point it begins to increase again, due to the model over-focusing on patterns from the training data, while training error remains stable or continues to decrease. A generic representation of overfitting is shown in Figure 1.1. In NMT this can become apparent by the model repeating a small amount of phrases over and over in its output, for example. We say that the model is overfitting after the lowest point of the test error curve. Intuitively, we can think of this as the point where the model starts recognizing too many patterns in the training data that are not universal or cannot be generalized to other datasets.

We can think of this excessive recognition of patterns in the training data as a problem in the *capacity* of the model. *Representational capacity* (or model complexity) refers to the family of functions that the learning algorithm can choose from to maximize a training objective.

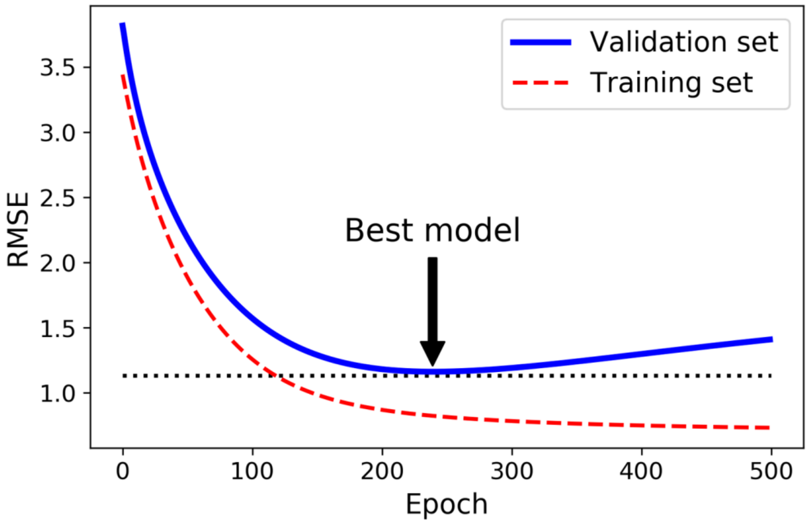

Figure 1.1: Graphical representation of overfitting due to over-training (Géron, 2017, Figure 4.20). In this image, RMSE is simply an example of an error metric.

Effective capacity is the capacity the model has once a function has been chosen.[1] More complex models (i.e. models with a larger capacity) will be more likely to overfit, and therefore will require more regularization. In other words, the larger the representational capacity of a model, the more constraints we need to place in order to lower its effective capacity — but without lowering it so much that the model underfits. *Encoding* or latent space is part of a model's capacity, regarding specifically the richness with which each input token can be represented. Embedding size and sequence context (input layer size) are its two main parameters.

We have introduced two of the three fundamental concepts – overfitting and capacity – to better understand the notion of regularization. The following observation will lead us to introduce the third one: the more capacity a model has, the more variance error and the less bias error it will have, as represented in Figure 1.2. Generalization error is the sum of the *bias* and the *variance* errors (plus irreducible error, due to the natural variability of the data).

A high-variance model is excessively sensitive to variations in the data, which will make it more likely to overfit.[2] If we were to feed an infinitely large training set to our model (with infinite capacity), the only generalization error possible would be bias error: a high-bias model learns wrong patterns on the data, which makes it more likely to underfit.[3] Although high bias

[1]Representational capacity is determined by various factors, including the amount of layers of the neural network, linear vs. nonlinear activations, total amount of trainable parameters (weights), etc., whereas effective capacity is determined by various factors of the training regime, such as early stopping or dropout. If the chosen training function once the model has been trained is suboptimal, the model's effective capacity will be lower than its maximal representational capacity.

[2]In its extreme form, a *high-variance low-bias* model represents a data sample as an atomic element: it is unable to establish any patterns between two data samples if there is just one difference among the two.

[3]In its extreme form, a *high-bias low-variance* model outputs the same prediction regardless of what input data

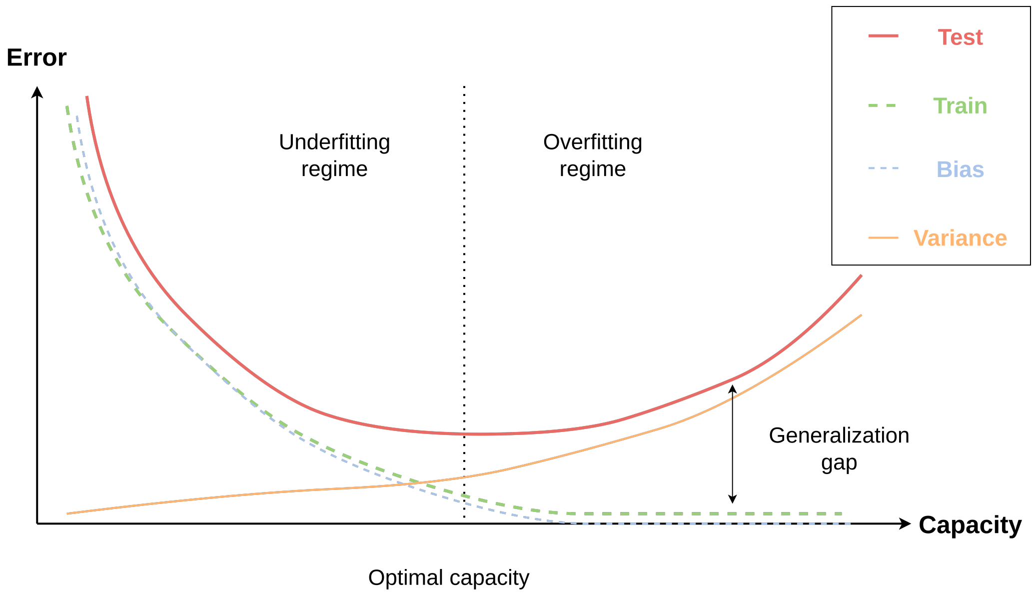

Figure 1.2: Effects of capacity on overfitting and generalization, and bias-variance error. Inspired by Goodfellow et al. (2016, Figure 5.3, 5.6).

and variance or low bias and variance models are possible, in practice successfully training a model is a trade-off between bias and variance. Regularization consists in lowering the variance error without increasing bias error too much.

We could be tempted to position all generalization errors on the underfitting-overfitting error spectrum: either a model does not generalize well to other datasets because it has not recognized all useful patterns in its training, or because it has over-focused on some of them that are not easily generalizable. Suggestions that generalization can be divided between over- and underfitting have appeared in the machine learning literature for a long time (Guyon et al., 1991; Wang et al., 1993; Hochreiter and Schmidhuber, 1994).

Other likely causes of generalization errors, such as insufficient training data, or domain mis-match between train and test sets, could also *a priori* be placed on the same spectrum. Indeed, Figure 1.1 shows a typical overfitting situation as a common cause of generalization error. The solution is to stop training the model (before getting to 500 epochs), around the point marked as "best model", which would lower test error while slightly increasing train error. This fits our previous definition of regularization, and can be achieved in this case by early stopping.

it is given.

1.1.2 Flat Minima Theory

The loss of a model is a real-valued function of the model's parameters (the weights of the network) that serves as the model's training objective. In other words, it is a performance measure that the model iteratively tries to minimize. Throughout the thesis, the negative log-likelihood is computed: the cross-entropy between the training data (typically in a minibatch, a small subset of the training data) and the predictions of a model on that same data. Formally, a loss function J of a model with weights θ is defined as

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(y|x),$$

where p_{model} are the predictions of the model, and x, y is the input data and correct labels, respectively (Goodfellow et al., 2016, Eq. 6.12).

The aspect of the loss, which we call the *loss landscape*, depends on the data that is used to compute the loss for a given value of the weights. Two different datasets, such as for instance train vs. test data, will lead to two different landscapes. Nonetheless, the more similar the two datasets are, the more we expect the two landscapes to be similar. In a nutshell, the *flat minima theory* states that a model whose weights are in a region where the loss landscape is rather flat will have a higher chance of generalizing well compared to a model whose weights are around a sharper point of the loss landscape (Hochreiter and Schmidhuber, 1994; Keskar et al., 2016). The higher chance of generalizing well is due to the fact that on a new dataset, the loss landscape is different from the training set, but if the region of the parameters is flat, then the loss on the new dataset will be close to the training loss, hence presumably still quite low.

How can we guide a model's weights during training towards flatter minima of the loss landscape? Certain hyper-parameters can help to increase the amount of regularization that a model has during training: some examples are the batch size, the learning rate, or the dropout. A more regularized model will have a less precise optimizer over the train set, and therefore will be more likely to set its weights in flatter regions rather than in sharper ones, according to the flat minima theory. We illustrate this in Figure 1.3, and we further analyze the literature on this theory in Section 2.4.

As introduced in Subsection 1.1.1, overfitting is usually characterized by some form of over-training curve where the generalization gap is deemed "too large" (Goodfellow et al., 2016, page 110). It is less clear whether the generalization error resulting from optimizing weights into suboptimal regions of the loss landscape (i.e. sharper ones, as explained above) is still due to overfitting. Some authors have claimed that regularization, or the use of less accurate gradients, leads the weights towards flatter regions of the weight space during optimization, and therefore explicitly minimize overfitting (Hochreiter and Schmidhuber, 1994). Other authors, however, have explicitly rejected the idea that the improved generalization is due to less overfitting (Keskar et al., 2016).

In Figure 1.4, we illustrate with examples of real systems the problem of explaining the role

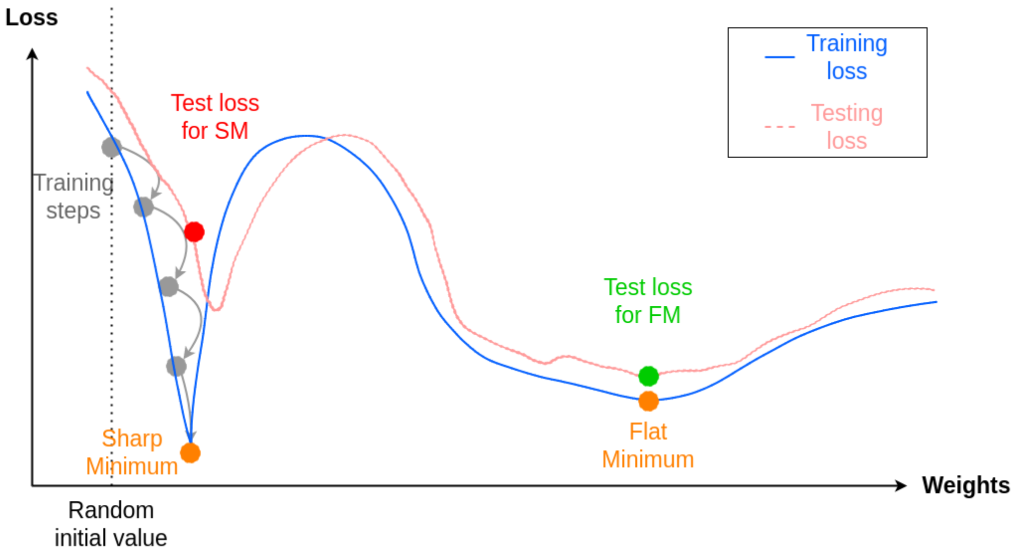

Figure 1.3: Illustration of the effect on generalization of the loss landscape (the values of the loss in the space of the model's weights). Weights from a flat minimum generalize well in the sense that the test loss will likely be similar to the train loss.

of regularization, in relation to flat minima and the generalization gap (or, more broadly, training and testing error). In the upper left part of Figure 1.4 we show an graph from Keskar et al. (2016, Figure 2): it shows a modification of Alexnet (a convolutional neural network) trained on a computer vision task (CIFAR-10) either with small batches (SB) or with large ones (LB), (i.e. with more or less regularization, respectively). The other three graphs are our own Transformer-based models for low-resource NMT tasks, trained with baseline hyper-parameters or with a combination of more regularized ones: dropout, gradient norm, warmup steps, and learning rate.[4]

While Figure 1.1 showed a typical example of overfitting, which can be solved by reducing the amount of over-training (e.g. performing early stopping), consider, however, the upper left graph in Figure 1.4, which appears to show a different cause for generalization error. Obtaining the lowest test error in this setting requires increasing regularization (SB), which improves test and train accuracy. Additionally, if we look at the final accuracy results between SB and LB, there is no significant reduction in the generalization gap either. This seems to show that the generalization error from the less regularized model (LB) is not due to overfitting – a point that the authors themselves explicitly make (Keskar et al., 2016).

The remaining three graphs in Figure 1.4 are all generated from our experiments with neural machine translation presented in Chapter 3, in similar settings: their training sets are all low-resource translation sets, ranging from 20k lines to 120k, and each of them trained in a different language pair. The changes in hyper-parameters between Baseline and Regularized

[4]The models are presented in detail in Chapter 3, Section 3.8. Upper right: DE-EN with 120k training lines, lower left: SK-EN with 60k lines, lower right: HSB-DE with 20k lines.

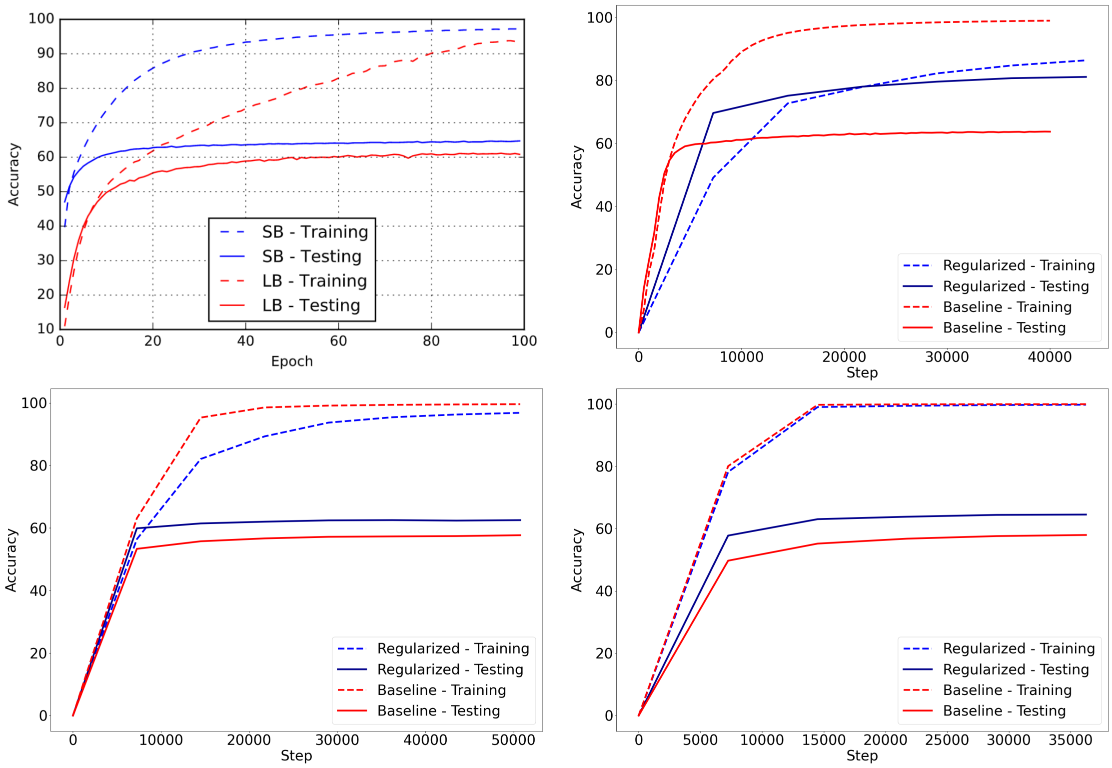

Figure 1.4: Effects of increased regularization factors on the generalization of a model, as found by experiments in computer vision (Keskar et al., 2016, Figure 2) (upper left) and low-resource NMT (from our experiments in Chapter 3). Models with more regularization are in blue.

models are identical. In all cases the Regularized model's test accuracy increases. However, we can see different behaviors regarding their train and the test set accuracies: in the upper right graph, the training accuracy of Regularized underperforms when compared with the training accuracy of Baseline, and the generalization gap of Regularized is very small, i.e. the accuracy in the train and test set are very similar. In the lower left graph, the training accuracies of Baseline and Regularized are much more similar, but the reduction in generalization gap that Regularized offers is a lot smaller. In the lower right graph, the training accuracies of Baseline and Regularized are identical, but we still observe an improvement in generalization from Regularized.

In all of the fourteen models trained in Section 3.8, we did not observe in any case the *training* accuracy of the more regularized model ever surpassing the training accuracy of the baseline model. The improvements provided by the Regularized models is not due to a reduction of overfitting with respect to the Baseline ones, since no over-training curve can be observed, and both training and testing accuracy continue improving throughout its training. Instead, as we argue in Chapter 3 this is due to this more aggressive regularization implicitly guiding the weights towards flatter regions of the loss landscape.

As a conclusion, we can see that the impact of regularization on low-resource NMT is not fully understood, as discussed in Section 2.5, but explaining it only as a reduction in overfitting does not seem sufficient. At the same time, differences between training and testing performance (such as the generalization gap) also seem unstable and hard to predict, as shown in the disparity across our three models in Figure 1.4. Results from computer vision explaining it in terms of flat-optimization cannot be accepted in a straightforward way for NMT, due to the big variability on results between datasets.

1.2 Research Questions

Both machine translation and text generation experience the problem of data shortage: in order to function properly and achieve high, usable levels of quality, lots of data are needed to train these neural networks. In a best-case scenario, if data is available, proper training requires long training times, which can be unaffordable. In a worst-case scenario, data shortage blocks the development of systems for most practitioners, particularly for some languages or domains. *Low-resource settings* are those where the amount of data available to train the systems is small. Hence, training with such data requires careful tuning of the training process, sometimes also with particularly tailored techniques, like back-translation, in order to achieve satisfactory results. In a low-resource setting, maximizing a model's generalization abilities is harder to do, because the model does not have as good a frame of reference to judge whether a pattern in the training data is an artifact of the small dataset, that would not be prominent anymore if it were trained on more data, or whether it is a pattern in a language, or between two languages, that should hold in other datasets as well (in other words, that is generalizable).

The main goal of our thesis is to build neural models by training them to identify patterns in the training data which are generalizable to other pieces of data. We do not want our model to learn all patterns occurring on the training set, since then it will perform poorly on test sets where these patterns are not found. How, then, can we increase the robustness of encoder-decoder models when trained in low-resource settings for machine translation and text generation? We break down this goal into three main research questions, which we answer respectively in each part of the thesis.

1. **What role does regularization through hyper-parameters play in low-resource NMT?** Since some hyper-parameters such as batch size or learning rate can act as regularization factors, their impact, both individually and when combined, must be understood in low-resource settings. In particular, their effect on flat vs. sharp optimization also affects the generalization abilities of low-resource NMT models. This has been studied with some detail in computer vision, but much less in NLP, and minimally in low-resource NMT.

2. **How to make use of auxiliary data for low-resource NMT?** Data in the form of monolingual data, synthetic data, or additional languages can help the training of low-resource

NMT systems. We put aside approaches that use several models (transfer learning) due to considerations of efficiency and simplicity. Approaches that focus on the exploitation of such data within one model include multitasking, multilingual training, and back-translation. Although well-performing low-resource systems often make use of these techniques, the individual impact of each task or dataset is still not well understood. This calls for a study of the optimal use of auxiliary data, particularly when it features different levels of complexity, or when many different datasets are available, such as in multilingual NMT.

3. **How to improve form and textual diversity for text generation with LMs in low-resource settings?** When training in a low-resource setting, it is not only sequence-to-sequence models on tasks like MT that encounter the problems presented above: LMs face the same problems. In particular, when used for text generation, autoregressive LMs trained or fine-tuned with small amounts of data easily overfit to their training data, and they require of some form of control over their generation in order to be able to properly output a variety of forms and domains.

1.3 Contributions and Outline of the Thesis

When we train an MT system in a low-resource setting, we run intro problems that call for special solutions that we do not have as much need when we have plenty of available training data. Intuitively, these problems mainly arise from the fact that our model does not have enough data to differentiate between patterns specific to the training sample and those that will allow it to generalize to unseen data.

In this thesis, we propose three regularization solutions to answer the research questions regarding the difficulties of low-resource training: **i)** in Part I, we show how to use regularization factors to indirectly guide an NMT model to settle on flatter regions of its parameter space, and propose a method to estimate the neighborhood of the loss landscape of a model; **ii)** in Part II, we study the usage of multitasking with denoising tasks or additional languages for NMT; **iii)** and in Part III, we demonstrate that we can use synthetic data for LMs for poetry generation, a very specific text generation task.

In Chapter 3 (Part I) we explore the roles and interactions of the hyper-parameters governing regularization, and propose a range of values applicable to low-resource neural machine translation. We demonstrate that default or recommended values for high-resource settings are not optimal for low-resource ones, and that more aggressive regularization is needed when resources are scarce, in proportion to their scarcity. We explain our observations by the generalization abilities of sharp vs. flat basins in the loss landscape of a neural network. Results for four regularization factors corroborate our claim: batch size, learning rate, dropout rate, and gradient clipping. Moreover, we show that optimal results are obtained when using several of these factors, and that our findings generalize across datasets of different sizes and languages.

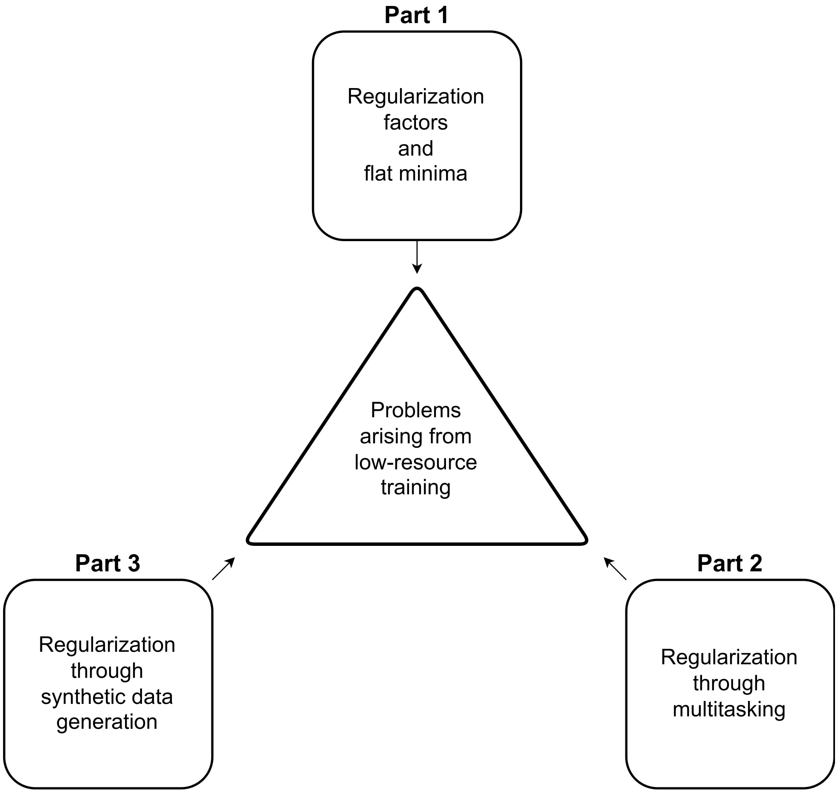

Figure 1.5: Diagram of the thesis. We propose a solution in each of the three parts of the thesis to the problems that arise from training in a low-resource setting.

In Chapter 4 (Part I) we propose a method to estimate the flatness of the neighborhood of a point (a model checkpoint) in the parameter space. We randomly perturb the model's weights and define some local region around the point by measuring the euclidean distance between the perturbed weights and the model checkpoint. Then we define several models with linear interpolation between the original point and the various perturbation to simulate a much less expensive mapping of the loss landscape around the central point. We propose various metrics to measure the flatness of this neighborhood. We also propose a method to directly train a system into flatter regions. This method uses our estimations of the neighborhood of the loss landscape of a point, and consists of comparing various candidate sets of weights before updating the model, and choosing one based on its neighborhood's flatness.

In Chapter 5 (Part II) we propose a simplified pipeline for low-resource and unsupervised MT, which we compare to the best submissions to the WMT 2021 Shared Task on Unsupervised MT and Very Low Resource Supervised MT. Training neural MT systems for low-resource language pairs or in unsupervised settings (i.e. with no parallel data) often involves a large number of auxiliary systems. These may include parent systems trained on higher-resource pairs and used for initializing the parameters of child systems, multilingual systems for neighboring languages, and several stages of systems trained on pseudo-parallel data obtained through back-translation. Our pipeline only needs two parents, two children, one round of back-translation for low-resource directions and two for unsupervised ones and obtains better or similar scores when compared to more complex alternatives.

In Chapter 6 (Part II) we present the systems submitted by our team from the Institute of ICT (HEIG-VD / HES-SO) to the Unsupervised MT and Very Low Resource Supervised MT task. We first study the improvements brought to a baseline system by techniques such as back-translation and initialization from a parent model. We find that both techniques are beneficial and suffice to reach performance that compares with more sophisticated systems from the 2020 task. We then present the application of this system to the 2021 task for low-resource supervised Upper Sorbian (HSB) to German translation, in both directions. Finally, we present a contrastive system for HSB-DE in both directions, and for unsupervised German to Lower Sorbian (DSB) translation, which uses multi-task training with various training schedules to improve over the baseline.

In Chapter 7 (Part II) we design and present a self-paced multilingual translation system. In particular, we design a many-to-one system that balances the training on tasks (which are translations into English) based on the smoothed per-task variation of training weights, measured with Kullback-Leibler divergence. Our objective is to not over-train on tasks in which the model is already competent, and allocate more training time to tasks in which it is not. We observe that training on simple, uniformly balanced, multilingual updates performs better than more sophisticated monolingual updates. We also note that the model's weight variation in the training set, as measured by Kullback-Leibler divergence, is not a good competence measure, at least when dealing with small datasets. This is because with smaller datasets, confirming that a model has been well-trained on the training set does not entail a good generalization ability.

In Chapter 8 (Part II) we present additional results, which complement those in Chapter 7 (Part II), on the impact of increased regularization and many-to-many training for low-resource translation. We observe that very low-resource translation benefits greatly from a combination of the two, when translating to up to three languages. Moreover, better-resourced translation tasks (but still low-resource ones) benefit less from added target languages. With this combination, we observe improvements of around 8 BLEU points for a lower-resourced pair and 5 for a more resourced one, when compared to a less regularized unidirectional baseline.

In Chapter 9 (Part III) we describe a system for interactive poem generation, which combines neural language models (LMs) for poem generation with explicit constraints that can be set by users on form, topic, emotion, and rhyming scheme. LMs cannot learn such constraints from the data, which is scarce with respect to their needs even for a well-resourced language such as French. We propose a method to generate verses and stanzas by combining LMs with rule-based algorithms, and compare several approaches for adjusting the words of a poem to a desired combination of topics or emotions. An approach to automatic rhyme setting using a phonetic dictionary is proposed as well. Our system has been demonstrated at public events, and log analysis shows that users found it engaging.

In Chapter 10 (Part III) we propose a novel solution for learning to rhyme, which is required for poem generation with language models, based on synthetic data generated with a rule-based rhyming algorithm. The algorithm and an evaluation metric use a phonetic dictionary and the definitions of perfect and assonant rhymes. We fine-tune a GPT-2 English model with 124M parameters on 142 MB of natural poems and find that this model generates consecutive rhymes infrequently (11%). We then fine-tune the model on 6 MB of synthetic quatrains with consecutive rhymes and obtain nearly 60% of rhyming lines in samples generated by the model. Alternating rhymes (ABAB) are more difficult to model because of longer-range dependencies, but they are still learnable from synthetic data, reaching 45% of rhyming lines in generated samples.

1.4 Chapters of the Thesis as Publications

Work on this thesis has resulted in published articles in peer-reviewed conferences. We now present each of the chapters that correspond to a published piece, plus my specific contribution in case of collaborative work.

Chapter 3 “Regularization Factors and Flat Minima":

- **Àlex R. Atrio** and Andrei Popescu-Belis. 2021. Small Batch Sizes Improve Training of Low-Resource Neural MT. In *Proceedings of the 18th International Conference on Natural Language Processing (ICON 2021)*, pages 18–24, National Institute of Technology Silchar, Silchar, India.
- **Àlex R. Atrio** and Andrei Popescu-Belis. 2022. On the Interaction of Regularization Factors in Low-resource Neural Machine Translation. In *Proceedings of the 23rd Annual Conference of the European Association for Machine Translation (EAMT 2022)*, pages 111–120, Ghent, Belgium.

Chapter 5 “A Simplified Training Pipeline for Low-Resource and Unsupervised MT". I was responsible for the design of the entire system, with help from the second author for the implementation:

- **Àlex R. Atrio**, Alexis Allemann, Ljiljana Dolamic and Andrei Popescu-Belis. 2023. A Simplified Training Pipeline for Low-Resource and Unsupervised Machine Translation. In *Proceedings of the Sixth Workshop on Technologies for Machine Translation of Low-Resource Languages (LoResMT 2023)*, Dubrovnik, Croatia.

Chapter 6 “Fixed-Scheduled Multitasking". I was responsible for the design of the baseline system and the design and implementation of the contrastive system:

- **Àlex R. Atrio**, Gabriel Luthier, Axel Fahy, Giorgos Vernikos, Andrei Popescu-Belis, and Ljiljana Dolamic. 2021. The IICT-Yverdon System for the WMT 2021 Unsupervised MT and Very Low Resource Supervised MT Task. In *Proceedings of the Sixth Conference on Machine Translation (WMT 2021)*, pages 973–981, Online.

Chapter 7 “Can the Variation of Model Weights be used as a Criterion for Self-Paced Multilingual NMT?". I was responsible for the design of the self-paced algorithm, which was implemented by the second author:

- **Àlex R. Atrio**, Alexis Allemann, and Andrei Popescu-Belis. 2023. Can the Variation of Model Weights be used as a Criterion for Self-Paced Multilingual NMT? *Submitted to ACL Rolling Review (4 pages).*

Chapter 9 “Constraining at Inference-Time for Targeted Generation". I designed and implemented the first generation stage (using a general LM) and the module applying the rhyming scheme at the last stage of the process:

- Andrei Popescu-Belis, **Àlex R. Atrio**, Valentin Minder, Aris Xanthos, Gabriel Luthier, Simon Mattei, and Antonio Rodriguez. 2022. Constrained Language Models for Interactive Poem Generation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference (LREC 2022)*, pages 3519–3529, Marseille, France.

Chapter 10 “Domain-Specific Data Augmentation". I contributed to the design of the experiment and the design and implementation of an automatic measure of rhyming, while the implementation of the learning experiment was done by three EPFL MSc students (3rd, 4th and 5th authors):

- Andrei Popescu-Belis, **Àlex R. Atrio**, Bastien Bernath, Étienne Boisson, Xavier Theimer-Liemard, Teo Ferrari and Giorgos Vernikos. 2023. GPoeT: a Language Model Trained for Rhyme Generation on Synthetic Data. In *Proceedings of the 7th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature (LaTeCH-CLfL 2023)*, Dubrovnik, Croatia.

During this thesis I have participated in the following projects:

1. “On-demand Knowledge for Document-level Machine Translation" (**DOMAT**), funded by the Swiss National Science Foundation, grant n. 175693 (2019-2023).
2. **FamilyMT**, funded by Armasuisse (2020-2021).
3. **UNISUB**, funded by Armasuisse (2022-2023).
4. Agora project “**Digital Lyric**", funded by the Swiss National Science Foundation, (grant n. 184330, Agora Optimus prize) (2019-2020).

2 Background

In this chapter we present background concepts that are relevant to this thesis, either from general machine learning or specifically from NMT. Section 2.1 covers various general concepts from machine learning, from the theoretical aspects of regularization and generalization, to more applied ones such as techniques like dropout, or the function of the learning rate. In Section 2.3 we introduce concepts specific to NMT and Language Models, mainly techniques commonly used in a low-resource setting, relevant for Parts II and III. Section 2.4 discusses theories regarding the parameter space, sometimes called the loss landscape, which we refer to in Part I. In Section 2.5 we discuss recent studies on the effects of regularization on NMT, which we also refer to in Part I. Finally, in Section 2.6 we cover related work on poetry generation, both constrained and unconstrained, relevant to Part III.

2.1 Fundamentals of Machine Learning

In this section we present three core, theoretically important concepts on machine learning: regularization, capacity, and generalization.

2.1.1 Definitions of Regularization

There is no single theoretical definition of regularization: Goodfellow et al. (2016, page 224) view it as a collection of methods "intended to reduce generalization error but not training error." Géron (2017, pages 57, 406) defines it as "constraining" to make a model simpler and reduce overfitting, thus improving generalization ability. Schmidhuber (2015, page 89) simply refers to regularization as "searching for solution-computing but simple, low-complexity" models. Overall, regularization is commonly understood in terms of reducing overfitting, as well as reducing model complexity (Section 2.1.2), improving generalization (Section 2.1.3), (which is related to data diversity, see Section 2.3.4).

Recent NMT models are based on the Transformer (Vaswani et al., 2017), a deep encoder-decoder neural network which is quite sensitive to the hyper-parameters governing regularization factors during training. In the absence of a general treatment of regularization factors, most studies combine them empirically and search only a very small part of the hyper-parameter space.

Peng et al. (2015) study regularization techniques independently as well as in combination, although without a common theoretical underpinning. On two NLP tasks, they observe that using two factors – namely, L2 norm of weights and embeddings, and dropout – is better than using them independently. Moreover, when using both factors, if one is set to its optimal value obtained when used alone, the other one must be lowered, which means that the optimal values obtained individually are not optimal when used together.

Kukačka et al. (2017) provide a taxonomy of regularization factors, but continue to define them simply as techniques to improve generalization. Similarly, in their survey, Moradi et al. (2020) consider as regularization any "component of the learning process or prediction procedure that is added to alleviate data shortage," but do not provide a common measure of regularization or consider the combination of factors.

2.1.2 Capacity and Encoding Space

Regularization is related to the complexity of a network: a generalizable model tends to be a large model with good regularization, instead of a more tight-fitting one (Goodfellow et al., 2016, page 225). Likewise, regarding overfitting, a classic consideration is that an over-capacity model (w.r.t. the complexity of the task) is more likely to overfit when not properly regularized (Goodfellow et al., 2016, page 110). A model with sub-optimal capacity will underfit, and as capacity is increased, assuming no additional constraints, training error will start to go down, but eventually generalization error will increase. Super-optimal capacity means the size of the gap between the training and test errors increases while decreasing training error, thus overfitting (Goodfellow et al., 2016, page 113).

The complexity of a model can be presented as representational capacity when considering the family of functions "the learning algorithm can choose from when varying the parameters in order to reduce a training objective", but since the final training function is likely not optimal, the network's effective capacity is smaller than its representational capacity (Goodfellow et al., 2016, 111). Regularization, such as early stopping, can be considered as constraining the effective capacity of a model which has much larger representational capacity (Goodfellow et al., 2016, 243).

2.1.3 Generalization

Bias error is the measure of the expected deviation between the predictions of a model and the true value of the training function. Formally, the bias of a model $\hat{\theta}_m$ is

$$\text{bias}(\hat{\theta}_m) = \mathbb{E}(\hat{\theta}_m) - \theta,$$

where θ is the true data-generating function (Goodfellow et al., 2016, Eq. 5.20).

Variance error is the deviation that can be caused between different models depending on particular sampling of the training data (Goodfellow et al., 2016, page 127). Géron (2017, page 196) defines generalization error as the sum of bias error, variance error, and noise or irreducible error. Bias error means the model likely underfits the training data, and excessive variance error makes the model overfit, with both of them limiting generalization capacity. Additionally, a model's complexity usually increases its variance and reduces its bias. Regularization, then, can also be understood as a trade-off between bias and variance (Goodfellow et al., 2016, page 225).

In order to diminish the generalization gap, Lin et al. (2020) proposed to use an efficient method known as extragradient (Korpelevich, 1976), consisting of computing the gradient of a local extrapolated point in the parameter space for distributed training, which is different from the actual point from which the model will be updated, essentially performing a look ahead step. This results in a smoothing of the loss landscape, which stabilizes the trajectory of training over the loss landscape while avoiding sharp minima, that is, points in the parameter space where the loss changes suddenly in their immediate neighborhood (see Section 2.4).

The connection between generalization and memorization is also related to the generalization gap. For instance, when training a network on a classification task with random labels, the model is able to learn the training set: Zhang et al. (2021a) show that standard convolutional networks (CNNs) have the necessary effective capacity to memorize random labels.

Chatterjee and Zielinski (2022) study the following question: why are neural networks able to generalize when they have enough capacity to memorize the data? They hypothesize that this is related to the similarity between training examples. If learning different examples results in moving in different directions in the parameter space, learning an average of them will give more importance to shared directions among examples. The more commonality there is for the gradient on each individual example, the larger the average gradient will be, and therefore the updates of the weights will be more significant as examples are more similar. In the opposite case, if a group of examples have no common direction in their gradients, training on their average will result in memorization.

Over-parameterized networks, the standard in modern deep learning, are able to generalize well, despite a seeming unbalance between their bias and variance, and despite having the necessary capacity to memorize the training data. An explanation for this has been given in the form of "deep double descent" (Belkin et al., 2019): as model capacity increases, the test error initially decreases, and then increases as the model overfits (as expected in a classical setting), but then starts to decrease again after a point of over-parameterization of the model. This point is called an interpolation threshold, i.e. a point in the representational capacity of a model where the model can memorize the training data. This point determines an over- or under-parameterized model, and the former can still generalize well, even though it could simply memorize the data. The authors observed this behavior on a variety of computer vision tasks, without experimenting with Transformer-based models.

A similar behavior was observed by Power et al. (2022), which the authors call “grokking", where long after the model has overfit, test error is very high, and no changes in train or test error have occurred, test error suddenly decreases until achieving very high generalization scores.[1] They show that dataset size is a relevant factor to explain why over-parameterized networks are able to generalize well, specifically by observing that models require larger amounts of optimization to generalize when training on smaller datasets, past the point of overfitting. Davies et al. (2023) provide a potentially unifying treatment of grokking and deep double descent, although in other papers they are still treated as separate concepts, despite their relatedness in terms of generalization, capacity, and overfitting.

In her recent thesis, Karimi Mahabadi (2023) explores generalization for pre-trained language models when fine-tuned for specific NLP tasks, such as natural language understanding or entailment, although not MT. She proposes to remove biases from the training data by leveraging these biases with bias-only models, which can improve generalization of a base model. The bias-only models are used during the training of the base model to adjust the base model’s loss by weighting down the biased samples and focusing training on harder examples, a kind of implicit scheduling (Karimi Mahabadi et al., 2020). Additionally, she proposes to use regularization by compressing the encoder’s sentence representation and deleting irrelevant features when fine-tuning on low-resource target tasks (Karimi Mahabadi et al., 2021a). She also uses multitasking to increase generalization abilities, in particular during the fine-tuning regime, by defining adapter hypernetworks that are shared across different tasks, trained to generate task and layer-specific adapter parameters (Karimi Mahabadi et al., 2021b). Although research on generalization for NLP is active, further research on generalization for MT on low-resource settings still needs to be done.

Li et al. (2020) showed that extremely over-parameterized models can be more computationally-efficient than smaller models, because they can converge earlier. Additionally, vastly over-parameterized models are more robust to compression than smaller ones, so they recommend to prune the larger models to a smaller model instead of training a smaller model from scratch.

Until now, we have shown how the relation between regularization, capacity, and generalization is not straightforward. Classical treatments like trade-off between bias and variance to explain overfitting or lack of generalization, for instance, are not sufficient in modern deep learning treatments. More recent approaches focus on characteristics of the data (Chatterjee and Zielinski, 2022), training time (Power et al., 2022), over-parameterization (Belkin et al., 2019; Li et al., 2020), and recent interest specifically on NLP (Hupkes et al., 2022; Karimi Mahabadi, 2023).

The amount of data available to train also is an important factor for determining generalization. Bottou and Bousquet (2011) propose a qualitative difference between small-scale and large-

[1] “to grok" means to understand something extremely well.

scale training, due to computational bottlenecks, where the latter depends on the properties of the training function, and the computational properties of the training algorithm (SGD, in their case). They also note that faster convergence is not necessarily desirable, since it may lead to overfitting. We continue our treatment of regularization, specifically for NMT, in Sections 2.4 and 2.5, where we focus on the shape of the parameter space and the impact of regularization factors, respectively.

2.2 Regularization Factors

In this section we present four common factors in various applications of deep learning that affect regularization, which are also common in NMT: batch size, learning rate, dropout, and gradient clipping, to which we later refer in Section 2.5 and the entire Part I.

2.2.1 Batch Size in Minibatch Gradient Descent

Optimizers for machine learning typically compute each weight update based on an expected value of the cost function, using only a subset of the terms of the full cost function. Instead of evaluating the model on every example in the dataset, these expectations are computed by randomly sampling a small number of examples, which are then averaged. Minibatch gradient descent (GD) uses more than one sample but not the whole training set, stochastic GD uses a single sample at a time, and deterministic (or batch) GD uses the entire training set, processed simultaneously in a single batch.

The effects that batch learning can have on training are presented in Section 2.4 through a theory on the shape of the parameter space, and in Section 2.5 we explain its role as a regularization factor specifically regarding NMT. In the field of computer vision observations have been made on the effect of batch learning. For instance, in a model trained for image classification on ImageNet Goyal et al. (2017), large batch sizes cause a loss of accuracy in the trained model, although this can be addressed by linearly scaling the learning rate as a function of the minibatch size. Other adaptive strategies have been adopted, but by varying the batch size dynamically, instead of the learning rate (Devarakonda et al., 2017). Golmant et al. (2018) observed that across many tasks, domains, and architectures in computer vision, increasing the batch size beyond a certain point may not benefit the quality of the model. Increasing the batch size, in some occasions, does not result in less training time until convergence, and this batch size limit tends to be well below the standard capacity of common GPUs. Formal ways of optimizing batch size have also been proposed, for instance by measuring gradient noise (McCandlish et al., 2018; Smith et al., 2017).

2.2.2 Learning Rate

The learning rate is a positive scalar that controls how much the weights are updated. A commonly used dynamic learning schedule is 'noam' (Vaswani et al., 2017, Eq. 3). During the initial steps, known as warmup, the learning rate increases linearly from zero, reaching its highest value at the last warmup step w. Afterwards, it decays proportionally to the inverse square root of the step number s. At each step, this value is multiplied by a factor based on the output size of the embedding layer d_{model} (512 in Transformer-Base). Moreover, in some frameworks such as OpenNMT-py, an additional scaling factor (sf) is included. The learning rate lr at each step s is defined as follows:

$$lr(s) = sf \cdot d_{model}^{-0.5} \cdot \min\left(s^{-0.5},\ s \cdot w^{-1.5}\right) \tag{2.1}$$

Many different constant and dynamic learning rates are used, sometimes with or without initial warmup steps. Some dynamic learning rates can be cyclical, instead of decaying, or adaptive. For instance, Baydin et al. (2018) propose a dynamically-updated learning rate that leverages the model's gradient for continuous updates. They show that their method reduces notably the importance of selecting an optimal initial learning rate. The learning rate is a regularization factor, because it can alter the optimization of the model during training, as we develop in Sections 2.4 and 2.5.

2.2.3 Dropout

Dropout (Srivastava et al., 2014) is a regularization factor consisting of a masking noise: a probability that a unit is turned off during training. It is usually applied on the output of each hidden layer, including the output of the attention layers, but not on the embedding layers, so no loss of input or output data occurs. This encourages each hidden unit to perform well regardless of other units, discouraging co-adaptation of units (Goodfellow et al., 2016, Chapter 7.12).

Although dropout is usually applied randomly, Guided Dropout (Keshari et al., 2019) is a variation where individual units are dropped based on their importance. In particular, very active units are dropped more often, so that less active units are less likely to remain co-adapted. This importance value is a learned parameter, per-unit, based on comparing a unit with others from its own layer.

R-Drop (Liang et al., 2021) is a training strategy to force the network's output to be more consistent when normal dropout is applied. This is achieved by introducing a regularization term to standard negative log-likelihood loss that measures the bidirectional Kullback-Leibler divergence between two different runs of the same input with dropout.

AutoDropout (Pham and Le, 2021) sets different adaptive dropout patterns per layer based on a trained controller model, which learns from the target model's validation performance.

2.2.4 Gradient Clipping

Gradient clipping (Mikolov, 2012) consists of renormalizing the gradient g to a threshold v if the gradient exceeds the threshold. We consider it another factor for regularization, since limiting the norm of the gradient places a constraint on the effective capacity of the model. The norm of the gradient is clipped so that if $||g|| > v$, then $g \leftarrow gv/||g||$ (same direction but bounded norm). Therefore, the smaller the value of v, the more aggressively the gradients are clipped (Goodfellow et al., 2016, Chapter 10.11.1). The larger the difference between gradient g and threshold v, the more different the weight update is with the clipped gradients than it would be normally, and therefore less accurate to the training data of the step.

2.3 Neural Machine Translation and Language Models

In the previous section we introduced key concepts in machine learning, theoretically important for our research. In this section we present specific concepts on NLP directly tied to our research, in order to explain the main techniques used in low-resource NMT. First, we introduce the architecture used in most NMT and in this thesis, the Transformer, as well as the state of the art regarding Language Models, which we use in Part III for text generation. Then, we describe the main techniques used specifically in low-resource NMT: data augmentation, transfer learning, and multitask training.

2.3.1 Transformer Models

In the early years of NMT, Recurrent Neural Networks (RNNs) were the most common used architectures, although others like Convolutional Neural Networks (CNNs) were used as well. RNNs were used largely due to their ability to model long-term dependencies instead of more limited context windows. More specifically, Long Short-Term Memory networks (LSTMs) (Sutskever et al., 2014) became popular, as well as Gated Recurrent Unit networks (Cho et al., 2014). A particularly significant improvement on long-term dependency modeling occurred with the introduction of attention (Bahdanau et al., 2015). Although RNNs initially underperformed when trained with small amounts of data (Koehn and Knowles, 2017), it has been shown that with careful tuning they can perform well in a low-resource setting (Sennrich and Zhang, 2019).

The Transformer model (Vaswani et al., 2017) shows that attention by itself is enough, without recurrence being necessary, in order to achieve state-of-the-art performance on translation. Transformer-based networks have became the norm in other areas of natural language processing. Long-term dependencies are also modeled better, since the Transformer does not process the input data sequentially, but as a whole, and instead introduces a positional embedding to represent sequential information, which avoids the vanishing gradient problem that plagued recurrent networks. Transformer-based systems have been shown to be able to train with very small amounts of data, although careful hyper-parameter tuning is necessary

(Araabi and Monz, 2020). Although variations of the original Transformer have been proposed, such as for tackling multiple or large documents, like Longformer (Beltagy et al., 2020) or Hierarchical Transformer (Liu and Lapata, 2019), the standard Transformer (Vaswani et al., 2017) remains the most common architecture in use to date.

Decoding is usually performed using beam search (Graves, 2012; Sutskever et al., 2014), where the predicted sentence is generated token-by-token, left-to-right, by maximizing the conditional probability while keeping a number of candidates at each step with the highest probability (this beam size is usually five). The final translation is the candidate (or beam) with the highest log-probability normalized by its size.

2.3.2 Language Models

Any model that assigns or defines a probability over a sequence of tokens (like words) is a language model (Jurafsky and Martin 2023, p.31; Goodfellow et al. 2016, p.456). Specifically, a neural language model is a neural network that learns an output probability distribution over a vocabulary from an input sequence of tokens.

In recent years, a shift has occurred where extensive quantities of unlabeled data are now essential for effective training. This data is incorporated into the network through the utilization of simple tasks. Stand-alone language models can be useful, for instance as text generators, as shown in the Section 2.6. However, they can be useful also for various transfer learning approaches, in particular for initializing the weights of NMT systems. In Table 2.1 we present a selection of recent LMs that are particularly relevant to this thesis, either due to their use for text generation in Part III, or due to their multitask training being relevant to Part II.

2.3.3 Tokenization

Input sequences for NMT are not split into individual words before being fed into the neural system. This is done to avoid dealing with an excessively large vocabulary, as even a small corpus can contain tens of thousands of different word types. Splitting the sequences into words would lead to encountering out-of-vocabulary words more frequently. Instead, the standard method to address these issues is to tokenize the data into subwords: units of variable length, including entire words, fragments of words, and characters. Various algorithms have been used successfully, such as BPE (Sennrich et al., 2016b), WordPiece (Wu et al., 2016), or Unigram (Kudo and Richardson, 2018).

These methods all rely on a pre-tokenization step: the raw text is split into tokens (which usually are words, with possible exceptions like "Mr."), which is typically done with a rule-based model. Then, a subword vocabulary is assembled by combining individual characters from the raw text. Additionally, when using a subword vocabulary to tokenize raw text, removing some subword units has been shown to make the model representations more robust (Provilkov et al., 2020), as well as serving as a form of data augmentation, by performing

Model	Description
BERT (Devlin et al., 2019)	BERT consists of an encoder-only Transformer trained on a primary masking task, where tokens of the input sequence are substituted by a [MASK] token (12%) or replaced by another (1.5%), and a secondary next-sentence prediction task. Some variations have also been introduced such as RoBERTa (Liu et al., 2019).
MASS (Song et al., 2019)	MASS uses a sequence-to-sequence model to reconstruct masked input sequences, training both the encoder and decoder during pre-training, both of them being later successfully used when fine-tuning on downstream tasks.
GPT, GPT-2, GPT-3 (Radford et al., 2018, 2019; Brown et al., 2020)	The models of the GPT family are decoder-only Transformers trained on a causal language modeling task (predicting the following token given a sequence), which means that the model does not take advantage of the bidirectionality of systems like BERT, but improves on standard left-to-right English text generation. The three GPT models largely consist of a similar training regime but increased amount of model parameters and training data.
Chat-GPT by OpenAI[2]	Chat-GPT is a chatbot application by OpenAI, released in late 2022, consisting of a language model (GPT-3) fine-tuned through reinforcement learning with human feedback on a downstream task. Chat-GPT has made an enormous impact in mainstream media, artistic environments, and even on more specialized research and technical areas, such as programming.
CTRL (Keskar et al., 2019)	CTRL introduces tags (control codes) in the training data that denote domains (such as Wikipedia, Books, Horror, etc.) and tasks (question and answering, translation) among others. CTRL is trained on English, German, French and Spanish, allowing for user prompts in either language, and with combinations at inference time between language prompts or control codes not seen in the training data (zero shot). The authors also present a new sampling strategy to achieve near-greedy decoding, but without the known problem of repetitive output, by explicitly excluding from the probability calculation the scores from previously generated tokens.
XLM (Conneau and Lample, 2019)	XLM introduces a parallel translation language modeling task, in which source and target were concatenated in the input, with both being masked, encouraging the model to learn an alignment between the source and target in order to help with the unmasking process.
BART (Lewis et al., 2020)	BART uses a standard encoder-decoder Transformer, expanding the token masking task to also sentence permutation, document rotation, token deletion, and text infilling, and removed the next-sentence prediction task from BERT. This was also extended to mBART, a massive multilingual system (25 languages) which included low-resource languages (Liu et al., 2020b).
T5 (Raffel et al., 2020)	T5 uses a standard encoder-decoder Transformer, and extended the kinds of tasks used for pre-training to others that would otherwise require changes in the network's architecture, such as coreference resolution or sentiment analysis, by presenting them all as text-to-text tasks.

Table 2.1: A selection of LMs from recent years, relevant to this thesis.

several rounds and then combining the results.

SentencePiece (Kudo and Richardson, 2018) is an exception since it forgoes pre-tokenization, which is useful for languages with no whitespaces, or when a trained pre-tokenizer is not available. Although other forms of subword tokenization have been studied, such as byte-based (Costa-jussà et al., 2017), character-based (Libovický et al., 2022), or Huffman-coding-based (Wolleb et al., 2023), they are not as common as the above ones.

Although Transformer-based architectures have also been adapted for tokenization into smaller units, such as bytes (Xue et al., 2022) or characters (Tay et al., 2021), the original Transformer with subword tokenization remains the state of the art.

2.3.4 Data Augmentation

The addition of synthetic data and forms of data augmentation can also be considered a type of regularization (Goodfellow et al., 2016, page 253).

Khayrallah and Koehn (2018) compare how different artificial noises in the data degrade performance for NMT, with a recurrent neural network. They show that for some sources of noise, even a small amount of noise may significantly lower scores. This shows that the regularization benefits provided by noise in the data need to be carefully managed not to produce negative results. Paraphrasing can be used as a data augmentation technique with an authentic parallel corpus by simply training a source→target system, and generating translations of the source side. These translations are paired with the authentic target to assemble a paraphrasing dataset. Khayrallah et al. (2020) showed that this generation can be improved with respect to standard beam-search, greedy, or sampling decoding, by trying to approximate the full space of possible translations by sampling a paraphrase of the reference sentence from a paraphraser model and training the MT model to predict the paraphraser's distribution over possible tokens.

Back-translation is another form of data augmentation, particularly common in low-resource NMT. It consists in automatically translating monolingual data from the target language, in order to create a synthetic parallel corpus which can be used for training (Sennrich et al., 2016a). Edunov et al. (2018) showed that the benefits of back-translated data depend on the decoding algorithms used to generate it, and that beam search is not the best-performing option unless the amount of data to back-translate is small. This, however, can be mitigated by differentiating authentic and synthetic data with tags (Caswell et al., 2019). This process can also be performed iteratively, as shown by Hoang et al. (2018), with either the same model generating initial back-translated data, improving its performance, and re-generating the data, or by training a new model for each round of back-translation, which improves the quality of the synthetic data. Iterative back-translation for low-resource translation has been shown to offer diminishing returns after approximately two rounds (Haddow et al., 2022).

Morphology-based tasks, such as re-inflection after lemmatization, can also be used in addition to the translation task to improve performance on some languages (Dhar et al., 2022). A simple copying task may also be used as an auxiliary task, sometimes even achieving comparable results with back-translation, albeit at a much smaller expense (Burlot and Yvon, 2018).

2.3.5 Transfer Learning for Low-Resource and Unsupervised NMT

Transfer learning, as well as data augmentation, is a common technique in low-resource NMT. The lack of training data can be mitigated by leveraging models that were trained with more easily available data. This means training a model with data from a different domain than the target one, or, the case in this thesis, from other pairs of languages, and then fine-tuning (by using the already trained weights) into the target languages. More specifically, transfer learning consists in training a model on a high-resource pair (parent) that initializes a model trained on a lower-resource one (child). Initially, Zoph et al. (2016) kept the same target language between parent and child. Kocmi and Bojar (2018), however, showed that the identity or relatedness of the target languages is not essential, and that all of the weights of the child systems can be initialized with those of the parent model without changing the training routine.

Models that have been trained without child-specific subword vocabulary can also be used for transfer learning. Kocmi and Bojar (2020) showed that transforming the parent vocabulary into the child vocabulary can significantly improve training time when fine-tuning. This is done by keeping the shared tokens between the two vocabularies, and substituting the tokens in the parent vocabulary if they do not appear in the child vocabulary with the most common tokens in the child vocabulary.

Artetxe et al. (2020b) proposed that the learned weights from a parent model can be leveraged for fine-tuning by freezing them, and training the model on a child dataset (with the same masked language model objective) with a new embedding layer.

More recently, Gogoulou et al. (2022) showed that a direct transfer where the entire model is fine-tuned on child data can still be successfully done even if no child vocabulary is present in pre-training, simply by switching the parent and child vocabularies in the model, under the assumption that the ordering by token frequency of both vocabularies will mean that the ith child token is initialized with the ith parent token, which may play a similar role.

When large monolingual corpora are available, fully unsupervised NMT can be achieved by using masked language modeling, denoising, or translation language modeling (Lample et al., 2017, 2018; Conneau and Lample, 2019). This results in cross-lingual language models (Conneau and Lample, 2019), which can further be trained on back-translated data. Such systems perform best when jointly trained on very large monolingual datasets and when a small amount of parallel data is available (Song et al., 2019; Liu et al., 2020b). Baziotis et al. (2021) compare various pre-training objectives for unsupervised translation: masking, shuffling, word-replacement based on context. In (semi-) supervised NMT, varying the pre-training objective leads to surprisingly small differences in the fine-tuned performance, unsupervised NMT is much more sensitive to it. When starting from pre-trained models the difference shrinks even further. In both cases, shuffle produces the worst results.

2.3.6 Multitasking, Adaptive Scheduling and Curriculum Learning

Multitasking (Caruana, 1993) on low-resource NMT consists in training on additional tasks, other than the target translation task. These additional tasks can use other kinds of data, such as monolingual data or parallel data between other languages. The training between the different additional tasks and the target translation task is performed by the same model, which distinguishes multitasking from transfer learning more generally.

Multitasking can provide an improvement during training, because training in one of the auxiliary tasks may help with the main task, and these auxiliary tasks can provide regularization so that the model does not overfit to the main task.

With a standard single-encoder and single-decoder Transformer-based model, it is possible to train on several tasks at the same time by tagging each of them with a unique tag, usually prepended to each sample. Alternatively, however, it may be beneficial to introduce them in the training regime following some schedule. Curriculum learning (Bengio et al., 2009) is a type of schedule in which the order of tasks is determined based on their complexity.

Multitasking has been proven to benefit many NLP tasks (Raffel et al., 2020). Similarly to data augmentation, multitasking can improve generalization by pooling the examples of several tasks. These examples can be seen as soft constraints imposed on the parameters (Goodfellow et al., 2016, page 258).

Additional languages (multilinguality) can also serve as multitasking. A common approach for low-resource translation is to develop massive multilingual models, that translate many-to-many languages, leveraging better-resourced pairs. Aharoni et al. (2019) translate up to 100 languages with one model, and observe improvements in low-resource languages (see Section 7.2 for more recent work on multilingual and adaptive NMT.).

In Section 6.2 we present additional related work, specifically on multitask and adaptive NMT, and curriculum learning. In Section 7.2 we discuss relevant work, specifically on multilingual NMT.

2.4 Topography of the Weight Space

The generalization gap, defined as the difference between the training and the testing error (see Section 2.1.3), has been explained using the shape of the loss landscape. This shape (or topography) can be defined as the variation rate of the cost function around a given point of the parameter space. For instance, smaller variations of the cost function imply a flatter landscape, and vice-versa. In this section we will review studies that argue that models that are optimized in flatter regions of the parameter space tend to generalize better. We will use this hypothesis to explain our results in Chapter 3, and we will further study it in Chapter 4. More specifically, it is argued that less accurate gradients give models a higher probability of finding these flatter regions, because these regions require less precision to be described.

Since regularization factors affect the accuracy of the gradient with respect to the model's training data, these factors may have an effect on the topography of the loss landscape of the trained model. For instance, Goodfellow et al. (2016, Chapter 8.1.3) explain that models trained with smaller batch sizes tend to optimize into low-precision regions because they use noisier gradient estimates than when training with larger batch sizes.

If we want to train our models so that their weights are set in flat regions, we may think of two possible ways, which we will present in the following two subsections: the first option is to approach it indirectly, making use of theoretical considerations. For instance, if optimizing into flat regions requires less precision than optimizing into sharp points, a reduction of the gradient accuracy by lowering the batch size will increase the likelihood for the model's weights to set into flatter regions. Alternatively, we may approach it directly, by measuring during training the neighborhood around a point in the parameter space, computing its flatness, and updating the model accordingly. Since this direct approach would mean to evaluate an infinite amount of points in order to measure the flatness around a central point, this second approach must rely on some kind of estimation of the true parameter space.

2.4.1 Indirectly Training into Flat Regions

Hochreiter and Schmidhuber (1994) propose an algorithm (Flat Minimum Search) for training low-complexity networks to find flatter regions, which they define as a connected region where weights have small losses. Flat minima, then, would correspond to simple models and low expected overfitting. This approach means that models whose parameters are optimal within a flat region do generalize better because they are lower-complexity than models optimized around sharper regions (Schmidhuber, 2015).

Keskar et al. (2016) study fully connected networks and CNNs on computer vision tasks. They show that training with small batch sizes tends to lead to systems that generalize better than when training with larger ones. They explain this by their more noisy gradient estimation, which makes them more likely to optimize into flatter regions of the parameter space. In order to simulate the parameter space they compare linear and curved path methods from Goodfellow et al. (2015), consisting in evaluating the loss of a model $\hat{\theta}$ for various values of α between two true points θ_0 and θ_1 as: $\hat{\theta} = (1-\alpha)\theta_0 + \alpha\theta_1$. They then propose a measure for sharpness to approximate the sensitivity of the training function, by looking at the highest training loss in a small neighborhood around a point. They also show that small batch sizes allow for an exploration phase, where their sharpness and training loss are similar to those of larger batch sizes, but as training continues they guide optimization into flatter regions.

Following the hypothesis that noisier gradients improve the chance of a model to optimize into flatter regions, which also affects computer vision tasks, Smith and Le (2017) and Smith et al. (2017) propose a gradient noise scale to measure how learning rate should be adjusted to the batch size, on image data. They estimate the average gradient noise g for each batch as $g = \epsilon\,(N/B - 1) \approx \epsilon N/B$ where ϵ is the learning rate, N the size of the training set, and B the batch size, assuming that $N \gg B$. This shows that "increasing the batch size and decaying the learning rate are quantitatively equivalent" (Smith et al., 2017, Sec. 1).

Hoffer et al. (2017) explore the reason for better generalization when training with smaller batch sizes on computer vision with the same networks as Keskar et al. (2016). They show that during the warmup of the learning rate the amount of distance that weights move grows rapidly (logarithmically) with the number of updates, and explain the generalization gap by the difference between the number of steps that small and large batch sizes result in. They show that adapting the regime by modifying the learning rate may bridge the gap depending on the batch size.

Jastrzębski et al. (2018) note that the proportionality of batch size and learning rate is crucial: higher ratios lead to flatter regions, which generalize better. This observation has been noted elsewhere too, although whether the relation between batch size and learning rate is linear, squared, or otherwise, has not been conclusively determined (Krizhevsky, 2014; Hoffer et al., 2017; Popel and Bojar, 2018).

Stochastic-Weight Averaging (SWA) (Izmailov et al., 2018) consists in optimizing into flatter minima by averaging weights, which in practice supports the previously mentioned hypothesis of these points requiring less precision to optimize than sharper ones. This, together with a cyclical learning rate on computer vision tasks, allows for greater exploration of flatter basins. This has recently been successfully used also for knowledge distillation on pre-trained LMs (Lu et al., 2022).

Wortsman et al. (2021) outperform SWA by learning diverse, high-accuracy subspaces consisting of lines, curves, and simplexes, which are later ensembled. This diversity is achieved with the addition of a regularization term between subspaces, whose objective is to increase weight distance by keeping their cosine similarity at 0.

2.4.2 Directly Training into Flat Regions

Li et al. (2017) propose a visualization of the weight space with CNNs on computer vision, and show that skip (residual) connections promote flat minimizers of the train and test loss functions. Additionally to the previously mentioned method of linear path, they propose a filter-wise normalization to mitigate the scale invariance problem between network weights. Dinh et al. (2017), however, provide evidence against the flat minimizer hypothesis by showing that the weights that define a flat minimizer can be transformed to land in a sharp point, while still defining an equivalent model, i.e. a model such that it produces the same loss for each sample of the training set as the competing, flat-optimized model. They try three competing definitions of flatness and show that equivalent models corresponding to arbitrarily sharper minima can always be built. Chaudhari et al. (2017) add a regularization term to stochastic gradient descent in order to maximize an estimation of the depth and flatness of the neighborhood around a minimizer. This, however, has a noticeable computational cost, essentially creating a nested loop to compute the gradient of the local entropy for each true step. Dziugaite and Roy (2018) verify that this method works works by optimizing the Bayesian prior, developing PAC-Bayes bounds for data-dependent priors obtained via some differentially private mechanism. Sankar et al. (2021) propose measuring the landscape per layer, to check how it correlates with the overall space, by computing each layer's Hessian, which they show to be similar to the entire network's Hessian. With this, they present a regularizer that mainly affects the middle layers, although yielding modest improvements.

Sharpness-Aware Minimization (SAM) (Foret et al., 2020) introduces a measure of sharpness to the standard loss minimization, obtaining good results in computer vision tasks. This is done by computing at each step an adversarial loss of the model θ as $\hat{\epsilon} = \rho \frac{\nabla L(\theta)}{||\nabla L(\theta)||_2}$, where ρ is a hyperparameter to measure the neighborhood radius. This adversarial loss is then added as a regularization term to the normal loss. SAM therefore requires two forward-backward passes per update, although the adversarial loss can be computed with micro-batches, noticeably reducing training time. Du et al. (2022) made SAM more efficient, reducing its overhead from twice the training time to only an additional 0.4, through an approximation of the sharpness measure by stochastically sampling a set of weights in each step, and using only a subset of data particularly sensitive to sharpness in order to minimize the SAM loss. Recently, an application of SAM to natural language has been made by Bahri et al. (2022), on question-answering and language understanding, with particularly good improvements on low-resource settings, by fine-tuning T5 (Raffel et al., 2020) and mT5 (Xue et al., 2021), with only an additional 0.25 compute time. They note that the neighborhood radius value, ρ, requires careful tuning. Kaddour et al. (2022) compare SWA and SAM on many tasks, in particular question-answering and natural language understanding. They show their key difference, SWA being implicitly biased towards flat minima, and SAM explicitly approximating it, resulted in SAM optimizing more on the limits of flatter regions, closer to sharper areas than SWA.

As we can see, in recent years there has been ample research on the effects that the topography of the parameter space has on the generalization abilities of a model, and promising results have been obtained, both through indirect and direct means. However, only a small percentage of these studies have consisted of NLP tasks, and no direct study has been made in relation with NMT, let alone low-resource NMT.

2.5 Effects of Regularization in NMT

Empirical studies have been made regarding the use of regularization on low-resource NMT, which can be understood together under the flat minimizer hypothesis presented in Section 2.4. We now present the most relevant empirical studies on regularization factors specifically for NMT.

Popel and Bojar (2018) report that BLEU scores increase with batch size in a Transformer-based NMT system, although with diminishing returns, and recommend setting a large batch size. They observe moderate changes across a large range of learning rates, and find thresholds beyond which training was much slower or diverged. They make similar observations for warmup steps, concluding that the search space for learning rate and warmup steps was wide. Their experiments are performed on large datasets, leaving their questions open for low-resource settings.

Sennrich and Zhang (2019) experimented with a recurrent neural network in a low-resource setting and found that smaller batch sizes were beneficial, along with other regularization factors. They experimented with two batch sizes of 4,000 and 1,000 tokens, and observed improvements with the latter of 0.30 and 0.04 BLEU points on data sets with 5k and 160k sentence pairs, respectively. As the regularization factors were not disentangled, it is difficult to predict their individual influence. They applied learning rate changes together with additional forms of regularization, so it is not possible to judge the individual improvement provided by learning rate. Similarly, they also applied dropout at the same time as other factors, so it's not known how much each improves performance. It is difficult to predict from these results what the optimal batch size is for Transformer-based NMT.

Araabi and Monz (2020) conducted an empirical investigation, utilizing the Transformer model, to explore the effects of varying batch sizes and other hyperparameters on different subsets of a single dataset, similar to the study conducted by Sennrich and Zhang (2019). Their findings indicate that decreasing the default batch size of 4,096 did not yield any discernible benefits. However, when working with corpora consisting of 80k and 165k sentence pairs, they observed improved performance by increasing the batch size to 8,192 and 12,288, respectively. In order to explain their differences in results with respect to Sennrich and Zhang (2019), who observed benefits from reducing batch size even with comparable or larger training sets, they argue that Transformer models require larger batch sizes compared to recurrent networks. It is important to note, however, that a potential flaw in this comparison arises due to the grid search methodology employed. Specifically, Araabi and Monz (2020) tested various batch sizes towards the end of their hyperparameter optimization process, by which point they had already introduced additional regularization to the model, primarily through increased word, activation, and layer dropouts, in contrast to Sennrich and Zhang (2019). Considering the possibility that smaller batch sizes yield improved generalization by introducing noise to the gradient, the absence of benefits observed in reducing batch size aligns with the aforementioned theory. Consequently, it becomes evident that the experimental schedule employed by Araabi and Monz (2020) is inadequate for accurately assessing the impact of batch sizes on low-resource Neural Machine Translation (NMT). It can be argued that if the order of hyperparameter search were reversed, i.e., if batch size were explored earlier and subsequent regularization techniques were introduced later, the authors might have observed benefits in reducing batch size, followed by diminished benefits with the introduction of various dropout techniques. This rationale also elucidates why Araabi and Monz (2020) did not observe any benefits in altering the learning rate or warmup steps, the final hyperparameters to be tested, considering the model was already subject to sufficient regularization.

Xu et al. (2020b) computed gradients while accumulating minibatches, and observed that increasing batch size stabilizes gradient direction up to a certain point, after which it starts to fluctuate. Performing an update when the direction of the gradient starts to fluctuate allows them to dynamically adjust batch sizes while training. In their experiments with large training sets (4.5M and 36M sentence pairs), their average batch size was around 26k on two GPUs, and never lower than 7k. Their observations on the gradient direction as more minibatches are accumulated are consistent with the findings of Popel and Bojar (2018), who see diminishing returns when increasing batch size.

Studies on the optimization and effects of regularization factors thus remain scarce. Many previous studies optimize parameters in sequence. While this strategy is certainly a faster approach to optimization, it does not shed full light on each factor in isolation, as we do below in Sections 3.3 to 3.6, or in combination, as we study in Sections 3.7 and 3.8.

2.6 Poetry Generation

Regularization is important for low-resource NMT, as we have shown. In other fields of NLP, such as Natural Language Generation (NLG), it can be also necessary, particularly when operating in a low-resource setting. NLG makes use of LMs, usually by sampling from the LM's output distribution, and feeding the context plus the newly generated token back to the model. State-of-the-art language models, however, as we have seen in Section 2.3.2, are nearly universally trained with vast amounts of data. When much less data is available, for example due to a specific domain, LMs benefit from training in a more similar way to what we have shown for low-resource NMT, for instance by making use of multitasking or data augmentation techniques (Hangya et al., 2022; Popescu-Belis et al., 2023). In this section we will review some related work on one of these specific domains within NLG, poetry generation, an area which we will study in Part III.

Poetry generation is a specific task within the field of NLG, but also a topic of interest in the field of computational creativity (McGregor et al., 2016). Before the advent of deep neural LMs, various combinations of rule-based approaches and n-gram LMs have been tried. For instance, in their broad discussion of computational creativity, McGregor et al. (2016) define a poem generation model, which uses word vectors to infer semantic relations, followed by a phonological model, an n-gram LM, and a sentiment model. Their basis for poem generation are topics from Switchboard conversations annotated with sentiment scores.

Large neural LMs have brought high expectations regarding their capacities to generate structured texts such as poems, and clearly improved fluency for high-resource languages. Poem generation with GPT-2 (Radford et al., 2019) was discussed, for instance, by Branwen and Presser (2019) in a blog entry shortly after the model was made available. More recently, ChatGPT has tremendously improved the quality and relevance of generated text. However, anecdotal evidence shows that it cannot reliably generate a given rhyming pattern. Deep neural networks have been used several times for poem generation, mostly in English or in Chinese (see Section 9.2). The main challenge is to learn and use the constraints of poetic style from the data, and most studies focus on rhythm and rhymes. Additionally, the lack of training data due to the specificity of the domain poses serious problems to the training of LMs for poetry generation.

2.7 Conclusion

In this chapter we have presented background for our thesis, both theoretical and applied. We have introduced general concepts from machine learning, as well as more specific concepts regarding regularization factors and NMT, which will be relevant for Parts I and II. We have also presented the state of the art regarding LMs, which will be relevant for Part III, on poetry generation. Similarly, we have provided background for a treatment of multitasking and scheduling, specific to Part II, as well as a treatment of the theory of the flat minima and the effects of regularization on NMT, with a specific focus on low-resource NMT, which will be addressed in Part I. Finally, we have introduced a specific area of NLG, poetry generation, which we address in Part III. We will now present our work on the effect of regularization factors on low-resource NMT, explaining the empirical results under the theoretical umbrella of the flat minima hypothesis.

Part I

Regularization of Low-Resource NMT Into Flatter Regions of the Parameter Space

3 Regularization Factors and Flat Minima[1]

In this chapter, we explore the roles and interactions of the hyper-parameters governing regularization, and propose a range of values applicable to low-resource neural machine translation. We show that default or recommended values for high-resource settings are not optimal for low-resource ones, and that more aggressive regularization is needed when resources are scarce, in proportion to their scarcity. We explain our observations by the generalization abilities of sharp vs. flat basins in the loss landscape of a neural network. Results for four regularization factors corroborate our claim: batch size, learning rate, dropout rate, and gradient clipping. Moreover, we show that optimal results are obtained when using several of these factors, and that our findings generalize across datasets of different sizes and languages.

3.1 Introduction

The training of neural machine translation (NMT) models is governed by many hyper-parameters, which play a central role in the performances of the trained models, especially their generalization abilities. While most of the NMT frameworks recommend default values for the hyper-parameters, when it comes to low-resource settings, fewer guidelines are available.

This chapter systematically explores the roles and interactions of a subset of hyper-parameters in low-resource NMT settings, namely those acting as *regularization factors*. Regularizers do not fall under a single theoretical definition: Goodfellow et al. (2016, page 224) view them as a collection of methods "intended to reduce generalization error but not training error." We present here a unified perspective on several regularizers which act upon the estimation of the gradients during back-propagation. We focus on constraining effective representation, so we do not consider changes to the architecture of the network. Additionally, we condense the search on various parameters associated with dropout (such as activation dropout, attention dropout, etc.) into general dropout (i.e., each unit of a hidden layer is dropped with probability p). Due to experimental constraints, we consider the most common hyper-parameters used,

[1] This work was published in Atrio and Popescu-Belis (2021, 2022)

although additional ones like label smoothing could also be considered.

When computing the value of the loss of a model (which in this chapter is calculated with cross-entropy, as previously presented formally in Section 1.1.2) while varying its weights, the aspect of this function, both on train and test data, will have flatter and sharper regions. The flat minima hypothesis (Hochreiter and Schmidhuber, 1994; Keskar et al., 2016) (introduced above in Section 2.4 and illustrated in Figure 1.3) states that models whose weights settle in flatter regions of this loss landscape will have better abilities for generalization to unseen data than those on sharper regions. Additionally, since a small variation of the weights on sharper regions results in a large change of the loss, the less precise an optimizer is, the less likely it will be to settle on sharper regions and more likely on flatter ones. We measure the generalization ability of our models computing BLEU and chrF (see Section 3.2). Specifically, in this chapter, we defend three claims:

1. *NMT models benefit from more aggressive regularization when the amount of training data is small.* We demonstrate this for four different regularizers: batch size, learning rate, dropout, and gradient clipping. We compare the default regularization hyperparameters of the OpenNMT-py framework for mid-to-high resources – comparable to those of the original Transformer (Vaswani et al., 2017) – with the ones we optimized for a low-resource setting (Sections 3.3-3.6)
2. *The combination of different regularization sources is preferable over their individual use.* When used together, an amount of regularization from each of the four factors under study outperforms the use of any single one alone, and the best scores are robust with respect to the variation of each factor (Section 3.7).
3. *Regularization factors optimized on one low-resource dataset are beneficial for low-resource datasets in other languages, and benefit from more aggressive regularization as the amount of training data decreases.* We demonstrate this by comparing our default and optimized settings on data samples of varying sizes from our main corpus and four additional low-size datasets (Section 3.8).

3.2 Data and Systems

We train our NMT systems with the Upper Sorbian (HSB) to German (DE) training data of the WMT 2020 Low-Resource Translation Task (Fraser, 2020). We also use the HSB-DE development and test sets provided by the WMT 2020 and 2021 Low-Resource Translation Tasks (Fraser, 2020; Libovický and Fraser, 2021), each consisting of 2k sentences. As length-based filtering does not show significant differences, we do not filter our data. Additionally, in Section 3.8, we train systems for translation from Galician (GL), Slovenian (SL), and Slovak (SK) into English (EN), using tokenized and cleaned transcriptions of TED Talks, with the provided dev and test sets (Qi et al., 2018).[2] Finally, we train a larger German to English system using

[2] https://github.com/neulab/word-embeddings-for-nmt

Dataset	Src-tgt	Train	Words (tgt)	Dev	Test
WMT20 Low-res	HSB-DE	60k	823k	2000	2000
=	=	40k	550k	=	=
=	=	20k	273k	=	=
NewsComm. v13	DE-EN	120k	3M	1500	1500
TED Talks	SK-EN	61k	1.3M	2271	2445
=	SL-EN	19k	443k	1068	1251
=	GL-EN	10k	214k	682	1007

Table 3.1: Numbers of lines of the original corpora used in our experiments. Sections 3.3-3.7 use only the first dataset. We do not use monolingual or back-translated data, and train our tokenizers using only each parallel corpus.

120k lines from News Commentary v13 (Bojar et al., 2018), and sample 1,500 lines each for development and testing. Table 3.1 presents these resources.

Tokenization into subwords is done with a Unigram LM model (Kudo, 2018) from SentencePiece.[3] For each language pair we build a shared vocabulary of 10k subwords (based on preliminary tests) using only the parallel corpus, with character coverage of 0.98, *nbest* of 1 and *alpha* of 0.

We use the Transformer-Base architecture (Vaswani et al., 2017) implemented in OpenNMT-py (Klein et al., 2017, 2020).[4] Our default setting of hyper-parameters is the one recommended by OpenNMT-py[5] which is close to the original Transformer (Vaswani et al., 2017). For Adam, $\beta_1 = 0.9$, $\beta_2 = 0.998$ and $\epsilon = 10^{-8}$. The regularization factors appear with relatively low strengths in this setting, as is usual when large datasets are available. The setting includes the 'noam' learning rate schedule with a scaling factor of 2, warmup steps of 8000, dropout rate of 0.1, batch size of 4096, and no gradient clipping.

We train our models for a maximum of 100 hours, although they generally converge earlier. When comparing batch sizes in Section 3.3, it could be argued that epochs might provide a fairer comparison, but we measure real clock time as the most relevant measure for practitioners.

A batch consists of lines (tokenized sentences) that are translated one by one, with a fixed maximum length of 512 tokens for Transformer-Base. Lines are padded if shorter, and filtered out if longer. Following common practice, sentences are sorted by their length in tokens to avoid unnecessary padding. We train all models on two GPUs with 11 GB of memory each (GeForce RTX 1080Ti). Each device processes several batches, depending on the batch size, which are afterwards accumulated and used to update the model. The *effective batch size* and the `batch_size` parameter of OpenNMT-py are two different values: the former

[3]https://github.com/google/sentencepiece
[4]We make public our configuration files at https://github.com/AlexRAtrio/reg-factors.
[5]https://opennmt.net/OpenNMT-py/FAQ.html#how-do-i-use-the-transformer-model

is $G \times A \times$ `batch_size`, where G is the number of GPUs and A the number of accumulated batches, here equal to two.[6] Throughout the chapter, we report the `batch_size` parameter, but the effective batch size is in fact four times larger.

Based on preliminary tests, we generate translations with a beam size of seven, with consecutive ensembles of four checkpoints. For each model we report the highest BLEU score (Papineni et al., 2002) calculated with SacreBLEU (Post, 2018) on detokenized text[7] as well as the chrF score (Popović, 2015). We test the statistical significance of differences in scores at the 95% confidence level using paired bootstrap resampling from SacreBLEU.

3.3 Batch Size

In this section we train models with batch sizes ranging from 500 to 10,000, with all other hyper-parameters set to default. Models with batch sizes of 100 and 250 were also trained, but did not converge. The largest tested batch size is the largest value supported by our GPUs.

Batch size	**train**		**dev**		**test**	
	Xent	Acc.	BLEU	chrF	BLEU	chrF
0.5k	0.02	99.93	50.54*	73.35	43.95^	69.25
1k	0.01	99.94	**52.02**	**74.63**	**44.40^**	**70.02**
3k	0.01	99.96	50.16*	73.38	43.91^	69.16
6k	0.01	99.97	49.66+	73.09	42.55−	68.85
9k	0.01	99.96	49.42+	73.10	42.22−	68.40
10k	0.01	99.97	48.46	72.49	42.19−	68.38

Table 3.2: HSB-DE scores with various batch sizes, all other settings being default ones. Values with the same color or symbol are *not* significantly different. The highest scores are in bold.

The BLEU and chrF scores in Table 3.2 show that lowering the batch size improves quality of NMT, likely due to the regularizing effect of a less accurate gradient, according to our theoretical perspective. In particular, we observe improved results with a batch size smaller than 3,000 (+1.71 BLEU) and an optimal size around 1,000 (+2.21), with scores gradually decreasing as batch size increases. These results are in line with previous observations (Sennrich and Zhang, 2019; Atrio and Popescu-Belis, 2021).

There is no clear correlation between the training accuracy or cross-entropy loss and the generalization capacity, i.e. the scores on the development and test sets. The lower scores of models trained with larger batch sizes are likely not due to overfitting, because the *testing* curves of these models do not show any decrease late in the training. This further supports the claim that better generalization abilities are due to flat minima (Keskar et al., 2016, Section 2.1).

[6] https://forum.opennmt.net/t/epochs-determination/3119

[7] https://github.com/mjpost/sacrebleu with the signature nrefs:1|bs:1000|seed:12345|case:mixed|eff:no |tok:13a|smooth:exp|version:2.0.0.

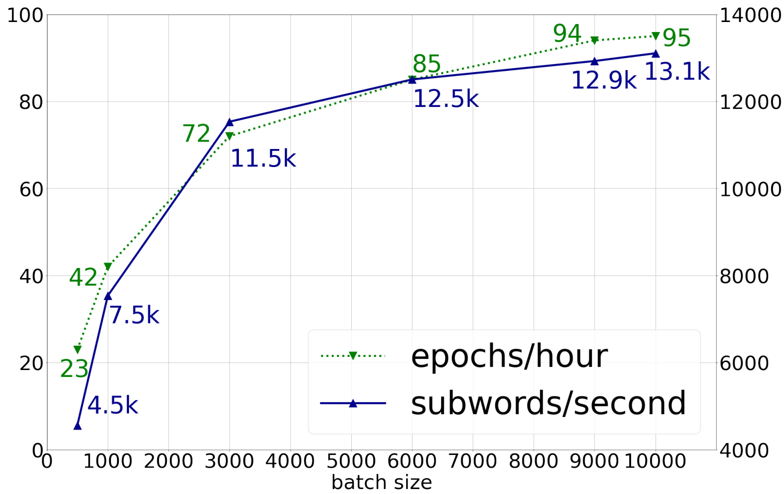

Figure 3.1: Throughput (subwords/second, in blue) and speed (epochs/hour, in green) for the tested batch sizes.

Our results are competitive with the submitted systems from the WMT20 shared task on low-resource NMT for HSB-DE (Fraser, 2020), which used the same parallel data.[8] The baseline BLEU scores of Knowles et al. (2020), Libovický et al. (2020) and Kvapilíková et al. (2020) were respectively 44.1, 43.4, and 38.7 on the test set.

Regularization through smaller batch sizes thus provides visible improvements with respect to the default setting. Larger batch sizes, however, exploit more fully the memory of the GPUs, which enables higher throughput in terms of subwords processed per second, as illustrated in Figure 3.1, although this does not increase linearly: instead, we observe diminishing returns as batch size increases. Still, while a batch size of 10k has the lowest BLEU scores, it nearly doubles the throughput with respect to the highest-scoring batch size (1k). Due to differences in hardware and software, these values are difficult to compare to other studies, but the trends are similar to those observed by Popel and Bojar (2018, Section 4.1).

If the regularization attained with lower batch sizes can also be obtained by using other regularization factors, this would allow the use of larger batch sizes for a more efficient training. Therefore, in the next sections, we will compare a large batch size (10k) and the optimal, regularized one (1k), and verify that none of the other regularization factors that will be optimized have an impact on speed.

3.4 Learning Rate

Previous studies by Smith et al. (2017) and Smith and Le (2017) have shown that the regularization effects of the batch size and of the learning rate may be comparable. In this section,

[8]Some of these systems used in fact larger monolingual HSB, DE and/or CS datasets for training their tokenizers, while we only used 60k lines of parallel HSB-DE text.

we study the role of varying schedules of the learning rate (3.4.1) and the effect of resetting the schedule in mid-training, i.e. suddenly increasing the learning rate before another decrease (3.4.2).

3.4.1 Regularization through Learning Rate

Since all our models have the same dimension of embeddings (d_{model} in Eq. 2.1 above), the only variables influencing the learning rate in the 'noam' schedule are the number of warmup steps and the scaling factor (Vaswani et al., 2017, Eq. 3). We test two different values for the former: 8k (default) and 16k. For the latter, we test even values from 2 (default) to 14. Figure 3.2 displays some tested schedules, including our default one (8k, 2) and the 'noam' original one (4k, 1).

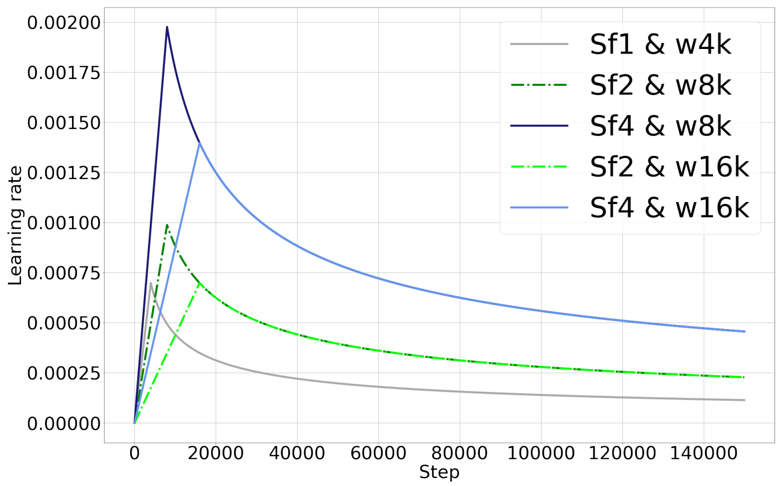

Figure 3.2: 'Noam' learning rate schedules with different scaling factors (*sf*) and numbers of warmup steps (w).

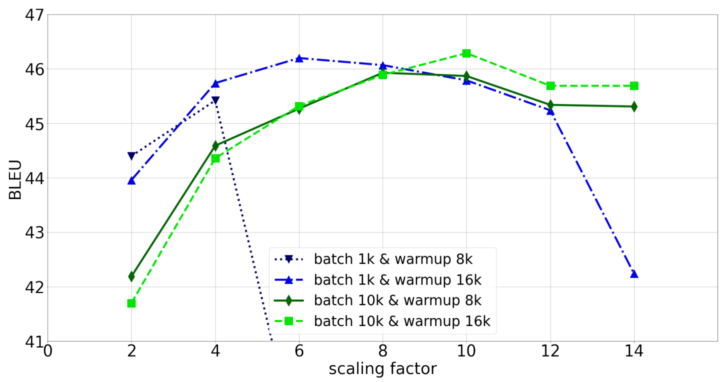

Figure 3.3: BLEU scores on the test set for 10k and 1k batch sizes, 8k and 16k warmup steps, and varying scaling factors (x-axis). The y-axis is cut-off at 41 BLEU to zoom on the extremes of the curves.

The results in Table 3.3 show that both batch sizes reach similar maximal scores (46.20 and 46.29), although with different scaling factors: 6 for a batch size of 1k vs. 10 for a batch size of 10k. The improvement is 1.8 BLEU points for a batch size of 1k, and 4.1 for 10k. As a batch size of 1k is already a strong regularization factor, a smaller value of the learning rate (hence less regularization through this factor) is sufficient, compared to the case of a larger batch size.

Batch size	**Warmup**	**Scaling factor** 2	4	6	8	10	12	14
1k	8k	44.40	**45.42**	38.90	0.65	0.18	0.05	0.60
	16k	43.96	45.74	**46.20***	46.07*	45.79*	45.24*	42.24
10k	8k	42.19	44.59	45.27⋆	**45.93**−	45.87−	45.34⋆	45.31⋆
	16k	41.70	44.36	45.32+	45.89^	**46.29**^	45.69+	45.69+

Table 3.3: BLEU scores on the HSB-DE test set for batch sizes of 1k and 10k and various learning schedules. We denote scores that are *not* significantly different row-wise with the same color or symbol.

The models trained with the larger batch size (10k) are more stable when learning rates increase (larger scaling factors) likely due to more accurate estimates of the gradients (compare lines 1 vs. 3, and 2 vs. 4). Similarly, these models have a higher maximal learning rate beyond which they diverge (compare in Table 3.3 the large difference between lines 1 and 2 with the small difference between lines 3 and 4). This shows the importance of increasing the number of warmup steps as the scaling factor increases, to avoid reaching high maxima of the learning rate (the peaks visible on the schedules in Figure 3.2). Moreover, the regularization provided by other factors (in this case, batch size) needs to be taken into account when increasing the amount of regularization from the learning rate. Finally, as long as the maximal learning rate remains below the values that make a model diverge, the BLEU scores do not change significantly when the scaling factor increases above a certain value, as also observed by Popel and Bojar (2018, 4.6, Fig. 7).

3.4.2 Resetting the Learning Rate during Training

From the perspective of the loss landscape, we hypothesize that introducing more noise into the gradient when the scores have already leveled-off, namely by resetting the learning rate schedule, should increase the probability for the weights to escape the sharp minima basins and avoid falling back into them, which should improve the generalization abilities of the trained model. Since a model trained with a smaller batch size has a higher chance, during the first part of training, to fall into flat minima due to an increased gradient noise (Smith et al., 2017), we expect the larger batch sizes to benefit more from this strategy than the smaller ones.

In Table 3.4 we provide the scores after training for 50 hours (half of their training time); the scores after 100 hours when continuing to train from the 50-hour checkpoint; and the final score after training for 50 hours with a schedule reset at the 50-hour checkpoint. The results corroborate our hypothesis: both batch sizes benefit significantly from the strategy of resetting

Batch size		Hours 50	100 no lr reset	100 reset lr
1k	BLEU	44.25	44.40	**45.85**
	chrF	69.78	70.02	70.84
	Train. Acc.	99.93	99.94	99.84
	Xent	0.02	0.01	0.02
	Δ		*+0.15*	***+1.60***
10k	BLEU	41.60	42.19	**45.25**
	chrF	68.03	68.38	70.57
	Train. Acc.	99.94	99.97	99.92
	Xent	0.01	0.01	0.01
	Δ		*+0.59*	***+3.65***

Table 3.4: BLEU and chrF scores on the HSB-DE test set, training accuracy and cross-entropy on the training set, and change of BLEU scores when continuing training until 100 hours vs. resetting the learning rate at 50h.

the learning rate, and the large batch size more than the smaller one ((+3.65 vs. +1.6 BLEU points). As both models reached their highest BLEU scores before 25 hours, the difference is likely not due to that fact that the first model trained with a larger batch size saw more times the training data thanks to its higher throughput. Furthermore, after increasing the learning rate mid-training, both the loss and training accuracy worsen or remain stable, while BLEU scores improve, likely due to reaching flatter basins, not lower minima.

3.5 Dropout Rate

The dropout of a certain proportion of neurons during training is another frequent source of regularization. As this amounts to removing certain terms from the summation of gradients, its role can also be considered from the perspective of flat vs. sharp minimizers.

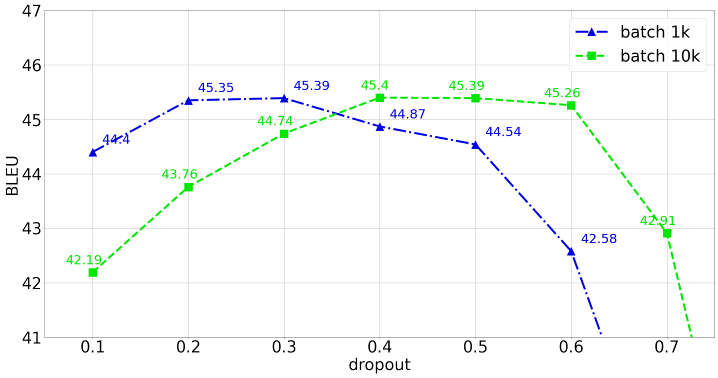

Figure 3.4: Scores on the test set for batch sizes of 10k and 1k. Dropout rates of 0.8 and 0.9 are not shown due to small scores, but they continue the trend.

Batch size	Dropout							
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
1k	44.40*	45.35+	**45.39+**	44.87*	44.54*	42.58	37.69	19.83
10k	42.19	43.76	44.74	**45.40^**	45.39^	45.26^	42.91	35.52

Table 3.5: Dropout scores on the HSB-DE test set for 1k and 10k batch sizes. We denote row-wise lack of significant differences with the same color or symbol. Dropout rates of 0.9 have considerably lower scores.

BLEU scores in Table 3.5 show that the model trained with a larger batch size – hence subject to less regularization – requires a more aggressive dropout of around 0.4–0.6 in order to reach its highest scores, with respect to a model trained with a smaller batch size, which reaches its highest score for 0.2–0.3. This is consistent with our previous findings from Section 3.4.1 and Table 3.3, which also showed that the model subject to less regularization from a factor (larger batch size) required more regularization from another factor in order to reach its highest scores.

3.6 Gradient Clipping

Finally, we experiment with our fourth regularization factor: gradient clipping. Since it directly involves constraining the norm of the gradient, the perspective based on flat vs. sharp basins in the loss landscape also holds for it.

Batch size	Dropout	Gradient Clipping				
		None	20	10	5	2.5
1k	0.1	44.40	44.75	**44.92**	44.74	44.54
10k	0.1	42.19	**42.41**	42.01	42.30	42.20
10k	0.2	43.76	44.15	**44.34**	43.98	43.85
	0.3	44.74	**45.36**	44.72	44.75	44.99
	0.4	45.40	**45.56**	45.30	45.45	45.48
	Scaling factor					
10k	2	42.19	**42.41**	42.01	42.30	42.20
10k	6	45.27	**45.61**	45.24	45.29	45.32
	10	**45.87**	45.45	45.77	45.55	45.35

Table 3.6: BLEU scores on the HSB-DE test set for batch sizes of 10k and 1k on the test set, with a dropout rate of 0.1 (default), for several upper limits of the gradients. Scaling factor rows have default warmup steps of 8k.

As in the previous sections, we compare models trained with batch sizes of 1k and 10k, but observe no statistically significant differences between them when using default values for other hyper-parameters, with BLEU scores shown in Table 3.6 – although values of 10 or 20 are always among the best. This is likely because default settings do not feature enough regularization (i.e., they do not increase enough the gradient's norm) for the gradients to be affected by clipping. For this reason, we perform additional experiments with a batch size of 10k (due to its advantage for speed) with more regularizing dropout values of 0.2, 0.3, and 0.4, and scaling factor of 6 and 10. Regarding the models with increasing dropout rate, we only observe a statistically significant difference between the best and worst results (for dropout of 0.2), the best and two worst results (for 0.3), and no differences at all (for 0.4). Regarding the models with increasing scaling factor, there is no row-wise statistically significant difference. We conclude that gradient clipping only marginally affects training in these settings.

3.7 Combining Regularization Factors

We will now show that a combination of regularization factors can produce higher scores than individual factors used separately, and that the maximal scores are stable when varying the strengths of regularizers around their optimal values. The batch size is fixed at 10k, since this enables a higher training speed than 1k with similar best scores, provided that other regularization factors are used, as shown in Tables 3.3, 3.4 and 3.5.

The number of warmup steps is fixed at 16k since we showed in Section 3.4.1 that this parameter mainly limits the peaks of the learning rate and thus prevents models from diverging early in the training. Our search space for the other regularization factors is shown in Table 3.8.

Factor	**Value**	**Xent**	**Tr. acc.**	**BLEU**	**chrF**	**Δ**
Defaults	-	0.01	99.97	42.19	68.38	-
Batch size	1k	0.02	99.94	44.40	70.02	+2.21
Scaling factor	10	0.01	99.94	45.93	70.74	+3.74
S.f. + warmup steps	10 + 16k	0.01	99.94	46.29	71.22	+4.10
Learning rate reset	50%	0.01	99.92	45.25	70.57	+3.06
Dropout	0.4	0.07	99.46	45.40	71.00	+3.21
Clipping	10	0.01	99.96	42.41	68.43	+0.22
Combination	Table 3.8	0.03	99.78	**47.11**	**71.88**	**+4.92**
+ l.r. reset	-	0.06	99.30	**47.20**	**71.80**	**+5.01**

Table 3.7: HSB-DE scores on the test set when the regularization factors are used either independently (lines 2–6) or in combination (line 7), in the latter case with the optimal values from Table 3.8. The last column shows increases in BLEU scores over the default settings.

We present in Table 3.7 the highest scores achieved using individual regularization factors, along with those from the default setup (first line) and from the combination of factors (last two lines). Regularization factors are already present in the default setup, but at low strengths.

The comparison of scores in Table 3.7 shows that each factor used independently allows the model to outperform the default setting by 2–4 BLEU points. However, the use of a combination of factors achieves the highest score of 47.20 BLEU points (+ 5.01), which is significantly above all others. In the case of resetting the learning rate, although this has a visible effect when used with default parameters, its effect is much smaller when used jointly with other regularization factors, likely because a flat basin is found before the reset. Moreover, the combination of factors results in a higher loss and a lower accuracy on the train set than the default setup or factors used individually, which supports our interpretation of the improvement based on flatter minima.

Gradient clipping	**Scaling factor**	**Dropout** 0.1	0.3	0.5	0.7
None	2	42.19	44.74	45.39	42.91
	6	45.32	46.70	46.22	43.66
	10	46.29	47.06	46.93	43.18
	14	45.69	46.84	47.07	43.61
	18	45.26	46.89	46.67	43.19
5	2	41.39	44.47	45.05	43.48
	6	45.20	46.62	46.70	43.88
	10	45.65	**47.11**	46.76	44.04
	14	45.57	**47.11**	47.06	43.63
	18	44.72	46.59	47.02	42.72

Table 3.8: HSB-DE BLEU scores for a combination of the scaling factor, gradient clipping, and dropout rate, for a batch size of 10k and 16k warmup steps. The highest scores are in bold.

Table 3.8 shows that the best scores reached with increased regularization are quite stable when varying the intensity of the factors. The optimal region of the scaling factor is around 10, with a relatively flat neighborhood, similar to the case when it was optimized individually (Section 3.4). Optimal dropout rates are now around 0.3–0.5, compared to 0.4–0.6 when used individually (Section 3.5). Finally, gradient clipping has only a marginal effect in combination with other factors, presumably because it cannot help to increase the gradients.

3.8 Testing on Additional Corpora

In this section, we confirm our claims using additional low-resource datasets. We consider two smaller samples with 40k and 20k lines from the HSB-DE corpus, as well as parallel datasets for Galician, German, Slovak and Slovenian (see Section 3.2). We do not optimize regularization factors on each dataset, but only use the optimal hyper-parameters found above on HSB-DE with 60k lines.

Table 3.9 demonstrates that these hyper-parameter values bring significant improvements of BLEU and chrF scores over the baseline for all datasets, including four new low-resource datasets (rows 4-7) and two sub-samples of the dataset used until now (rows 2 and 3). When comparing the latter, we find that as the amount of data decreases, the positive effects of our regularization parameters increase, with up to 21% improvement in BLEU scores for the smallest subset. Furthermore, we also observe an increase in the loss over all datasets with the optimized setup, which shows that the reason why their less accurate gradients generalize

Corpus	**Lines**	**Default**				**Optimized**				**Δ**
		Xent	Tr. Acc.	BLEU	chrF	Xent	Tr. Acc.	BLEU	chrF	BLEU
HSB-DE	*60k*	*0.01*	*99.97*	*42.19*	*68.38*	*0.06*	*99.30*	***47.20***	*71.80*	***+5.01***
HSB-DE	40k	0.01	99.98	32.38	60.68	0.03	99.80	**37.63**	65.12	**+5.25**
HSB-DE	20k	0.01	99.98	22.93	51.42	0.02	99.93	**27.84**	56.27	**+4.91**
DE-EN	120k	0.10	98.20	29.94	56.81	0.60	84.71	**35.77**	61.44	**+5.83**
SK-EN	61k	0.02	99.89	25.61	46.42	0.40	89.29	**29.71**	49.67	**+4.01**
SL-EN	19k	0.01	99.93	15.53	34.99	0.09	98.89	**18.43**	37.75	**+2.90**
GL-EN	10k	0.01	99.98	16.00	34.52	0.04	99.69	**19.04**	37.84	**+3.04**

Table 3.9: BLEU scores on test sets of different corpora and subsets of our main HSB-DE corpus (first line), comparing our default setup and our optimized setup as presented in Section 3.7.

better is not due to finding lower but rather flatter minima of loss.

Araabi and Monz (2020) report similar improvements with another optimization of the Transformer for the low-resource datasets GL-EN, SL-EN, and SK-EN, although their scores are not fully comparable with ours, since there are slight differences between their training data sizes and ours. In particular, their improvements are 13.1 to 22.3 for GL-EN, 9.1 to 15.5 for SL-EN, and 24.8 to 29.9 for SK-EN.

3.9 Conclusion

In this chapter, we presented a unified perspective on the role of four regularization factors in low-resource settings: batch size, learning schedule, gradient clipping and dropout rate. The results support our claim that more regularization is beneficial in such settings, with respect to the default values that are recommended for high-resource settings. We first substantiated the claim for each factor taken individually, and then showed that a combination of factors leads to improved scores and is robust when factors vary. Finally, we showed that our findings generalize across different low-resource sizes and languages. Overall, we interpreted the results from the perspective of the loss landscape, and argued that more regularization is beneficial because the noise it introduces in the estimation of gradients leads to finding flatter minima of the loss, which have better generalization abilities.

4 Loss Landscape

4.1 Introduction

In this chapter we study the role of the loss landscape for low-resource NMT. In particular, we study the relation between the area within the loss landscape that a model occupies (whether it is flatter or sharper) and the model's generalization ability. As in Chapter 3, we measure the latter with BLEU (Papineni et al., 2002), and the loss is the negative log-likelihood between the models' predictions and the true data, that the model trains to minimize.

First, we study how well the loss landscape can be measured using linear interpolation, and propose a cost-effective method to estimate the immediate neighborhood around a point. Then, we propose various ways to measure the flatness of this neighborhood. Finally, we propose a solution to integrate an estimation of the loss landscape into the SGD algorithm to improve training of low-resource NMT systems, and perspectives to expand this work by using our method to quantify the relation between flatness and regularization and generalization.

4.2 Estimating the Loss Landscape

In this section we study whether an interpolation method is suitable for measuring the loss landscape around a point in the parameter space (Section 4.2.1). We build on these observations and propose a cost-effective method to estimate the immediate neighborhood of a point in the parameter space without the use of back-propagation (Section 4.2.2).

4.2.1 Accuracy of Interpolation Method

Data and Systems

In this section we use a Transformer-base system from OpenNMT-py, trained on two low resource datasets, in either directions.[1] We train the system on a HSB-DE dataset (60k lines)

[1] This system is presented in Chapter 8.

and a DE-EN dataset (120k lines) and evaluate it on the HSB-DE test data, as presented in Chapters 3 and 8 . The hyper-parameters for our optimized systems are the ones obtained for our best-performing models in Chapter 3. We choose this model as a well-performing system on low-resource NMT datasets, trained with large amounts of regularization.

We use Google's SentencePiece[2] to learn a joint trilingual vocabulary, from the HSB-DE and the DE-EN dataset. We train the tokenizer with *character_coverage*=1. We add language tags in the vocabulary for each target language (`<HSB>, <DE>, <EN>`).

Linear Interpolation between Two Parameter Sets

We define linear interpolation between two points of the weight space θ_{t_1} and θ_{t_3} as $(1-\alpha)\cdot\theta_{t_1}+\alpha\cdot\theta_{t_3}$. We define α as a default to 0.5. We take this strategy from Goodfellow et al. (2015), who use it to plot the training trajectory of a CNN and of fully connected feed-forward networks.

We take a model, trained normally, and for each ordered group of $X = 5$ checkpoints (separated by $Y = 2500$ steps each) θ_{t_1}, θ_{t_2}, θ_{t_3}, we interpolate θ'_{t_2} with θ_{t_1} and θ_{t_3}, and compare the difference in score between θ_{t_2} and θ'_{t_2}. If the difference is small, this means that linear interpolation serves as an accurate approximation to map the loss landscape. Conversely, if the loss landscape is highly irregular, we expect that a simple interpolation of the weights for each layer will not fall in the same region as gradient descent would.

That being said, if the interpolated models score higher, this can be explained as this method allowing the weights to set in flatter regions (since they require less precision to define), which mean more generalizable models. If the scores of the interpolated weights, however, are lower, then the method is not accurate enough to plot the landscape.

From Table 4.1 we can conclude that when training a model normally, between two steps, either (i) the trajectory of the training in the weight space is very straightforward, which means that we can estimate it by using linear interpolation, or (ii) the amount of space around a model that is typically covered within a few steps is very regular, and even if linear interpolation is not retracing the path that the training trajectory took, it stills results in models of comparable quality. We also note that the model shown here is very regularized, both in terms of hyper-parameters and auxiliary data, which might explain why the area around all the trained checkpoints is regular; since a heavily regularized model is more unlikely to optimize into sharper points.

Using Extrapolation to Predict New Weights

We now show how to use an interpolation method to predict the direction for a new set of weights. We present the following experiment: by increasing α above 1, we should be able to

[2] https://github.com/google/sentencepiece

Step	Original	Estimated	Difference
12500	43.1	43.3	0.2
25000	47.0	47.9	0.9
37500	48.2	48.1	-0.1
50000	48.6	48.7	0.1
62500	48.3	48.7	0.4
75000	48.6	48.6	0.0
87500	48.7	48.9	0.2
100000	49.0	49.2	0.2
112500	48.6	49.1	0.5
125000	49.2	49.2	0.0
137500	48.7	49.2	0.5

Table 4.1: Linearly interpolating selected checkpoints of a trained model with previous and next checkpoints, with $\alpha = 0.5$. We note the difference between the BLEU development scores from the estimated weights and the original ones, with positive values indicating better scores with the estimated weights. We can observe very small differences in scores between the actually trained models and the interpolated models from trained models.

predict, or extrapolate, a new set of weights. With this, we consider the following idea: with a hyper-parameter σ, we take initial existing weights θ_{t_1}, θ_{t_2}, and then for every σ steps, we define θ'_{t_3} as the extrapolation between θ_{t_1} and θ_{t_2} with $\alpha > 1$. This means that when $\sigma = 1$, only the first two sets of weights would be actually learned, and when $\sigma = 10$, every tenth set of weights would be predicted, not learned.

In Figure 4.1 we show BLEU scores with $\sigma = 5$ and $\alpha = 1.5$. We observe, particularly at the beginning, that this prediction is noticeably worse, although as weights stabilize by training for longer, the decrease in score is not as significant.

Extrapolation	Model	BLEU	$\theta'_{t_3} - \theta_{t_3}$
-	θ_{t_1} (7500 steps)	36.8	-
-	θ_{t_2} (10000 steps)	40.9	-
-	θ_{t_3} (12500 steps)	**43.1**	-
	extrapolate($\theta_{t_1}, \theta_{t_2}, \alpha = 1.15$)	41.0	-2.1
	extrapolate($\theta_{t_1}, \theta_{t_2}, \alpha = 1.25$)	40.9	-2.2
θ'_{t_3}	extrapolate($\theta_{t_1}, \theta_{t_1}, \alpha = 1.50$)	40.3	-2.8
	extrapolate($\theta_{t_1}, \theta_{t_2}, \alpha = 1.75$)	36.4	-6.7
	extrapolate($\theta_{t_1}, \theta_{t_2}, \alpha = 2.00$)	3.7	-39.4

Table 4.2: Predicting the equivalent of θ_{t_3} (θ'_{t_3}) for two training steps, by extrapolating θ_{t_1} and θ_{t_2}. Using σ of 5, we take the first and last data point in Figure 4.1, and together with the value of α of 1.5 already shown in the figure, we also extrapolate with various values. The checkpoint being extrapolated is found early during the training (at 12500 steps out of 137500).

We now take the first checkpoint (for $\sigma = 5$) and the last one for our trained model, and

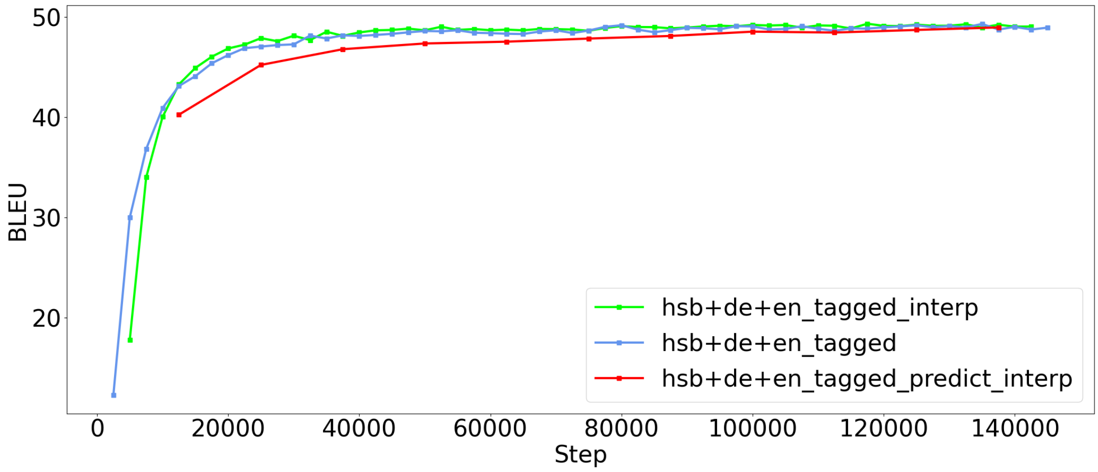

Figure 4.1: Scores of extrapolated weights. The normally-trained model (`hsb+de+en_tagged`) overperforms the points predicted with extrapolation (`hsb+de+en_tagged_predict_interp`, presented in Section 4.2.1), but when using interpolation to find middle points between the trajectory of normal training, we obtain slightly higher scores (`hsb+de+en_tagged_interp`, presented in Section 4.2.1).

Extrapolation	**Model**	**BLEU**	$\theta'_{t_3} - \theta_{t_3}$
-	θ_{t_1} (132500 steps)	49.0	-
-	θ_{t_2} (135000 steps)	**49.3**	-
-	θ_{t_3} (137500 steps)	48.7	-
θ'_{t_3}	extrapolate($\theta_{t_1}, \theta_{t_2}, \alpha = 1.15$)	49.2	0.5
	extrapolate($\theta_{t_1}, \theta_{t_2}, \alpha = 1.25$)	49.2	0.5
	extrapolate($\theta_{t_1}, \theta_{t_2}, \alpha = 1.50$)	49.0	0.3
	extrapolate($\theta_{t_1}, \theta_{t_2}, \alpha = 1.75$)	38.4	-10.3
	extrapolate($\theta_{t_1}, \theta_{t_2}, \alpha = 2.00$)	34.9	-13.8

Table 4.3: As in Table 4.2, but extrapolating one of the last checkpoints during training until model convergence.

compare the trained checkpoint θ_{t_3} with our estimated model θ'_{t_3}, from θ_{t_1} and θ_{t_2}, with various values of α. We observe differences depending on the point in the training that the model is. At the beginning of training (Table 4.2), we cannot obtain significantly better results by predicting a step with linear interpolation than by learning the weights (which is a reasonable result). Additionally, quality quickly degrades the farthest away we move from our last set of trained weights, θ_{t_2}.

However, when the model has been well trained (Table 4.3), it seems that the space around it in the loss landscape is different. We see how smaller values of α result in an increase of half a BLEU point. This can be explained by the model setting on sharper and lower regions of the landscape, in order to minimize training loss: the less precise method of interpolation improves the results, since it is more likely to set on flatter areas (due to it being less precise), which in turn generalizes better to the test set.

These results show that interpolating from existing sets of weights can provide further regularization to the model, as well as additional exploratory abilities, all with a lower computing time than training another step. However, if we compare the results from Tables 4.2 and 4.3, we observe that deciding at which points during training to apply this method is a non trivial question.

It remains to be seen whether curved interpolation ($sin(\frac{\alpha\pi}{2})\theta_t + cos(\frac{\alpha\pi}{2})\theta_{t1}$) might model with more precision the loss landscape, as well as whether considering various previous models, instead of just two as we have done in this section, might improve as well the modeling of the training trajectory, or training a model to interpolated a set of weights, from an ordered sequence of previous ones.

4.2.2 Flatness of the Loss Landscape

In this section we estimate the flatness of the loss landscape around a point in the parameter space, that is, a model θ.

Data

For our experiments in this section, we train an NMT system on the very low-resource Galician (GL) - English (EN) dataset from the multilingual TED corpus (Qi et al., 2018), with sizes of training, development and test sets shown in Table 4.4.

	Train		**Dev**	**Test**
	Lines	Words (tgt)	Lines	Lines
GL-EN	10017	214k	682	1007

Table 4.4: Data used in this chapter.

We filter lines longer than 100 tokens, resulting in 9970 lines. We train a Unigram LM model (Kudo, 2018) from SentencePiece[3] with a character coverage of 1.0, *nbest* of 1 and *alpha* of 0, and build a vocabulary with tokens occurring at least twice, which results in a shared vocabulary of 9271 tokens.

NMT System

For this section, we train a Transformer model (Vaswani et al., 2017) in our own Pytorch-based implementation[4], and in order to reduce training time, we consider a Transformer with only 3 layers in the encoder and another 3 in the decoder. Additionally, our model consists of 8

[3] github.com/google/sentencepiece

[4] Based on github.com/bentrevett/pytorch-seq2seq.

attention heads, hidden layer of 256 units, and FFN of 512 units. We use the Adam optimizer (Kingma and Ba, 2014), with beta values of 0.9 and 0.998, dropout rate of 0.1, and a batch size of 90 lines. We use cross-entropy loss. This results in ~12M parameters, compared to the ~50M that Transformer-base models tends to have.[5] We train on a single GPU (Nvidia GTX 1080 or RTX 2080) with no gradient accumulation. Our implementation is simpler than the one used in Section 4.2.1 to better monitor the training algorithm and to be able to modify easily the training method subsequently.

To assess the impact on translation quality of our simple implementation and reduced architecture, we train four different baseline models and compare their scores. In particular, we compare values of the constant learning rate 0.0001, 0.0005, and 0.00001, dropout rates 0.1 and 0.3, and varying training times. We conclude by training a model with a constant learning rate of 0.0005, which after 50 epochs (in about thirty minutes) obtains development and test scores of 15.38 and 13.16 BLEU points, respectively, with a loss value on the train set of 0.346.

Translations are generated with greedy decoding and no model ensembling. For each model we consider the scores of the checkpoint with the highest development BLEU score (Papineni et al., 2002), calculated with SacreBLEU (Post, 2018) on detokenized text.[6]

For comparison, with a very optimized model, Araabi and Monz (2020) improve scores on the same dataset from a Transformer-base baseline of 13 BLEU points on the test set to 22, which are comparable to what we also obtain in Chapter 3 on the same dataset. Furthermore, we note that compared with a Transformer-base model trained with OpenNMT-py (Klein et al., 2017, 2020), our systems train in less than half the amount of time, reaching similar loss values on the train set and development BLEU scores, with only a decrease of ~1.5 BLEU points.

Estimating the Neighborhood

Given a model θ, we estimate its neighborhood in the loss landscape in the following manner. We define $N = 10$ random directions from a point by iterating over all the weight matrices of the point N times, and for each one performing a random perturbation of the weights. This results in models $\theta_{rand_1}, \ldots, \theta_{rand_N}$. Then, we estimate the area between θ and each of the random directions by the linear interpolation method described in Section 4.2.1.

In particular, we fix `n_alphas`$= 25$, and interpolate `n_alphas` between θ and each of the random directions. We therefore estimate the neighborhood around θ as a matrix Φ of shape $(N,$ `n_alphas`$)$, such that (i, j) is $(1-\alpha_j)\cdot\theta + \alpha_j \cdot \theta_{rand_i}$, where α_j is a given value of α. In particular, we sample `n_alphas` values of α evenly from the interval [0,0.1]. This results in a maximal euclidean distance between θ and θ_{rand_i} of ~2000. Since we use a constant learning rate, the average euclidean distance between each epoch during normal training is ~225. This

[5]Transformer-Base models have 6 encoder/decoder layers, 8 attention heads, hidden layer of 512 units, and FFN of 2,048 units.

[6]github.com/mjpost/sacrebleu with the signature nrefs:1|bs:1000|seed:12345|case:mixed|eff:no |tok:13a|smooth:exp|version:2.0.0.

heuristic choice means that our notion of neighborhood comprises the area that would be traveled in approximately eight epochs during normal training. Further empirical testing on different orders of magnitude of the euclidean distance is required to ascertain up to what distance the interpolation method can reliably estimate the immediate neighborhood of a model.

For each set of weights in Φ we build the model (with the same architecture as θ) and compute its training loss, obtaining a matrix of train losses Ψ: the loss neighborhood of θ. We empirically observe near identical values when measuring the training loss over the entire dataset and over a random collection of 10 batches of 90 lines each (~10% of the dataset), at only a third of the training time. Overall, computing Φ and train loss values for each of its models only takes 3.5 minutes on a RTX 2080.

Flatness Measures

We now compare the following methods in order to measure the flatness of the loss landscape of the loss values in Ψ.

1. Standard deviation (STD): $\sqrt{\frac{1}{M-1}\sum_{i,j}(\Psi_{i,j}-\overline{\Psi})^2}$, where $\overline{\Psi}$ is the average of the loss neighborhood Ψ, and M is the number of elements of Ψ.
2. Peak-average ratio (PAR): $\Psi_{max}-\overline{\Psi}$, where Ψ_{max} is the highest value of Ψ, and $\overline{\Psi}$ is the average of Ψ.
3. Shannon entropy (ENT): $-\sum_{i,j} p(\Psi_{i,j})\log p(\Psi_{i,j})$, where $p(\Psi_{i,j})$ is the result of normalizing Ψ as $\frac{\Psi_{i,j}}{\sum_{p,q}\Psi_{p,q}}$.
4. Variance drop (VAR): we propose to iteratively compute the variance of the flattened Ψ, sorted from highest train loss to lowest, and at each time removing the highest value. Each variance is divided by the median of Ψ, and the flatness of Ψ then is the mean of the normalized variances.

When computing Φ, we also compute the Euclidean distance between θ and each model θ' in Φ as follows: $\sum_{l=1}^{L}\sum_{i,j}||\theta^l_{i,j}-\theta'^l_{i,j}||$, where L is the number of layers in θ. We use its inverse to create weights, so that models that are farthest away are less important to the calculation of the flatness of Φ, since the linear interpolation method is less reliable the farthest away it is. This results in a total of eight scores, since we compute each of the metrics with and without weights.

Results

We present here a selection of the scores obtained from the metrics presented above, computed for 20 epochs, evenly sampled from the total of 50, of the system described in Section 4.2.2.

In particular, in Table 4.5 we show the total Pearson correlation of each metric (with and without weights) with respect to BLEU score on the development set. We can see that our proposed measure of Variance drop, unweighted, is useful to predict generalization score on our system,which is consistent with our observations on Section 4.2.1. Similarly, although with less strength, other metrics also show that can be considered to give important information on the generalization ability of a model.

Metric	ρ
Weighted VAR	-0.9085
Train Loss	-0.9046
Weighted ENT	-0.7439
ENT	-0.6149
PAR	-0.2031
STD	-0.1568
Weighted PAR	-0.1289
Weighted STD	-0.0818
VAR	0.6200
Epoch	0.7414

Table 4.5: Pearson correlation of the flatness metrics with BLEU score on the development set, measured each epoch during training for a total of 50 epochs. This indicates that our proposed Variance drop metric (VAR) is useful to predict model generalization, and most neighborhood flatness metrics provide as similar information as the train loss of the model itself.

4.3 Training Weights into Flat Minima

In this section we present our strategy to integrate an estimation of the loss landscape into the SGD algorithm to improve training of low-resource NMT systems.

Given a training set $\mathscr{D} := \cup_{i=1}^{n}\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, training steps s, learning rate schedule $\eta_1, \ldots, \eta_s$, batch size $|\mathscr{B}|$ and loss function L, we add the following hyper-parameters: a number of training branches $\kappa > 0$ a number of local random directions $N > 0$, and the amount of interpolated models `n_alphas`> 0, which results in a neighborhood of size $N \times$ `n_alphas`. We propose the following, for each step t on regular intervals:

1. Sample a mini-batch $\mathscr{B}$ from $\mathscr{D}$.
2. Draw κ models $\theta_t^{(1)}, \ldots, \theta_t^{(\kappa)}$ from θ_t, by training on microbatches with high regularization. The objective of this is to store simultaneously various candidates for $\theta_t + 1$, branching from θ_t, that are far enough apart in the weight space that they can be considered to inhabit different neighborhoods.
3. In order to cost-efficiently estimate the neighborhood of each of the κ branches, for each of them N copies are created of of $\theta_t^{(i)}$, resulting in $\theta_{t_{rand_1}}^{(i)}, \ldots, \theta_{t_{rand_N}}^{(i)}$, as explained

in Section 4.2.2.

4. For each direction $\theta^{(i)}_{t_{rand_i}}$, we compute `n_alphas` models by $(1-\alpha)\cdot\theta^{(i)}_{t} + \alpha\cdot\theta^{(i)}_{t_{rand_N}}$. We sample `n_alphas` values of α evenly from the interval $[0,0.1]$, as detailed in Section 4.2.2. This results in the estimated neighborhood of $\theta^{(i)}_{t}$, a matrix of models of size $N \times$ `n_alphas`.

5. For each estimated neighborhood, training losses are computed with a small amount of batches, and their flatness is measured with a metric presented in Section 4.2.2.

6. θ_{t+1} is defined as the model in $\theta^{(i)}_{t_{rand_1}},\ldots,\theta^{(i)}_{t_{rand_N}}$ with the lowest value of the mean of its loss and the flatness of its neighborhood.

4.4 Conclusion and Perspectives

In this chapter we have presented evidence that shows that linear interpolation is enough to estimate the loss landscape of a model. We have also provided evidence that at different points in training, a well-regularized model settles on different-looking regions of the landscape, which affects the precision of the interpolation method. We have proposed a cost-effective method to estimate the immediate neighborhood of a model in the loss landscape, as well as empirical demonstration of several metrics to measure its flatness for a normally-trained model. Finally, we have also proposed a solution to integrate this estimation into standard training, which at some computational cost, would help particularly low-resource NMT models to settle into flatter regions of the loss landscape, hence improving their generalization abilities.

We conclude by observing that empirical testing of the loss-landscape-aware SGD algorithm will require extensive future experiments. Additionally, more theoretical results stemming from our neighborhood-estimation method should also be studied experimentally, namely to find out whether models trained with different amounts of regularization settle on regions of the loss landscape with different flatness values, and whether a connection can be quantified between flatness and generalization ability of a model, by comparing how models settled on regions with different flatness values perform on a set of unseen datasets, ranked on similarity to the training set used to train the models.

Part II

Regularization of Low-Resource NMT Systems Through Multitasking

5 A Simplified Training Pipeline for Low-Resource and Unsupervised MT[1]

Training neural MT systems for low-resource language pairs or in unsupervised settings (i.e. with no parallel data) often involves a large number of auxiliary systems. These may include parent systems trained on higher-resource pairs and used for initializing the parameters of child systems, multilingual systems for neighboring languages, and several stages of systems trained on pseudo-parallel data obtained through back-translation. In this chapter, we propose a simplified pipeline, which we compare to the best submissions to the WMT 2021 Shared Task on Unsupervised MT and Very Low Resource Supervised MT. Our pipeline only needs two parents, two children, one round of back-translation for low-resource directions and two for unsupervised ones and obtains better or similar scores when compared to more complex alternatives.

After showing in this chapter that various techniques commonly used in low-resource NMT are often used superfluously, in the next two chapters of Part II we will study whether a more sophisticated approach to multitasking (Chapter 6) and multilingual training (Chapter 7) can prove beneficial in a low-resource setting.

5.1 Introduction

Several known techniques enable the design of neural MT systems with little or no parallel data for the source and target languages. Among them are the initialization with a parent model trained on parallel data from related languages (Zoph et al., 2016; Kocmi and Bojar, 2018) and repeated cycles of back-translation of monolingual data that create pseudo-parallel corpora used for training (Sennrich et al., 2016a; Hoang et al., 2018). When designing a very low-resource or unsupervised system, many practitioners rightfully consider as a guideline the best-performing systems found in several shared tasks, such as WMT Shared Task on

[1]This work was performed in collaboration with Alexis Allemann and Ljiljana Dolamic, and published in Atrio et al. (2023). Alexis Allemann worked on the training of the models.

Unsupervised MT and Very Low Resource Supervised MT (Fraser, 2020; Libovický and Fraser, 2021a; Weller-Di Marco and Fraser, 2022), where teams compete in order to achieve the highest translation quality among them. While these systems typically do obtain very high scores, in this chapter we show that the pipelines of the highest-scoring systems in this task may be unnecessarily complex, and they can be substantially simplified while still achieving comparable results. Our research sheds light on designing more efficient NMT systems, considering the balance of cost (in training time and complexity) and benefit that commonly used techniques have, when used in conjunction with each other.

To solve this shared task, high-resource parent models have been leveraged to initialize child models for low-resource languages, which in turn have been used to warm-start the training for unsupervised directions. However, the submissions to the above-mentioned shared task typically developed several dozen models, with numerous parent/child models in both directions as well as increasingly better models trained on several rounds of back-translated data. These models were finally ensembled for best results.

For the 2021 edition of the task, the unsupervised language pair was Lower Sorbian–German (DSB-DE), with parallel data only available for testing, while the low-resource pair was Upper Sorbian–German (HSB-DE). A large amount of German–Czech (DE-CS) parallel or monolingual data is available to train parent models, due to the similarity of Sorbian dialects to Czech. Moreover, given the similarity of the two Sorbian dialects, child low-resource models can become parents of "grandchild" systems for the unsupervised task. As a result, these systems are quite complex, which raises the question: up to which point can these architectures be simplified with virtually no loss of performance?

Our study answers this question by presenting a simpler pipeline than the ones submitted to the shared task, which reaches superior or comparable scores to the ones from the highest-scoring teams. In our pipeline, we apply the same selection and filtering of data as the best-performing team for comparability. We train high-resource parent models on authentic parallel data in two directions (CS↔DE), and then use them to initialize child low-resource models (HSB↔DE). We improve these systems with one round of back-translated monolingual data, and finally use them to initialize systems and to produce back-translated data for the unsupervised pair (DSB↔DE). More specifically, our simplifications are the following[2]:

1. only training from one initialization per parent-child-grandchild;
2. no multitasking and no multilingual models;
3. length-based filtering of back-translated data instead of language model-based one;
4. no monolingual data and only moderate amount of authentic parallel data for high-resource parent models;
5. a single round of back-translation for low-resource directions and two for unsupervised

[2]We make public the configuration files that create these systems within the OpenNMT-py framework (github.com/AlexRAtrio/simplified-pipeline).

directions;

6. same subword vocabulary for all translation directions;
7. moderately-sized Transformer-Base instead of Big;
8. unique set of values for hyper-parameters such as learning rate and label smoothing.

5.2 Analysis of Submissions to the WMT21 Shared Task

Six teams competed for the highest scores in the low-resource Upper Sorbian / German and the unsupervised Lower Sorbian / German translation tasks at the WMT 2021 Shared Tasks on Unsupervised MT and Very Low Resource Supervised MT (Libovický and Fraser, 2021a). The datasets used in the tasks are presented in Section 5.4.1 below. The organizers scored the submissions using automatic metrics (BLEU, chrF, BERTScore) over held-out test sets. NRC-CNRC (Knowles and Larkin, 2021) and LMU (Libovický and Fraser, 2021b) achieved some of the highest scores in both tasks. Other competitive scores were achieved by CL_RUG (Edman et al., 2021) and NoahNMT (Zhang et al., 2021c), followed at some distance by our own contribution labeled as IICT-Yverdon (Atrio et al., 2021). Since no team participated in both tasks, and NoahNMT used a particularly complex pipeline with very large amounts of training data and a pre-trained BERT encoder, we decided to work towards the simplification of the NRC-CNRC and LMU 2021 pipelines.

The NRC-CNRC submission (Knowles and Larkin, 2021) experimented with various numbers of BPE merges (Sennrich et al., 2016b) for different translation directions and for generating synthetic data for training. Their final vocabularies contain 25k and 20k subwords for the supervised and unsupervised models, respectively. They built the BPE tokenizer from upscaled HSB, CS and DE data, but without DSB. The architecture is based on Transformer-Base (Vaswani et al., 2017), with frequent ensembling throughout the pipeline. They use Moore-Lewis filtering (Moore and Lewis, 2010) of back-translated sentences. They train parent CS↔DE models on the entire parallel CS-DE data in Table 5.1, with BPE-dropout (Provilkov et al., 2020). From them, they initialize child HSB↔DE models, which are further fine-tuned into grandchildren DSB↔DE.

The final HSB→DE system from NRC-CNRC is an ensemble of eight different models. Six of them are children and grandchildren of CS-DE models, and two are multilingual CS-DE and HSB-DE models (with no transfer learning). Among the other six, there are different values for hyper-parameters like learning rate or label smoothing. After training with various filtering strategies for back-translated sentences, Moore-Lewis filtering was found to perform best, although differences are generally smaller than 1 BLEU point. Some models are fine-tuned only with back-translations, or only authentic data, or both. For DE→HSB translation, the translation is generated with an ensemble of seven systems. The final NRC-CNRC submission to the DSB→DE unsupervised task is an ensemble of two grandchild systems trained with different back-translated corpora, and for DE→DSB it is an ensemble of four grandchildren, with different rounds of back-translation, different learning rates, and at least one different

CS-DE parent model.

The LMU submission (Libovický and Fraser, 2021b) starts with a BPE tokenizer with 16k merges, on the entire HSB, DE, CS and DSB data. Parent Transformer-Base CS↔DE models are trained on the entire CS-DE parallel data, which is filtered by length and sentences that are not Czech or German. To this authentic data, they add 20M lines of monolingual CS and DE respectively for back-translation, which they use to train another set of parent models with Transformer-Big, sampling and tagged back-translation. Child HSB→DE and DE→HSB models (also Transformer-Big) are trained from CS-DE parents, first on authentic parallel data. Then, they are used to iteratively back-translate 15M lines of DE and the entire HSB monolingual data for four rounds, with a new model initialization for each round. To obtain DSB→DE and DE→DSB grandchildren systems, iterative back-translation is performed for eight rounds, initialized from the respective HSB/DE Transformer-Big child systems.

A similar shared task was again organized at WMT 2022, including HSB↔DE and DSB↔DE translation (Weller-Di Marco and Fraser, 2022). Additional parallel HSB-DE data was provided, increasing the total to about 0.5 million lines, which likely increased scores for the low-resource supervised tasks HSB↔DE. Moreover, new tracks were introduced so that all directions between HSB, DSB, and DE were studied in both low-resource and unsupervised settings.

Four teams participated in the low-resource supervised tasks, and three in the unsupervised ones. In most tasks, HuaweiTSC (Shapiro et al., 2022) achieved by far the highest scores, thanks to a deep 35-layer encoder, 6-layer decoder Transformer (Wei et al., 2021) and a parent multilingual model trained on vast amounts of data (including 55M lines of DE-CS, 66M lines of DE-PL, and 20M of monolingual DE). In addition to the techniques we study in this chapter, Shapiro et al. (2022) used regularized dropout (Liang et al., 2021) to improve consistency while training. Their setup thus also consisted of numerous and expensive training steps, just as the NRC-CNRC and LMU systems to which we compare our proposal.

5.3 Proposed Pipeline

We propose a simplified training pipeline represented in Figure 5.1, which reaches comparable or better results than the above systems. The pipeline is minimal, in the sense that only eight systems are trained for HSB↔DE and DSB↔DE translation, including parent systems for initialization. We show that one round of back-translation for low-resource directions and two for unsupervised ones are sufficient. In comparison with the numerous rounds and checkpoints of the NRC-CNRC and LMU systems, our pipeline is an order of magnitude smaller.

We start by training from scratch parent models DE→$\text{CS}_{\text{parent}}$ and CS→$\text{DE}_{\text{parent}}$ on authentic parallel data. From their best-performing checkpoint, we respectively initialize DE→$\text{HSB}_{\text{authentic}}$ and HSB→$\text{DE}_{\text{authentic}}$ models, which we train only on authentic parallel data. We then use

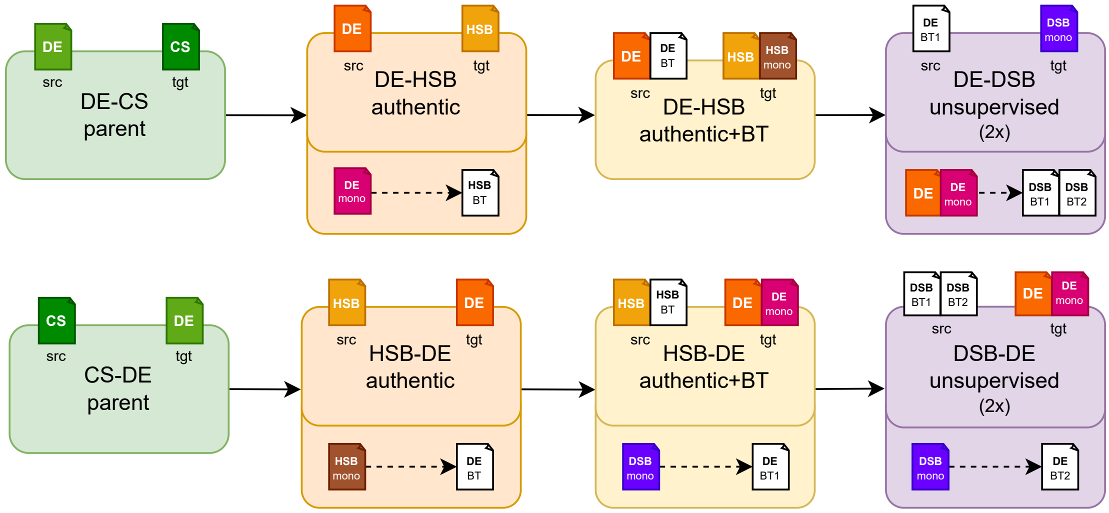

Figure 5.1: Pipeline of implemented systems. Solid arrows represent the parent systems used, and dashed arrows represent creation of synthetic data through back-translation. The datasets in color are those presented in Table 5.1. The datasets in white, to the right of dashed lines, are the back-translations (BT) generated by our systems. The unsupervised models are trained with two rounds of back-translation. Figure reproduced from Atrio et al. (2023).

their best-performing checkpoints to generate synthetic parallel data (back-translations) by translating monolingual target data (resulting in synthetic datasets HSB_{BT}-DE_{mono} and DE_{BT}-HSB_{mono}). We initialize from the best-performing checkpoints of the previous systems new models $\text{DE}\rightarrow\text{HSB}_{\text{authentic+BT}}$ and $\text{HSB}\rightarrow\text{DE}_{\text{authentic+BT}}$ which we train on upscaled authentic parallel data and back-translated data.

Finally, with the best-performing checkpoint of system $\text{HSB}\rightarrow\text{DE}_{\text{authentic+BT}}$, we perform back-translation of monolingual DSB data (resulting in DE_{BT1}-DSB_{mono}), and train with this first round of synthetic parallel data the unsupervised $\text{DE}\rightarrow\text{DSB}_{\text{unsupervised}(a)}$ model. We use this system for the first round of back-translation in the opposite direction, of the DE part of the HSB-DE authentic data and monolingual DE (resulting in DSB_{BT1}-DE and DSB_{BT2}-DE_{mono}) into DSB, on which we train the unsupervised $\text{DSB}\rightarrow\text{DE}_{\text{unsupervised}(a)}$ model. We then use this system for the second round of back-translation of monolingual DSB data and train another unsupervised $\text{DE}\rightarrow\text{DSB}_{\text{unsupervised}(b)}$ model, and with it we perform a second round of back-translation of monolingual DE to train a final unsupervised $\text{DSB}\rightarrow\text{DSB}_{\text{unsupervised}(b)}$ model.

Language	Name of dataset	Original		After filtering	
		sentences	words	sentences	words
DE-CS	DGT v8	4,924	88,616	4,894	85,423
	Europarl v8	569	13,337	569	13,337
	JW300	1,052	16,342	1,039	16,180
	News Commentary v16	204	4,494	197	4,445
	OpenSubtitles	16,380	115,950	16,358	115,729
	WMT-News	20	438	20	437
DE-HSB	WMT 2020 Train	60	741	60	741
	WMT 2021 Train	88	1,266	88	1,266
HSB$_{\text{mono}}$	WMT20 Sorbian Institute	340	5,951	340	5,951
	WMT20 Web	134	2,137	133	2,137
	WMT20 Witaj	222	3,236	222	3,234
DSB$_{\text{mono}}$	WMT21 Monolingual	145	2,381	145	2,381
DE$_{\text{mono}}$	WMT21 News Crawl 19	1,500	41,907	1,500	41,907

Table 5.1: Monolingual and parallel corpora with their languages and numbers of lines (sentences) and words, before and after filtering, in thousands. For the bilingual corpora we only give the values of the DE side.

5.4 Data, Preprocessing and Systems

5.4.1 Corpora

The datasets we use are listed in Table 5.1, and the identifiers correspond to those in Figure 5.1. They encode the language and index number for authentic parallel DE-CS, authentic parallel DE-HSB, and monolingual HSB, DSB, and DE. For the CS↔DE parent models we use parallel data from DGT (Tiedemann, 2012; Steinberger et al., 2012), Europarl (Koehn, 2005), JW300 (Agić and Vulić, 2019), OpenSubtitles (Lison and Tiedemann, 2016), News Commentary, and WMT-News.[3] Our HSB↔DE models use datasets from the 2020 edition of the task, with monolingual HSB data from three sources: (a) the Sorbian Institute provided a mix of high- and medium-quality HSB data; (b) the Witaj Sprachzentrum provided high-quality HSB data; (c) the Web data consists of web-scraped noisier HSB data gathered by the Center for Information and Language Processing at LMU Munich (Fraser, 2020). Our DSB↔DE models use only the monolingual Lower Sorbian (DSB) dataset from the 2021 shared task.

To evaluate our systems, we use the 'Newstest2019-csde' as a test set for our CS↔DE models. For our HSB↔DE and DSB↔DE models we use the 'devel' set from the WMT20 task during development, and 'devel_test' for final evaluations. Since the official scores of the task are calculated on an undisclosed subset of the blind test set, we cannot compare our results with the final official ones. We will thus compare them with the scores on 'devel_test' reported by each team in their articles. Our two evaluation metrics are the same as in the shared

[3] statmt.org/wmt20/translation-task.html

task. We use the SacreBLEU library (Post, 2018) to compute BLEU (Papineni et al., 2002).[4] We also use BERTScore[5] (Zhang et al., 2019a), with the XLM-RoBERTa-Large model (Conneau et al., 2020) for translations into German, as provided with the BERTScore toolkit. We test the statistical significance of differences in scores at the 95% confidence level using paired bootstrap resampling from SacreBLEU.

5.4.2 Data Filtering

For comparison purposes, we follow closely the data preparation procedure of the NRC-CNRC team (Knowles and Larkin, 2021). We first clean the training data with the `clean_utf8.py` script from `PortageTextProcessing`.[6] Subsequently, parallel training data is filtered with the `clean-corpus-n.perl` script from Moses (Koehn et al., 2007) to remove sentence pairs with a length ratio larger than 15. Punctuation is then normalized using the `normalize-punctuation.perl` script from Moses. Finally, non-breaking spaces (Unicode U+00A0 or '`\xa0`') and empty lines are deleted.

For all DE-CS parallel data and all monolingual DE and CS data, lines that contain characters which have not been observed in DE-HSB training data, WMT-News, or Europarl corpora are deleted. This is done to eliminate encoding issues and text that is clearly in other languages. The DE monolingual dataset consists of a likewise cleaned random sample of the full WMT21 News Crawl 19 corpus. The numbers of lines after filtering are shown in the two rightmost columns of Table 5.1.

5.4.3 Tokenization

We start tokenizing sentences into words with the Moses tokenizer: `tokenizer.perl -a -l $LNG`, where `$LNG` is `cs` or `de`, using the `cs` code also for HSB and DSB data. Then, we use Byte Pair Encoding (BPE) (Sennrich et al., 2016b)[7] to build a vocabulary of 20k subwords. For building the BPE models, we used all HSB-DE data, the Sorbian Institute and Witaj monolingual HSB data (but not the Web-crawled HSB data, which is too noisy), both sides of CS-DE data, and News-Commentary (DE) data. The HSB data was upscaled twice. The same datasets were used for extracting the joint vocabulary, which was then used to tokenize the source and target sides with a BPE-Dropout rate of 0.1 (Provilkov et al., 2020).

In post-processing, we detokenize BPE subwords with the BPE toolkit and then with a script from Moses: `detokenizer.perl -a -l $LNG`, where `$LNG` is `cs` or `de`, using the `cs` code also for HSB and DSB data.

[4] github.com/mjpost/sacrebleu, signature: `nrefs:1|case:mixed|eff:no|tok:13a|smooth:exp|version:2.3.1`.

[5] github.com/Tiiiger/bert_score, signature: `xlm-roberta-large_L17_no-idf_version=0.3.12(hug_trans=4.26.0)_fast-tokenizer`

[6] github.com/nrc-cnrc/PortageTextProcessing

[7] github.com/rsennrich/subword-nmt

5.4.4 System Architecture

We use Transformer models (Vaswani et al., 2017) from the OpenNMT-py library (Klein et al., 2017) version 2.3.0.[8] We use the following default values of hyper-parameters from Transformer-Base: 6 encoder/decoder layers, 8 attention heads, Adam optimizer (Kingma and Ba, 2014), label smoothing of 0.1, dropout of 0.1, hidden layer of 512 units, and FFN of 2,048 units. We share the vocabulary and use the same embedding matrix for both input and output languages. The batch size is 8,192 tokens, and the maximum sequence length for both source and target is 501 tokens. We keep OpenNMT-py's scaling factor of 2 over the learning rate. We use standard values for hyper-parameters in order to maintain a simplified pipeline, although it is likely that a more regularized system could further improve scores (Atrio and Popescu-Belis, 2022).

We do not use any early stopping measure and train for a sufficiently large amount of steps to ensure convergence. We train the parent CS↔DE models for 500,000 steps, and the children and grand-children ones for 100,000 steps. To train our models we use between one and four Nvidia RTX 2080 Ti with 11 GB RAM which amounts to around 80 hours for parent models, 30 hours for children models (systems 3/4 and 5/6), and 15 hours for grandchildren models. As better parent systems lead to better children, we trained the parents for a longer time, given also the larger parallel data available.

We save checkpoints every 4,000 steps during training, and obtain the testing scores from an ensemble of the four best checkpoints in terms of BLEU scores on the validation data. When testing, we use a beam size of 5 for all systems, except when indicated otherwise for back-translation.

5.5 Results of the Proposed Pipeline

5.5.1 Parent DE to and from CS Systems

We first train the DE→$\text{CS}_{\text{parent}}$ and CS→$\text{DE}_{\text{parent}}$ models (see Figure 5.1) on the authentic parallel CS-DE data presented in Table 5.1. The BLEU and BERTScore of these systems, shown in Table 5.2, are respectively 20.2 and 22.1. These are comparable with the ones reported by NRC-CNRC (22–25 BLEU points) and with those with the same architecture appearing in the Opus-MT leaderboard[9], trained on OPUS parallel data (Tiedemann, 2012) using Opus-MT-Train (Tiedemann and Thottingal, 2020).

Choosing Czech for the parent model is reasonable due to its similarity with Upper and Lower Sorbian, but we have found that this similarity is not crucial (Atrio et al., 2021). Using a similar setup, we observed almost identical results with a Polish↔German parent model, and a loss of only 1.3 BLEU points with a French↔German one.

[8] github.com/OpenNMT/OpenNMT-py
[9] opus.nlpl.eu/leaderboard/DE→CS and CS→DE

5.5.2 Child DE to and from HSB Systems

We initialize the child systems DE→HSB$_{\text{authentic}}$ and HSB→DE$_{\text{authentic}}$ models from the highest-scoring checkpoint of the respective parent, and trained them on authentic parallel HSB-DE data. The systems reached BLEU scores of 56.7 and 56.1 respectively (see Table 5.2).

System	BLEU	BERTScore
DE→CS$_{\text{parent}}$	20.2	.936
CS→DE$_{\text{parent}}$	22.1	.938
DE→HSB$_{\text{authentic}}$	56.7	-
HSB→DE$_{\text{authentic}}$	56.1	.975

Table 5.2: BLEU and BERTScore on `newstest2019` for CS-DE parent models and `devel_test` for HSB-DE models trained only on authentic data.

One round of back-translation. We hypothesize that due to the already existing authentic parallel data, one round of back-translation (BT) could be sufficient. We use the above systems to generate synthetic parallel data from monolingual DE and HSB corpora. To generate it, we decode by sampling from the entire model distribution rather than applying beam search, following Edunov et al. (2018). As shown in Figure 5.1, with the HSB→DE$_{\text{authentic}}$ and DE→HSB$_{\text{authentic}}$ systems we translate the DE$_{\text{mono}}$ data into HSB$_{\text{BT}}$. Similarly, we translate the HSB$_{\text{mono}}$ data into DE$_{\text{BT}}$. Therefore, we obtain two pseudo-parallel datasets with authentic target sides. We apply to them the same filtering process as in Section 5.4.2, except for a more restrictive cut-off for `clean-corpus-n.perl`, using a maximum ratio of 1.5 between sentences instead of 15. This filtering results in the deletion of respectively 5% and 11% of the HSB-DE and DE-HSB pseudo-parallel datasets.

We continue training the HSB→DE$_{\text{authentic}}$ and DE→HSB$_{\text{authentic}}$ systems with authentic parallel HSB-DE data and the back-translated data, with the former being upscaled to match the number of lines of the latter. We obtain respectively the systems noted HSB→DE$_{\text{authentic+BT}}$ and DE→HSB$_{\text{authentic+BT}}$. The improvements brought by this round of back-translation are only of about 1 BLEU point (see Table 5.5). Our scores are similar to those reported by NRC-CNRC without inter-model ensembling (57-58 BLEU). With the highest-scoring checkpoint for each of HSB→DE$_{\text{authentic+BT}}$ and DE→HSB$_{\text{authentic+BT}}$ we generate synthetic data for the unsupervised case by translating monolingual DSB and DE.

Iterative back-translation. We found that our pipeline does not benefit from multiple rounds of back-translation thanks to an additional experiment, not included in the final pipeline. Following Libovický and Fraser (2021b), for each round of back-translation i (with $i = a, b, c$), systems HSB→DE$_{\text{authentic+BT}(i)}$ and DE→HSB$_{\text{authentic+BT}(i)}$ are respectively initialized from the parent models CS→DE$_{\text{parent}}$ and DE→CS$_{\text{parent}}$ trained on CS-DE data, instead of child systems trained on only authentic data HSB→DE$_{\text{authentic}}$ and DE→HSB$_{\text{authentic}}$ as performed above. Decoding and filtering remain as described above as well. Otherwise, the first round of back-translation remains as above, and the second round results in new pseudo-parallel datasets on which we train new systems in both directions (also including upscaled authentic parallel data HSB-DE), resulting in systems HSB→DE$_{\text{authentic+BT}(b)}$ and DE→HSB$_{\text{authentic+BT}(b)}$. We perform a third round to obtain systems HSB→DE$_{\text{authentic+BT}(c)}$ and DE→HSB$_{\text{authentic+BT}(c)}$. Hence, this method differs from our main proposed pipeline in the usage of three rounds versus one, and the initialization of models from CS-DE parents instead of the child HSB-DE systems trained on authentic parallel data.

While several studies have suggested that multiple back-translation rounds are beneficial, our findings are more nuanced. As we observe in Table 5.3, for the direction DE→HSB, the first round of back-translation improves BLEU by 1.2 points, but afterwards scores decrease. For the direction HSB→DE, on the contrary, BLEU scores continue to improve with more iterations, although with diminishing returns, with a final improvement of 0.7 points. We hypothesize that this is due to the monolingual DE dataset being larger than the HSB one.

Direction	System	BLEU
DE→HSB	DE→HSB$_{\text{authentic}}$	56.7
	DE→HSB$_{\text{authentic+BT}(a)}$	**57.9**⋆
	DE→HSB$_{\text{authentic+BT}(b)}$	**57.6**⋆
	DE→HSB$_{\text{authentic+BT}(c)}$	57.4
HSB→DE	HSB→DE$_{\text{authentic}}$	56.1*
	HSB→DE$_{\text{authentic+BT}(a)}$	56.5*
	HSB→DE$_{\text{authentic+BT}(b)}$	56.5*
	HSB→DE$_{\text{authentic+BT}(c)}$	**56.8**

Table 5.3: BLEU scores for only authentic parallel data, and three rounds of back-translation: DE→HSB systems are trained with DE$_{\text{BT}(i)}$-HSB$_{\text{mono}}$ and HSB→DE systems are trained with HSB$_{\text{BT}(i)}$-DE$_{\text{mono}}$. We note in bold the highest score in each direction. We denote scores that are *not* significantly different per direction with the same symbol.

In contrast, Libovický and Fraser (2021b) observed more significant improvements over four rounds of iterative back-translation, although also with diminishing returns. For HSB→DE, their improvement was 2.7 (up to 56.1 BLEU), starting however from a lower score than ours (53.4) and getting half of the improvement in the first iteration. For the DE→HSB, they achieve a smaller improvement of 1.6, up to 56.5 overall, starting from 54.9. Their highest scores are obtained after two rounds. We hypothesize that the difference between our results and theirs regarding the HSB→DE direction is explained by their use of ten times more monolingual DE data, coupled with a larger architecture.

Following Edunov et al. (2018) we experimented with various decoding methods for the back-translation stage. As a comparison to the full unrestricted sampling we use in all systems, we studied restricted sampling of the top 10 candidates, as well as the dropout of 10% of the words after standard decoding, and their combination. For DE→HSB$_{\text{authentic+BT}(a)}$ the three methods obtained nearly identical scores (57.54, 57.54, and 57.51), and none of them substantially deviated from our original method. This supports previous observations by Edunov et al. (2018) showing that differences between decoding algorithms for back-translation are only noticeable when the monolingual data size is large (e.g. more than 8M lines).

5.5.3 Grandchild DE to and from DSB Systems

In contrast with the DE↔HSB low-resource case, we hypothesize that more than one round of back-translation may be useful in the unsupervised case, due to the lack of parallel data. We used system HSB→$\text{DE}_{\text{authentic+BT}}$ to create the pseudo-parallel dataset DE_{BT}-DSB_{mono}, with which we trained system
DE→$\text{DSB}_{\text{unsupervised}(a)}$. With this system, we generated synthetic DSB data from the DE part of the HSB-DE authentic data as well as monolingual DE, resulting in DSB_{BT1}-DE and DSB_{BT2}-DE_{mono}. For rounds b and c we repeated the process as with HSB-DE, initializing system DE→$\text{DSB}_{\text{unsupervised}(b)}$, (and then c) and system DSB→$\text{DE}_{\text{unsupervised}(b)}$ (and then c), respectively from the highest-scoring checkpoint from systems DE→$\text{HSB}_{\text{authentic+BT}}$ and HSB→$\text{DE}_{\text{authentic+BT}}$, and generating synthetic data with each other. Filtering removed between 6-9% of the lines. The scores of the resulting systems are shown in Table 5.4.

Direction	System	BLEU
DE→DSB	DE→$\text{DSB}_{\text{unsupervised}(a)}$	26.1
	DE→$\text{DSB}_{\text{unsupervised}(b)}$	**29.4⋆**
	DE→$\text{DSB}_{\text{unsupervised}(c)}$	**29.5⋆**
DSB→DE	DSB→$\text{DE}_{\text{unsupervised}(a)}$	36.5
	DSB→$\text{DE}_{\text{unsupervised}(b)}$	**38.1***
	DSB→$\text{DE}_{\text{unsupervised}(c)}$	**38.4***

Table 5.4: BLEU scores for three rounds of back-translation: DE→DSB systems are trained with $\text{DE}_{\text{BT}(i)}$-DSB_{mono} and DSB→DE systems are trained with $\text{DSB}_{\text{BT}(i)}$-$\text{DE}_{\text{mono}}$ and $\text{DSB}_{\text{BT}(i)}$-DE (the DE part of the authentic HSB-DE data). The highest score in each direction is in bold. Scores that are *not* significantly different per direction are marked with the same symbol.

System	DE→HSB	HSB→DE		DE→DSB	DSB→DE	
	BLEU	BLEU	BERTScore	BLEU	BLEU	BERTScore
NRC-CNRC	**59.9**	**60.0**	-	**31.0**	34.9	-
LMU	56.5	56.2	.938	30.1	33.8	.874
NoahNMT	58.3	58.5	-	-	-	-
CL_RUG	52.1	51.6	-	24.9	32.1	-
IICT-Yverdon	54.6	53.2	-	9.62	-	-
Ours	57.4	57.0	.976	29.4	**38.1**	.958

Table 5.5: BLEU and BERTScore on the 'devel_test' set of the best-performing system of each team, with our proposals at the bottom. The highest score per direction is in bold. The systems are referenced in Section 5.2 above, and '-' indicates that the score is not available.

For DE→DSB, the second round of back-translation produced a large improvement of 3.3 BLEU points over the first round, but the third round resulted in a minimal improvement of 0.1. The large improvement of system DE→$\text{DSB}_{\text{unsupervised}(b)}$ may be explained by the fact that the synthetic data used to train it is the first DE set translated by a true DSB system (DSB→$\text{DE}_{\text{unsupervised}(a)}$). For DSB→DE we also observe improvements from several rounds of back-translation, with the second one improving BLEU by 1.6 points and the third round improving only minimally by 0.3 points. We hypothesize that this difference is due to the lower amount of DSB monolingual data versus DE, and the back-translation of the DSB data being generated by a model that had not been trained on DSB. For both directions (DE→DSB and DSB→DE) the difference between systems *a* and *b* was significant, but not between *b* and *c*. As a result, we excluded extra rounds of back-translation for low-resource HSB-DE from our simplified pipeline, and only performed two rounds for unsupervised DSB-DE.

5.6 Discussion and Conclusion

We show in Table 5.5 the final results of our pipeline, compared to the highest scores for each direction obtained in the WMT 2021 shared task (Libovický and Fraser, 2021a). Scores from CFILT (Khatri et al., 2021) are not shown because we do not have access to their 'devel_test' scores. HSB-DE scores from CL_RUG are intermediate scores for their unsupervised DSB-DE systems.

On both low-resource directions (HSB↔DE) our simpler pipeline obtains comparable results to the three highest-scoring teams (NRC-CNRC, LMU and NoahNMT systems). Our scores on one unsupervised direction (DSB→DE) surpass those of the three participants, while on the other (DE→DSB) our scores are comparable to those of the two highest-scoring teams (NRC-CNRC and LMU). To explain the latter result, we hypothesize that our simplified pipeline is more sensitive to weight initialization, and therefore is less robust across all directions than a more complex pipeline.

Compared to the NRC-CNRC submission, our pipeline uses the same data selection and filtering, a single vocabulary for the tokenizer, trains from a single random initialization for each of the translation direction, does not train multitask or multilingual models, uses a much simpler filtering for back-translated sentence pairs, and sets a single set of values for hyper-parameters such as learning rate and label smoothing.

Compared to LMU, our pipeline uses a smaller amount of authentic parallel data for the parent CS↔DE models, does not use monolingual data back-translated for these parent models, and uses an architecture with fewer parameters (Transformer-Base instead of Big). Moreover, we use only one round of back-translation instead of four for the child HSB↔DE systems and two instead of eight for the grandchild DSB↔DE systems submitted by LMU.

NoahNMT also produced high scores on the supervised tasks, although with the use of a pre-trained BERT model (Devlin et al., 2019), vast amounts of monolingual data (100M lines), and dual parent transfer. CL_RUG scored well in the unsupervised tasks, but made use of sequence masking, denoising auto-encoding, cross-lingual back-translation, and vocabulary alignment between HSB and DSB with VecMap (Artetxe et al., 2018). IICT-Yverdon applied a scheduled multitask training to both the supervised and unsupervised directions, which appeared to be particularly ineffective for the unsupervised task.

We now provide some tentative explanations on why our simplified pipeline produces scores that are comparable with those from more complex ones. Firstly, a much better trained parent model does not necessarily result in noticeable better child models. Whatever the cause of the improvement of the parent models (additional parent training data, parent back-translation, or additional parent pairs), when several stages in the training pipeline can be found afterwards (such as training on authentic data, then children back-translation, then grandchildren back-translation, etc.), the initial benefit may be lost later in the pipeline. This is particularly exacerbated when child systems are later trained with data of dubious quality, such as back-translations. Artetxe et al. (2020a), for instance, showed that when performing iterative back-translation, the quality of the initial system has minimal effect on the final performance, as systems tend to converge to scores dictated by the monolingual data.

This first explanation feeds into a second explanation: large amounts of parent parallel or monolingual data make it reasonable for practitioners to choose larger architectures, which must then be carried over to the lower-resource children, since pruning rarely happens mid-pipeline. Although there is evidence that fitting large models to very small amounts of data is not necessarily detrimental (Belkin et al., 2019) and can even be beneficial (Li et al., 2020), it is unclear if this still holds with a more complex training pipeline. In any case, a smaller architecture in a low-resource setting, while still over-parameterized, can perform as well as a larger one.

Finally, modern Transformer-based systems are robust, and there seems to be a large area of “acceptable results" which is relatively easy to access, as we have empirically shown with our comparison to five different submissions to the WMT shared task. However, our pipeline is only trained on a group of similar languages (Czech, Upper Sorbian, and Lower Sorbian) to and from German, which may not generalize in the same manner to other languages or domains.

To sum up, although the competition to achieve first place in shared tasks such as the one discussed here leads participants towards increasingly complex pipelines, we have shown that competitive or even better results can be achieved with a much simpler training pipeline. Nonetheless, the lack of available parallel data for low-resource NMT calls for research on a useful usage of monolingual data even on a complex pipeline. In the following chapter, we study whether a more complex use of monolingual data, in the form of fixed-scheduled multitasking with auxiliary tasks, can improve the scores on a low-resource pipeline.

5.7 Perspectives

The conclusions from the main experiments in this study should be strengthened by replicating them with more datasets. It would be interesting to replicate our final results across several runs, particularly the unsupervised ones, to observe the variation that can be expected with different initial weights. Although it would be computationally expensive, a natural progression of this study would be to perform various ablation studies where we explicitly compare our pipeline to various specific techniques, in order to better control the effects of each low-resource translation techniques, instead of only comparing a simplified pipeline to other studies.

6 Fixed-Scheduled Multitask Training[1]

In this chapter, we present the systems submitted by our team from the Institute of ICT at HEIG-VD to the Unsupervised MT and Very Low Resource Supervised MT task at WMT 2021 (Libovický and Fraser, 2021a). We first study the improvements brought to a baseline system by techniques such as back-translation and initialization from a parent model. We find that both techniques are beneficial and suffice to reach performance that compares with more sophisticated systems from the 2020 task. We then present the application of this system to the 2021 task for low-resource supervised Upper Sorbian (HSB) to German translation, in both directions. Finally, we present a contrastive system for HSB-DE in both directions, and for unsupervised German to Lower Sorbian (DSB) translation, which uses multi-task training with various training schedules to improve over the baseline.

6.1 Introduction

In this study, we present the systems submitted to the WMT 2021 task on Unsupervised MT and Very Low Resource Supervised MT. We first build a series of baseline systems, driven mostly by considerations of simplicity, trained on data from the 2020 edition of the task, for translation between Upper Sorbian (HSB) and German (DE). These systems, described in Section 6.4, enable us to quantify the merits of using additional back-translated data (Sennrich et al., 2016a) and of initializing the system for a low-resource pair with parameters learned on a high-resource pair (same target language and related source language).

The systems described above serve as the basis for our submitted to the shared task, for DE→HSB and HSB→DE, presented in Section 6.5, which improves upon our 2020 baseline with the addition of more parallel data, and achieves competitive performance with the use of back-translation and parent-initialization only. However, this approach does not lead to an effective baseline for unsupervised German to Lower Sorbian (DSB) translation (Section 6.6).

[1] This work was performed in collaboration with Gabriel Luthier, Axel Fahy, and Giorgos Vernikos and published in Atrio et al. (2021). I was responsible for the design of the baseline system and the design and implementation of the contrastive system.

In Section 6.7, we present experiments with a contrastive system that implements multi-task learning, with several schedules, in which denoising tasks together with translation are presented to the systems in increasing order of complexity, leading to more robust HSB↔DE systems, together with a strategy of diverse ensembling. We also use our DE→HSB system to initialize a multi-task DE→DSB system for the unsupervised task, although in this case the performance is not competitive.

6.2 Related Work

Generally, as illustrated in Figure 6.1, there are three main components to scheduling of tasks: a difficulty measurer, which provides a quantified measure of the complexity of a task, a scheduler, which determines on what a step will be trained on, and a batcher, which determines how the samples of potentially different tasks will be assembled. In the following we provide a general summary for each of these concepts, and refer to Table 6.1 for the specific literature.

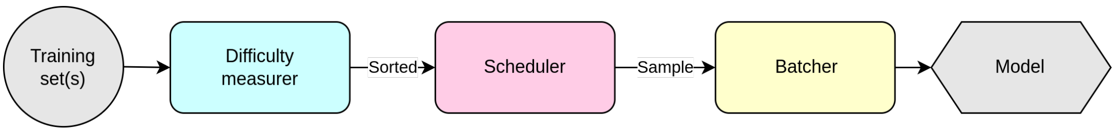

Figure 6.1: Abstract structure of adaptive scheduling.

A difficulty measurer can be static, i.e. determined before the multitask model starts training. This in itself can be a simple heuristic such as considering long samples as more complex than shorter ones, or it can rely on some pre-trained model. Alternatively, the complexity of a sample or task can be calculated during training, that is, dynamically, typically using some measure of the performance of the model at a given point during its regime.

A scheduler can be explicit (determining which examples can be sampled by the batcher to train on) or implicit (assigning different weights to examples that can be trained on). Each can be static or dynamic as well, and can range from simple methods such as a linear schedule over the total amount of training steps to scaling the gradients of samples or tasks based on the model's performance. Dynamic scheduling methods are sometimes called competence or self-pacing.

A batcher can be continuous if there is no intra-classification among samples or discrete otherwise. If it is continuous it typically samples examples based on length (the standard too outside of scheduled multitasking), and if it is discrete it may do so according to classifications based on task, linguistic features, or otherwise. Discrete batching can be called sharding.

Wang et al. (2020d) train on a bitext machine translation task and two auxiliary tasks (masked LM and denoising auto-encoding), and present two dynamic schedules: a dynamic sampling of tasks with a temperature-based scheduler based on epochs, and a dynamic noising ratio based on steps, to increase the difficulty of the auxiliary tasks. They observe improvements

Difficulty measurer: Static	
Heuristics (length, frequencies, etc.)	Kocmi and Bojar (2017)
	Platanios et al. (2019)
Linguistic features (parsing depth, etc.)	Kocmi and Bojar (2017)
Pre-trained model	Lu and Zhang (2021)
Filtering (Moore-Lewis, etc.)	Zhang et al. (2019b)
Difficulty measurer: Dynamic	
Loss (absolute)	Zaremoodi and Haffari (2019)
Decline of loss (relative)	Xu et al. (2020a)
Variance over M runs (dropout, etc.)	Wan et al. (2020)
Scheduler: Static	
{Steps, epochs, etc.} × {Linear, sqrt, etc.}	Platanios et al. (2019)
Boosting or replacing	Zhang et al. (2018)
Scheduler: Dynamic	
Current vs. pre-trained	Xu et al. (2020a); Jean et al. (2019)
Weighted loss	Zaremoodi and Haffari (2019)
	Gan et al. (2021); Wan et al. (2020)
Scaled gradients or learning rate	Jean et al. (2019)
Batcher: Continuous	
By length	Kocmi and Bojar (2017)
By difficulty	Xu et al. (2020a)
Batcher: Discrete	
By difficulty	Zhang et al. (2018)
By features	Kocmi and Bojar (2017)

Table 6.1: Main options for adaptive scheduling / curriculum learning. Certain methods, like genetic meta-learning, are not represented here, which intends to be a more tailored view of standard methods in NLP. For a more detailed taxonomy, see recent surveys by Wang et al. (2021); Soviany et al. (2022).

particularly on low-resource languages. Liu et al. (2020a) propose to use the sum of the norm of the token embeddings of a sample as a difficulty measurer, and the source embedding of the NMT model at a given step as a competence measure (to filter out examples from sampling). They also present a weighting formula from the two measures in order to not over-sample easy examples. They observe improvements in quality and training time on high-resource datasets.

In Section 7.2 we present more related work, specifically regarding data balancing for multilingual and adaptive NMT.

6.3 Datasets

We use various Upper Sorbian datasets from the 2020 edition of the task, and additional WMT data, as presented in Table 6.2, and previously in Chapter 5. The monolingual HSB data from 2020 comes from three sources: `sorbian_institute_monolingual` consists of a mix of high- and medium-quality HSB data provided by the Sorbian Institute; `witaj_monolingual` consists of high-quality HSB data from the Witaj Sprachzentrum; finally, `web_monolingual` consists of web-scraped noisier HSB data gathered by the Center for Information and Language Processing from LMU Munich (Fraser, 2020). We kept from all datasets only sentences that have strictly more than 2 and strictly fewer than 301 words.

Dataset	Lang.	Before filtering		After filtering	
		sentences	words	sentences	words
Sorbian Institute Mono.	HSB	339,822	5,044,079	339,822	5,044,079
Web Monolingual	HSB	121,003	1,661,898	115,632	1,651,154
Witaj Monolingual	HSB	222,027	2,672,255	215,370	2,660,805
Europarl v8	DE	2,234,583	48,430,884	2,186,477	48,347,698
JW300	DE	2,366,722	34,782,112	2,182,801	34,519,064
News Commentary v15	DE	422,009	8,942,517	409,955	8,939,335
Europarl v8 CS-DE	CS	568,589	11,571,876	562,716	11,561,049
Europarl v8 CS-DE	DE	=	13,098,638	=	13,086,320
JW300 CS-DE	CS	1,052,338	13,579,350	982,034	13,435,536
JW300 CS-DE	DE	=	15,133,882	=	14,992,424
News Comm. v13 CS-DE	CS	174,789	3,486,672	172,987	3,479,819
News Comm. v13 CS-DE	DE	=	3,751,102	=	3,746,708
WMT 2020 HSB-DE Train	DE	60,000	724,572	59,030	722,076
WMT 2020 HSB-DE Train	HSB	=	639,740	=	637,883
WMT 2021 HSB-DE Train	DE	87,521	1,251,339	87,502	1,251,287
WMT 2021 HSB-DE Train	HSB	=	1,094,421	=	1,094,375

Table 6.2: Monolingual and parallel corpora with their languages and numbers of lines (sentences) and words, before and after filtering by length (keeping sentences with more than 2 and fewer than 301 words).

6.4 Baseline HSB→DE System on 2020 Data

6.4.1 Subword Vocabulary

For the HSB→DE system, we use CS→DE initialization in several experiments, because Czech (CS) is a high-resource language and close neighbor to Upper Sorbian. Therefore, we create a tri-lingual shared subword vocabulary (CS, DE, HSB) using the Unigram LM model (Kudo, 2018) as implemented in SentencePiece.[2] We apply 32,000 merges and the other parameters of SentencePiece are kept to default values. We obtain 600k sentences of HSB data from `sorbian_institute_monolingual`, `witaj_monolingual` and `train.hsb-de`, the latter being the HSB side of the 2020 training data. We do not use `web_monolingual` as it appears to be noisy, due to the collection process. For CS and DE, 600k sentences are selected randomly from the monolingual corpora listed in Table 6.2. The vocabulary generated by SentencePiece is converted from log probabilities to frequencies using the `spm_to_vocab.py` tool from the

[2]https://github.com/google/sentencepiece (v. 0.1.95)

OpenNMT-py toolkit. Using a common SentencePiece model for the three languages is not obligatory, but appeared to improve the performance by 2-3 BLEU points in most cases.

6.4.2 System Parameters and Results

We use OpenNMT-py (Klein et al., 2017) for our experiments.[3] We start with Transformer-Base (Vaswani et al., 2017) (78M parameters) but also experiment with Transformer-Big (245M parameters), with their main parameters described in Table 6.3. We apply the same regularization and optimization procedures to the two models. We accumulate gradients over 2 batches and train on 2 GPUs, with a `batch_size` of 1k for Base and and 2k for Big. We use the "noam" learning rate schedule (Vaswani et al., 2017) with its values at each step multiplied by two, and 8k warmup steps. We evaluate and save checkpoints every 5k steps. Final translations are generated with a beam width of 5, ensembling the last two checkpoints in these experiments. We report BLEU scores (Papineni et al., 2002) obtained with SacreBLEU (Post, 2018) on detokenized text.

	N	h	d_{model}	d_{ff}	P_{drop}	steps
Base	6	8	512	2048	0.1	60k
Big	6	16	1024	4096	0.3	100k

Table 6.3: Parameters of the two Transformer models used in our experiments. Other parameters are set to the default values of the OpenNMT-py toolkit.

6.4.3 Use of Back-translated Data

The first HSB→DE system we trained, for comparison purposes, used only the HSB/DE parallel data provided for the WMT 2020 Low-Resource task. Its BLEU scores are 47.98 on the 'dev' set (`devel.hsb-de`) and 41.22 on the 'devtest' set (`devel_test.hsb-de`) after 60k steps of training (first line of Table 6.4). The already high BLEU scores that are reached, compared to scores generally observed on high-resource language pairs, indicate that the 'dev' and 'devtest' sets are probably quite similar to the training data.

We obtain additional training data through back-translation (Sennrich et al., 2016a) of widely available monolingual German data. To this end, we train a DE→HSB model on the same parallel corpus as above, which reaches BLEU scores of 45.23 / 40.62 respectively on 'dev' and 'devtest'. Using this model, we translate News Commentary V15 from German into Upper Sorbian. The resulting pseudo-parallel data (noisy on the HSB side) is used in addition to the initial data for training a new HSB→DE model, which reaches a score of 52.91 / 44.39 (second line of Table 6.4). The improvement of this single enrichment with imperfect data of the initial low-resource system thus exceeds 4 BLEU points.

[3] https://github.com/OpenNMT/OpenNMT-py (v. 2.0.1)

6.4.4 Initialization with Parameters from a High-Resource Pair

The second technique we use for improvement is transfer from a high-resource pair (Zoph et al., 2016; Kocmi and Bojar, 2018), i.e. initialization with parameters from an MT system trained on such a pair. As Upper Sorbian has many similarities with Czech, which is a high-resource language, we initialize the HSB-DE model with the parameters of a model trained for CS→DE, then train it with the same data as in the previous subsection. Firstly, the CS→DE model is trained using Europarl and News Commentary, and reaches a BLEU score of 27.13 on a sample test set extracted from these two corpora.

The resulting HSB→DE system reaches BLEU scores of 55.99 / 47.53, a further increase of about 3 BLEU points (third line of Table 6.4). The use of an even larger dataset further improves performance: the addition of the JW300 corpus (Agić and Vulić, 2019) to the CS→DE training data increases BLEU by half a point (56.5 on 'dev'). The rather small increase could be attributed to the large difference in domains between JW300 and the HSB/DE data.

Since back-translation can provide very large amounts of data, we also trained a Transformer Big (with the parameters shown in Table 6.3) with the addition of the monolingual German corpora of Europarl and JW300 backtranslated into Upper Sorbian. This model reaches 58.08 / 49.99 BLEU points respectively on 'dev' and 'devtest', improving performance by more than 1.5 BLEU points. This is currently our best baseline model for HSB→DE, obtained with two simple augmentation techniques only.

We can compare this score with three of the highest-scoring systems on the 2020 HSB→DE 'devtest' set, noting some of the differences between them and our baseline. Scherrer et al. (2020) achieved a BLEU score of 56.9 using back-translation and bilingual pre-training with CS→DE, but also scheduled multitask with several monolingual and multilingual tasks. Knowles et al. (2020) achieved a BLEU score of 58.9 using iterative back-translation, multiplication of the HSB data for BPE training, and character- and word-level lexical modifications of Czech to make it more similar to Upper Sorbian. Libovický et al. (2020) achieved a score of 56.0 with much larger corpora for back-translation and CS→DE pre-training (14M lines) and the use of an unsupervised CS→HSB system to translate the CS side of the DE/CS parallel data into HSB.

6.4.5 Initialization with Parameters from Other High-Resource Pairs

We studied the role of the closeness between Upper Sorbian and the high-resource source language used for initialization, by reproducing the above initialization experiments (CS→DE) with Polish and French instead of Czech. Polish is a West Slavic language just as Czech and Upper Sorbian, although geographically more remote, whereas French is a Romance language: we thus expected the former to outperform the latter. To keep training time more manageable, we used a Transformer-Base, and trained the parent model on Europarl and JW300, because News Commentary is not available for Polish. For each experiment we build a different tri-lingual SentencePiece model trained with 600k sentences per language.

System	HSB→DE		DE→HSB	
	dev	devtest	dev	devtest
1. Transformer-Base, 2020 parallel data	47.98	41.22	45.23	40.62
2. Add back-translated data to #1	52.91 *(+4.93)*	44.39 *(+3.17)*	51.00 *(+5.77)*	43.23 *(+2.61)*
3. Initialize #2 with high-resource pair	55.99 *(+3.08)*	47.53 *(+3.14)*	–	–
4. Transformer-Big with #3	58.08 *(+2.09)*	49.99 *(+2.46)*	–	–
5. Add 2021 parallel data to #4	59.29 *(+1.21)*	51.86 *(+1.87)*	57.22 *(+6.22)*	49.95 *(+6.72)*

Table 6.4: Scores of our 2020 (1–4) and 2021 (5) baseline systems, with absolute improvements brought by each additional technique or data set.

The use of the PL→DE model (with a 22.33 BLEU score on its respective test set) for initialization leads to a HSB→DE performance of 56.07 / 47.94, which is very similar to the system initialized with CS→DE parameters (55.99 / 47.53). The use of the FR→DE model (with a 19.25 BLEU score) for initialization leads to a HSB→DE system reaching 54.92 / 46.30. This is about 1.3 BLEU points lower than with Polish or Czech, although the difference is smaller than expected given the linguistic distance between French and Upper Sorbian. These results are in line with the findings of Aji et al. (2020) who argue that no parent is clearly better than other for transfer learning in MT.

6.4.6 Two Rounds of Back-Translation

Multiple rounds of back-translation can be done on each side, but this computational effort is not always compensated by a significant increase of the BLEU score. Using the best HSB→DE system above, we translate monolingual HSB data and use it to train an improved DE→HSB model, which reaches 51.00 on the 'dev' data (+5.77 with respect to the initial DE→HSB system) and 43.23 on the 'devtest' data (+2.61). We then use this improved model to translate the monolingual German data again and use the resulting pseudo-parallel data to train a new HSB→DE model. The model without CS initialization reaches BLEU scores of 53.62 on 'dev' (+0.62) and 44.95 on 'devtest' (+0.43). If CS initialization is used, the models reaches respectively 58.44 (+0.36) and 50.03 (+0.04) on 'dev' and 'devtest'. The improvement brought by the additional rounds of back-translation is quite marginal, therefore we do not pursue this approach, and focus on a system which is initialized from a parent high-resource pair and trained with original and back-translated data, where the latter comes from a reverse system trained only with the original parallel HSB-DE data provided by the shared task.

6.5 Baseline HSB↔DE Low Resource Systems for 2021

Given the results of the previous section, we choose the Transformer-Big for our 2021 baseline. We change the dropout level from 0.3 to 0.1 since our experiments revealed an increase in performance with the latter value. Furthermore, we add the 87,502 sentences of additional parallel HSB-DE training data provided in 2021 to the datasets used in our 2020 baseline. We use the same SentencePiece model with DE, HSB, and CS data that we used for our 2020 baseline system, with approximately 700k lines for each language. At translation time, after observing a number of out-of-vocabulary tokens, we replace the unknown tokens with the source token that has the highest attention weight. We do not make any further changes regarding our 2020 Transformer-Big model.

The scores of our baseline systems on 2020 and 2021 data are synthesized in Table 6.4 for the various techniques we experimented with. Our baseline HSB→DE model with combined 2021 and 2020 data is system #5 in Table 6.4: it reaches BLEU scores of 59.29 on the ‘dev’ set and 51.86 on the ‘devtest’ set after training for 150,000 steps and by ensembling the best 4 saved checkpoints. For our DE→HSB model, we obtain 57.22 on the ‘dev’ set and 49.95 on the ‘devtest’ set after training for 85,000 steps and by ensembling the best 4 saved checkpoints.

After the submission to the 2021 shared task, we continued training the above HSB→DE model up to 300,000 steps- Ensembling the *last* 4 saved checkpoints, BLEU scores were close to the ones shown in the last line of Table 6.4, reaching 59.42 on the ‘dev’ set and 51.37 on the ‘devtest’ set. However, several checkpoints gained almost 2 BLEU points on ‘dev’, pointing to the potential benefits of training for a longer time.

6.6 Baseline for Unsupervised DE→DSB Translation

Moreover, we studied the same techniques for translating Lower Sorbian (DSB), for which no parallel resources are provided. We translated the monolingual DSB data provided by the organizers with our HSB→DE model, hypothesizing that the differences between DSB and HSB are small enough to obtain an acceptable DSB-DE pseudo-parallel corpus, with high-quality text on the DSB side, following insights from our experience with Swiss-German dialects (Honnet et al., 2018).

We use the parameters from our best DE→HSB model to initialize a DE→DSB model that we train for 120k steps with the DSB-DE pseudo-parallel data. When ensembling the best 4 checkpoints, we reach BLEU scores of 8.25 / 8.22 without observing any significant increase of the scores during training. In fact, the initial score, which is the performance of a DE→HSB model on the DE-DSB ‘devtest’ data, is even slightly higher. An even lower BLEU score was reached when using our CS→DE model to translate monolingual DSB data into DE to obtain a pseudo-parallel corpus, thus confirming the finding that this approach does not lead to pseudo-parallel corpora of sufficient quality. Therefore, we did not submit these translations to the 2021 shared task.

6.7 Contrastive HSB↔DE and DE→DSB Systems using Multi-Task Learning

In contrast to the baseline systems presented above, we study an innovative approach, in which we train multitask systems with denoising auxiliary tasks that are presented in order of increasing complexity. This insight is drawn from curriculum learning (Bengio et al., 2009). We thus test whether increasing the complexity of the tasks makes it easier for an NMT model to learn the simple tasks first, and the harder ones later in training. In particular, we will use a heuristic difficulty measurer based on estimated task complexity (see Section 6.7.1). With respect to related work in Table 6.1, Kocmi and Bojar (2017) and Platanios et al. (2019) also used heuristics, although with different criteria. We will also use a static scheduler, comparable with Platanios et al. (2019); Zhang et al. (2018), that we present in Section 6.7.2.

As Raffel et al. (2020) showed, source–to–source pre-training and multitasking improves translation, but not enough to compete with state-of-the-art setups. Therefore, instead, we perform target–to–target and source+target–to–target denoising. Considering their findings, we decide not to introduce special tokens into our vocabulary, such as mask tokens (instead just deleting the tokens with wish to mask), or sentence and language separators. Finally, due to computational constraints, we use the Transformer-Base as our architecture.

6.7.1 Data and Auxiliary Tasks

For our contrastive system we consider two new monolingual corpora in Czech and in German: the document-separated news crawls from WMT20 (Barrault et al., 2020), consisting of text extracted from online newspapers. They contain 17M lines and 43M lines respectively in each language. To keep training time within acceptable limits, we sample 1.4M lines from these corpora (including empty lines that serve as document-separators), we apply the same length-based filtering criterion ($2 < L < 301$) as for our baseline data, and we also delete all sentences that are made of more than 15% non-alphabetic characters. The resulting Czech corpus is 1.3M lines and 131,644 documents long, and the German corpus is 1.2M lines and 130,891 documents long.

For our document-level denoising tasks, we first divide into "chunks" a tokenized document-separated corpus so that each chunk is no more than 500 subwords in length, made up of consecutive lines in the same document; we only select documents made of at least 3 sentences. In Table 6.5 we list all corpora that we use to create our auxiliary data, including monolingual corpora back-translated with our baseline systems. The DE→DSB back-translated data was obtained with a baseline DE→HSB model.

We make use of the four following auxiliary denoising tasks (the main task being of course standard sentence-level translation, with all parallel and back-translated data), with the first two inspired by Devlin et al. (2019); Raffel et al. (2020) and Conneau and Lample (2019):

1. **Masking** (`MASK`): randomly delete 15% of words of a line on the source side, but keep the full original sequence on the target side.
2. **Translation Language Modeling** (`TLM`): concatenate the source and target sentences from a parallel corpus, and apply separately the `MASK` algorithm to each one. The target is the original target sentence.
3. **Mask Document First Words** (`MF`): for each chunk, leave the first sentence untouched, and for the remaining ones delete the first word of each sentence, with the target being the full original sequence in the same language.
4. **Next Sentence Generation** (`NSG`): for each chunk, leave all the sentences untouched except the last one, of which delete all but the two longest words; the model has to output the full original sequence. Keeping the two longest words (in characters) is based on the assumption that they are the most informative ones in the sentence.

The denoising tasks are listed above by increasing complexity. Indeed, `MASK`, as a monolingual sentence-level task, is the simplest denoising task we present, with `TLM` following, as it includes a context in a different language which needs to be identified. The two document-level tasks are more complex, as they require a larger context. In particular, `NSG` is harder than `MF`, since it consists of reconstructing a whole sentence with just two words from the original sequence, forcing the model to look for a more abundant context to estimate the correct answer. Furthermore, predicting the first word requires to take into account exclusively inter-sentential context, whereas masking a single random word allows also for the use of intra-sentential context, with the latter providing more direct context than the former. These document-level tasks represent an increase in complexity with respect to the sentence-level tasks, and we additionally hope that the training on intersentential context might prove beneficial for our models.

Corpus	Lines	Words	Aux. tasks
CS-DE	1.5M	25M / 28M	`TLM`
HSB-DE	144k	2M / 2M	`TLM`
CS	1.3M	41M	`MF`, `NSG`
DE	1.2M	44M	`MF`, `NSG`
HSB	640k	9M	`MASK`
DSB	128k	2M	`MASK`
HSB→DE	4.5M	94M / 104M	
DE→HSB	637k	10M / 9M	
DE→DSB	124k	2M / 2M	

Table 6.5: Parallel (2), monolingual (4), and back-translated corpora (3) used for our contrastive system trained with multi-tasking. Each corpus is assembled from the raw datasets presented in Table 6.2 with the filtering setup described in Subsection 6.7.1. For bilingual corpora, we indicate the number of words in each language.

6.7.2 Training Schedules

All our models translate to one target language only, therefore the target side of our datasets is always the same language, be it for the monolingual denoising tasks or for TLM. Since all datasets correspond to sequence-to-sequence tasks, we are in essence simply removing and introducing datasets during training. The specific splits of the tasks in each training schedule have been manually set, guided by the reasons given below, without any attempt for fine-tuning.

All the hyperparameters of the models are those presented in Section 6.4, with the only exception of the parameters of CS↔DE models for initialization, which were trained on 4 GPUs to reduce training time. When we introduce new tasks during the training of a model, we continue training from the last checkpoint of the previous task[4].

Training CS↔DE models. Both directions are trained according to the same schedule, shown in Table 6.6, with simply the source and target languages switched. First, we train for 30k steps with a TLM task, then we train for another 30k steps with a mixture of the MF auxiliary task (50% of the samples) and the main translation task (50%). Then we continue for another 30k steps, changing MF to NSG. Finally, we finish with 30k steps on translation only. In total, the model is being trained for 30k steps (25%) with TLM, 15k steps (12.5%) with MF, 15k steps (12.5%) with NSG, and 60k steps (50%) with the main task, i.e. sentence-level translation.

	Steps $\times 1000$			
Task	0-30	30-60	60-90	90-120
TLM	100%			
MF		50%		
NSG			50%	
Translation		50%	50%	100%

Table 6.6: Training schedule of the parent models in CS↔DE. For each direction, the model is only trained to output target language, so corpora differ depending on the direction (see 6.7.1). Both models are trained for 120k steps with three auxiliary denoising tasks and the main sentence-level translation task.

HSB→DE. The schedules of the child models are shown in Table 6.7 for the (DE, HSB) pair. For HSB→DE, we continue training from the best scoring checkpoint of the last 60k steps of the parent CS→DE model, and start with a TLM task for 60k steps. Then, we introduce back-translated data only for 60k steps. We continue with 60k steps with true parallel data only.

Additionally, we train two more models by continuing to train another 60k steps from the

[4]We empirically confirm that results are stable across multiple runs of CS↔DE MF, so we only train one instance of each.

best scoring checkpoint (which is also the last one saved), with one of the models having its learning rate schedule reset. Although at first performance worsens due to a more aggressive learning rate during the warmup steps, the model ends up converging to a score similar to the one we obtain if we continue to train without resetting the learning rate schedule. The goal is to emulate a multiple-run seeding strategy for ensembling, by achieving a different weight distribution among the two models. We additionally train a randomly-initialized model with parallel data only, for 60k steps, also for ensembling. We generate our translations of the test data with an ensembling of 16 models: the best 4 checkpoints from the parallel-only randomly-initialized model, the best 4 of our main setup during the first 60k steps of parallel-only training, and the 4 checkpoints each for the two runs that continued to train with, and respectively without, resetting the learning rate schedule.

DE→HSB. We continue training from the best-scoring checkpoint of the last 60k steps of DE→CS, and provide it with a `MASK` task for 60k steps, since the model has not seen the target language at all during pre-training, for this direction. Then, we provide the model with a `TLM` task for 60k steps. Since in this direction we have much less back-translated data than in the opposite, we decide to train for 60k more steps with 50% of the samples being from the back-translated data, and the other 50% from the true parallel corpora. Finally, we continue training two more models in the same manner as explained for the HSB→DE direction. We additionally train a randomly-initialized parallel data only model for 60k steps for ensembling. We translate with the same ensembling setup as described for the HSB→DE direction.

	Steps ×1000		
Task	0-60	60-120	120-180
	HSB→DE		
`TLM`	100%		
Trans-BT		100%	
Trans-Parallel			100%
	DE→HSB		
`MASK`	100%		
`TLM`		100%	
Trans-BT			50%
Trans-Parallel			50%

Table 6.7: Training schedule of the child models for the HSB→DE and DE→HSB models presented in 6.7.2.

DE→DSB. We start training with a `MASK` task for 60k steps from the highest-scoring checkpoint DE→HSB. We continue training for 60k steps with just the back-translated data, although we notice that the quality of the translation affects negatively the scores. To address this issue, for another 60k steps we give it the back-translated corpus for 50% of the samples and the `MASK` task for the other 50%, starting training from the previous highest-scoring checkpoint. Finally, for another 60k steps we give it a parallel-only DE-HSB task for 50% of the samples, `MASK` for

30%, and back-translated data for 20%. After testing, using just the highest-scoring checkpoint for the back-translation only, back-translation + MASK, and DE-HSB + back-translation + MASK appeared to work better on the development data than using the highest four ones.

Task	Steps ×1000			
	0-60	60-120	120-180	180-240
MASK	100%		50%	30%
Trans-BT		100%	50%	20%
DE-HSB				50%

Table 6.8: Training schedule of the child DE→DSB models presented in 6.7.2

6.7.3 Results

The scores of the parent DE→CS and CS→DE models obtained with multi-task training are shown in Table 6.9. We observe that training the models on mixtures of MF and NSG with translation produces positive results (rows 1 and 2). Compared to the CS→DE models from Sections 6.4 and 6.5, the present models have markedly lower scores. This difference can be due to the use of Transformer-Base vs. Big, or to differences in training data, apart from the multi-task training procedure itself. Still, we decided to use these models as parents for initializing the DE→HSB and HSB→DE models respectively, so that both parents and children are trained with multi-tasking. Although changes in the parameters of a parent model that result in better translations may not necessarily also result in better child initialization, it would be interesting to also test here the parent models from Section 6.5.

System	DE→CS	CS→DE
1. MF + translation	14.05	15.46
2. NSG + translation	15.30	16.17
3. Translation	18.19	19.80

Table 6.9: BLEU scores of parent models after each stage of the training schedule described in 6.7.2, on the 'devtest' set from 6.5.

Our child DE↔HSB models show that the scheduled training improves results over the baseline. The HSB→DE model with a training schedule (system 2 in Table 6.10), trained with a lighter architecture (Base vs. Big) and lower quality parent model (19.8 vs. 24.5), achieves a higher BLEU score than the system in Section 6.5, as shown in Table 6.4: 52.2 vs. 51.86. Additionally, the diversity of the ensembling of the models appears to improve the overall quality of the translation.

The scores of our DE→DSB model (Table 6.11) show that the quality of the back-translated data with our HSB→DE model improved slightly with the addition of the MASK monolingual task, but not with the addition of a DE→HSB translation task. However, when including in the ensemble the models trained on a DE→HSB task, scores improved from 8.7 to 9.6 on the 'devtest' set. This was the version submitted to the shared task on unsupervised MT

System	DE→HSB	HSB→DE
1. Parallel data	50.37	48.50
2. Multi-task	52.10	52.21
3. #2 cont. train	53.42	52.37
4. #2 cont. train with l. r. reset	53.05	52.12
Ensemble	54.58	53.21

Table 6.10: BLEU scores of child DE↔HSB models for various training schedules on the 2021 ‘devtest’ set.

(DE→DSB).

System	DE→DSB
1. Back-translation only	8.23
2. BT + `MASK`	8.57
3. BT + `MASK` + DE→HSB	7.14
Ensemble	9.62

Table 6.11: BLEU scores of child DE↔DSB models for various training schedules on the 2021 ‘devtest’ set.

Finally, as we can see in Table 6.12, even with our possibly suboptimally trained parent models and lighter architecture, the strategy of diverse ensembles and scheduled multi-task training improved over our best performing baselines given in Section 6.5 for all directions of the low-resource MT task. It remains to be seen whether a combination of the fixed-schedule multitasking approach presented in the current chapter and the simplified pipeline presented in Chapter 5 (the experiments of which were performed after this chapter) would continue to improve these scores.

HSB→DE		DE→HSB		DE→DSB	
dev	devtest	dev	devtest	dev	devtest
62.74	53.21	62.49	54.58	9.22	9.62
(+3.45)	*(+1.35)*	*(+5.27)*	*(+4.63)*	*(+0.97)*	*(+1.40)*

Table 6.12: BLEU scores of our primary system’s final configurations, on the development data, with the improvements over our highest baselines from Section 6.5.

6.8 Conclusion

In this chapter, we showed that non-iterative back-translation and parent-model transfer learning provide improvements for translation in a low-resource setting. Furthermore, multi-task scheduled training with monolingual or cross-lingual tasks also resulted in better models. In particular, child models starting with Translation Language Modeling tasks and Masking tasks improved over the baseline in all translation directions. Finally, our strategy of ensembling diverse models also produced higher scores than a mere checkpoint ensemble strategy.

6.9 Perspectives

A comparison with stronger baselines may result in a smaller, but more reliable, improvement from our method. Additionally, it should be studied why our method does not perform successfully in an unsupervised direction. Finally, an interesting continuation of this study would be to replicate our strategy with languages for which document-level test data exists, and check their performance in this data. Since our two novel tasks are both document-level tasks, it is possible that the resulting models improve translation that involves longer-range dependencies.

7 Dynamically-Scheduled Multilingual Training[1]

Many-to-one neural machine translation systems have been shown to improve over one-to-one systems when training data is scarce. In this chapter, we design and test a novel algorithm for selecting the language of minibatches when training such systems. The algorithm changes the language of the minibatch when the weights of the model do not evolve significantly, as measured by the smoothed KL divergence between all layers of the Transformer network. This algorithm outperforms the use of alternating monolingual batches, although not the use of shuffled batches, in terms of translation quality (measured with BLEU and COMET) and convergence speed.

7.1 Introduction

Multilingual neural machine translation (MNMT) systems can be trained with several languages on the source side, or on the target side, or on both sides (Firat et al., 2016; Johnson et al., 2017). Many-to-one MNMT systems are particularly effective for low-resource languages (LRLs) on the source side, when they are accompanied by high-resource languages (HRLs) related to them (Gu et al., 2018). For instance, Neubig and Hu (2018) trained a many-to-one recurrent model on a multilingual dataset of almost 60 languages and showed that including HRLs in the training data reduces the chance of overfitting to the LRLs and improves translation quality. Aharoni et al. (2019) used Transformer models (Vaswani et al., 2017) to further improve over these results.

Many-to-one MNMT systems are usually trained with multilingual batches sampled from all source languages to avoid catastrophic forgetting (Jean et al., 2019), but the presence of several languages in a minibatch may ineffectively constrain the model and prevent it from training

[1]This work was performed in collaboration with Alexis Allemann, and is submitted to the ACL Rolling Review system for presentation to an ACL venue. I was responsible for the design of the system and the setup of experiments, while the implementation of the *self-paced* algorithm and the training of the models was done by Alexis Allemann.

on the languages where training is most needed. An open question in many-to-one MNMT, therefore, is how the data from different source languages should be sampled during training, particularly when massive imbalances in sizes or difficulties occur across languages.

In this chapter, we propose a dynamic scheduling approach which samples minibatches from the source languages based on the variation of weights in the layers of a Transformer. The main idea is the following one: when a model becomes competent for translating a certain source language, as indicated by a decreasing variation of a model's weights across consecutive training steps, then the language of the minibatches should be switched to a new one, in order to allocate more time to more challenging, hence presumably more useful tasks.

The main contributions of this study are the precise formulation and testing of the idea. Specifically, we propose to:

- measure variation of weights by comparing the weights of all layers of a Transformer across two consecutive training steps with the same source language;
- compare weights by using symmetric KL divergence between softmaxes of layers, with exponential smoothing across time;
- trigger a change of task, i.e. source language, when weight variation decreases;
- compare translation quality and convergence speed for 8-to-1 MNMT on a dataset with four language families on the source side, and one HRL and one LRL for each of them (Neubig and Hu, 2018).

7.2 Previous Work on Multilingual and Adaptive Methods for NMT

We now present relevant previous work, specific to this chapter, on balancing of various datasets for multilingual training and adaptive or self-paced multitask training.

Neubig and Hu (2018) study the upsampling of the LRL data when building minibatches, and observe that keeping the original proportions of HRL and LRL performs marginally better. Aharoni et al. (2019) also sample each batch uniformly from a concatenation of all language pairs. Arivazhagan et al. (2019) compare a simple concatenation with uniform balancing (Johnson et al., 2017), but observe better results for LRLs when translating into a HRL by using a temperature-based upsampling, which has been favored afterwards (Conneau et al., 2020; Tang et al., 2021).

Fan et al. (2021) propose a variation of temperature sampling, *Sinkhorn Temperature Sampling*, as an extension to a many-to-many setting so that the distribution of languages on source and target sides is equal to the given target distribution. On a 100-language mined dataset, they observe big improvements, particularly when translating between non-English directions with respect to both uniform and standard temperature-based scaling. Kumar et al. (2021) propose a Reinforcement Learning approach, with several contextual multi-arm bandits that independently learn per-language sampling policies jointly with the target NMT system. Zhou et al. (2021) apply Distributionally Robust Optimization (Ben-Tal et al., 2013; Duchi et al., 2016) to MNMT. This new training objective produces improvements on several low-resource languages, particularly among those that are not very low-resource. Fernandes et al. (2023) observe a Power Law relation between the weight of one direction on the multilingual training and the model's performance on it afterwards. Chen et al. (2023) further refine this by proposing a Double Power Law. They perform an extensive empirical comparison of multilingual models with different architectures and directions, and show that when noticeable data imbalances occur between the directions, increasing the weight of some directions on the training objective does not entail an improvement of performance on said directions. Training on several languages can produce interference among them, hence decreasing performance (Conneau et al., 2020). Shaham et al. (2023) propose that scaling up models and an modification of standard temperature sampling can mitigate this interference.

Pham et al. (2022a) apply a variation of Differentiable Data Selection (Wang et al., 2020b) to multi-domain adaptation NMT, which can also be used for MNMT, since different domain datasets amount to different tasks. In particular, they use reinforcement learning to learn a curriculum by evaluating the usefulness of samples during training. Wang et al. (2020a) propose a dynamic data selection curriculum also for multi-domain NMT, in their case by proposing instance-level features to select samples based on their usefulness across various domains at once (cross-entropy difference between a baseline model and fine-tuned models for specific domains). Instead of defining a schedule implicitly by assigning weights to tasks, or explicitly by ordering them, Pham et al. (2022b) adaptively learn to assign sub-networks to each task by applying group-dropout to selected sub-networks, by considering the similarity between tasks, but also by allowing smaller parts of the model to focus on individual tasks.

As a method for dynamic scheduling of multitask training (Caruana, 1993), self-pacing consists in using the target model to quantify the difficulty of each sample or dataset – that is, to measure the model's *competence* – and inform the scheduling module dynamically (Kumar et al., 2010). Self-pacing has been used in NMT at the sample-level, for instance by measuring variance across dropout runs (Wan et al., 2020). Similarly, Liu et al. (2020a) set a self-paced curriculum based on the norm of a token's embedding, for a single task.

For MNMT, Jean et al. (2019) compare adaptively upsampling a language depending on various factors, observing best results on the LRLs when dynamically changing the gradient norm following Chen et al. (2018). Wang et al. (2020c) adaptively balance the languages by learning language weights based on the model's competence on a development set. Zhang et al. (2021b) adaptively learn a sampling strategy by measuring per-language competence and LRL competence evaluated with a HRL's competence. Wu et al. (2021) also balance the data dynamically by measuring the model's uncertainty on a development set, estimated by the variance over several runs of Monte Carlo dropout (Gal and Ghahramani, 2016).

As previous work shows, when training MNMT models with a variety of datasets or tasks, balancing between the sizes of the different datasets is not a trivial matter, and complex methods like temperature-based upsampling have been shown to improve results, particularly on low-resource languages. Encouraged by the success of recent adaptive approaches to MNMT, in this chapter we will combine the two, by proposing an adaptive, self-paced approach to the balancing of dataset on low-resource languages.

7.3 Method for Self-Paced MNMT

To train a many-to-one MNMT model, we consider M parallel datasets which correspond to as many tasks $\mathcal{T} = \{T_1, \ldots, T_M\}$ with different source languages and their respective English translations. Our algorithm chooses on which task T_c to train the MNMT model, based on an estimation of the model's competence for each task (i.e., source language). The overall goal is to *increase time spent on tasks where the model is less competent, and to avoid over-training on tasks where the model is already competent.*

The proposed algorithm for dynamic scheduling (Algorithm 1 below) has the following rationale. If the network is trained on a task T_c and the weight variation across consecutive steps increases, we consider that the network lacks competence on T_c and should keep training on it. Conversely, the less the weights change, the more competent the model is. So, if weight variation slows down, then training on the same task produces diminishing returns, and the network should switch to a task on which it is less competent. This condition appears in line 8 of Algorithm 1. We note that our algorithm carries little computational overhead, since the self-assessed competence is obtained from the weight variation across standard training steps.

We propose to estimate the per-task competence of the model as the average variation of its weights in all layers (due to the back-propagation of gradients) at a given training step. We thus propose to measure competence as the Kullback-Leibler divergence (D_{KL}) between the updated weights and the weights at the previous step at which the model was trained on the same task.[2] The parameters (or weights) of the model at time step t are noted θ_t.

Originally used to quantify the dissimilarity between two probability distributions P and Q, D_{KL} is defined as: $D_{\mathrm{KL}}(P||Q) = \sum_x P(x)\log(P(x)/Q(x))$ where x are the possible values of the P and Q random variables. To use D_{KL} as a distance measure between two sets of weights in a neural network, we apply softmax σ to convert the weights to probability distributions. Moreover, we take the logarithm of the first term in D_{KL} to handle the potential issue of capacity overflow and maintain the stability of divergence calculations, following the example of Liang et al. (2021). Finally, we symmetrize the distance by summing D_{KL} divergence in both directions.

Therefore, we compute the average variation between two sets of values θ_{t-1} and θ_t of all the

[2] If the model is trained with monolingual batches, that is, all batches coming from the same dataset, then the per-task competence is measured by keeping track of on which dataset the model is training at a given step. With multilingual batches, computing per-task competence is more complicated, although we explain it below.

trainable weights of a Transformer network (layers 1 through L) as follows:

$$D(\theta_{t-1},\theta_t) = \frac{1}{2L}\sum_{i=1}^{L} D_{\mathrm{KL}}(\log(\sigma(\theta^i_{t-1}))||\sigma(\theta^i_t)) + D_{\mathrm{KL}}\log(\sigma(\theta^i_t))||\sigma(\theta^i_{t-1})).$$

With this, we compute per-task variation in weights between steps, and switch training to another task when the weight variation is lower than in the previous step of the same task, indicating that the model might be plateauing in its learning of that task. Furthermore, we ensure that when the training switches to another task, the model trains on it for at least two training steps, so that both θ_{t-1} and θ_t are the result of training on minibatches of the same source language: with this, we avoid measuring a large variation between weights simply as the result of switching between tasks.

In order to obtain a task-switching schedule that is robust to local variations, we apply exponential smoothing and compute per-task competence, transforming D into D'_c as follows:

$$D'_c(\theta_{t-1},\theta_t) = (1-w)D(\theta_{t-1},\theta_t) + wD'_c(\theta_{t-k},\theta_{t-1})$$

where $k \geq 2$ is the smallest value such that $B_{t-k} \in T_c$ (in other words, $t-k$ is the latest step before $t-1$ for which the model trained on a batch that was sampled from the current training task T_c). The smoothing weight was set at $w = 0.995$ after empirical analyses (see Section 7.5.2).

We define the model's per-task competences at step t as $\mathscr{C} = \{C_1,\ldots,C_M\}$, such that $C_c = D'_c(\theta_{j-1},\theta_j)$, and $j \leq t$ is the last step such that minibatch $B_j \in T_c$. That is, for each $T_c \in \mathscr{T}$, C_c is the result of exponential smoothing over the weight variations of all the steps in which θ has trained on a minibatch from T_c.

We define a sampling function – noted 'sample*' in line 10 of the algorithm – with the following role:

- in the initial phase, it randomly samples any of the $T_c \in \mathscr{T}$ on which the system has never been trained on;
- then, when all tasks have been seen at least once, it samples a new $T_c \in \mathscr{T}$ based on the softmaxed per-task competence distribution $\sigma(\mathscr{C})$.

Additionally, we introduce hyper-parameter α in order to compare the importance of previous weight variation versus the current one (line 8). However, after empirical analyses, we found that the best results were obtained with $\alpha = 1$ (see Section 7.5.3).

In this study, for each MNMT system, we compare three methods: first, we train a model on multilingual batches, by upsampling all the tasks until they are the same size and then *shuffling* them. Second, we apply a cyclical *alternation* of monolingual batches for each task, which results in the model being trained the same amount of time on all tasks. Third, we apply our *self-paced method* as described in this section.

Algorithm 1: Self-paced scheduling algorithm for MNMT using the variation of model weights.

Require: tasks $\mathcal{T} = \{T_1, .., T_M\}$, steps s

$T_c \leftarrow T_1$;
for $t \leftarrow 1, \ldots, s$ **do**
    Sample minibatch B_t from T_c;
    $\theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta_t} L_{B_t}(\theta_t)$;
    if changedTask **then**
        changedTask $\leftarrow$ False;
    end
    else if $D'_c(\theta_{t-1}, \theta_t) < \alpha D'_c(\theta_{t-2}, \theta_{t-1})$ **then**
        $C_c \leftarrow D'_c(\theta_{t-1}, \theta_t)$;
        $T_c \leftarrow$ sample$^*(\mathcal{T} - \{T_c\})$;
        changedTask $\leftarrow$ True;
    end
    $t \leftarrow t + 1$;
end

In training, we use the the 'noam' schedule (Vaswani et al., 2017, Eq. 3) to compute the learning rate, which increases linearly from zero during the warmup steps, and afterwards decays proportionally to the inverse square root of the current step. Although the variation of weights throughout the entire training is strongly influenced by the learning rate schedule (Figure 7.2), we find that when comparing the smoothed weight variations between two consecutive steps, the influence of the learning rate variation is negligible.

7.4 Data and Systems

7.4.1 Corpora

We experiment on a subset of the multilingual TED corpus (Qi et al., 2018). As in previous multilingual studies (Neubig and Hu, 2018; Wang et al., 2019), we select four pairs of LRL-HRL, such that each pair is related, but not across pairs. We show them in Table 7.1, with the goal of translating them into English (EN).

LRL	train	dev	test	HRL	train
Belarussian (BE)	4.51k	248	664	Russian (RU)	208k
Azerbaijani (AZ)	5.94k	671	903	Turkish (TR)	182k
Galician (GL)	10.0k	682	1.0k	Portuguese (PT)	51.8k
Slovak (SK)	61.5k	2.2k	2.4k	Czech (CS)	103k

Table 7.1: Data sizes for pairs of LRLs and HRLs.

7.4.2 Tokenization

As the data is already tokenized, we directly apply Byte Pair Encoding (BPE) (Sennrich et al., 2016b) for subword extraction and vocabulary construction. We learn a vocabulary by concatenating 10k random lines from each language in the training data, and upsample the LRL dataset if it has fewer lines. For experiments involving only one LRL and one HRL in the source, we learn a vocabulary of 10k subwords. For experiments involving all four LRLs and all four HRL in the source, our vocabulary has 32k subwords. To facilitate language identification, we prefix each line in a dataset with a tag indicating its language.

7.4.3 System Architecture

We use Transformer models (Vaswani et al., 2017) from the OpenNMT-py library (Klein et al., 2017) version 3.1.1. In all our systems we use the following default values of hyper-parameters from Transformer-Base: 6 encoder/decoder layers, 8 attention heads, label smoothing of 0.1, hidden layer of 512 units, and FFN of 2,048 units. We use Adam optimizer (Kingma and Ba, 2014) and a batch size of 10k tokens.

Moreover, we will compare systems trained with default regularization to systems using more aggressive regularization. The former consist of a dropout rate of 0.1, OpenNMT-py's scaling factor of 2 over the learning rate, 8k warmup steps, and no gradient clipping. For the latter, we increase the dropout rate to 0.3, the scaling factor to 10 and the number of warmup steps to 16k, and we re-normalize gradients if their norm is greater than 5. We obtain the values for the hyper-parameters with more regularization from our findings in Chapter 3.

Our 2-to-1 and 8-to-1 models have 59M and 93M of parameters respectively. We train all our models on 2 GPUs (GTX 1080 or RTX 2080) for a maximum of 26 hours. In this study we trained approximately 40 models.

7.4.4 Evaluation

For each system, we measure the BLEU score (Papineni et al., 2002) on the LRL test set using the SacreBLEU library[3] (Post, 2018) as well as the COMET score (Rei et al., 2020) using model `wmt22-comet-da`. We use bootstrap resampling from SacreBLEU to compute the 95% confidence interval around the mean of the BLEU score. We use a rolling ensemble of four (consecutive) checkpoints and select the best ensemble on the development set for the final translations.

[3] github.com/mjpost/sacrebleu
signature: `nrefs:1|case:mixed|eff:no|tok:13a |smooth:exp|version:2.3.1.`

7.5 Design Choices and Results

In this section, we perform a series of experiments to optimize our method presented in Section 7.3. Firstly, we study the effects of several metrics to measure weight variation in order to justify our choice of KL divergence (Section 7.5.1). Next, we perform various experiments in order to optimize our method, involving: the smoothing parameter w (Section 7.5.2), the importance of the previous weight variation α (Section 7.5.3), and training during the warmup steps only on the HRL, which simulates a pre-training regime (Section 7.5.4). Experiments in this section (except Section 7.5.1) are performed in a 2-to-1 setup (GL-PT-to-EN) using default hyper-parameters.

7.5.1 Weight Variation Metric

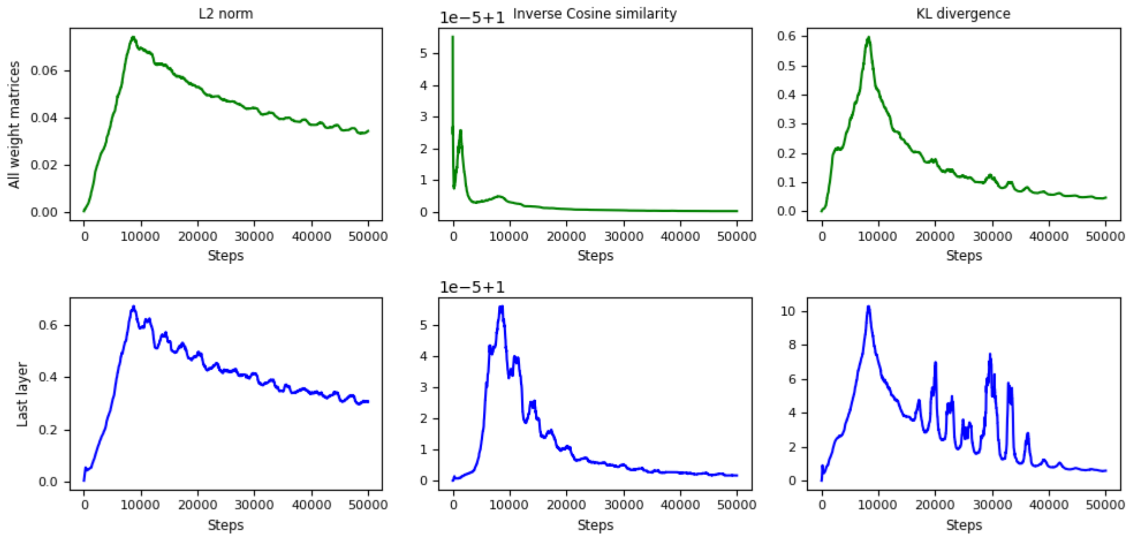

Figure 7.1: Comparison between three different metrics for model weight variation (Y axis): L2 norm, inverse cosine similarity, and KL divergence. For each of them we compare monitoring the average over all weight matrices and only the final output layer.

In order to measure the weight variation of a model between steps, firstly we train a model on SK-to-EN and compare measuring the average of all weight matrices versus the one of the last output layer only, and using as our metric one of: KL divergence, inverse cosine similarity, or L2 norm. We show in Figure 7.1 these six combinations, computing the variations of the weights every 10 steps (vertical axis) and performing a rolling average with a window size of 100. We can observe in all of them the effect of the learning rate schedule: during the warmup steps (which are 8k) the variation in weights increases rapidly, and afterwards begins to decay). Additionally, we also note a more regular curve when measuring the change across all matrices versus the last layer. We decide on using KL divergence as our measure due to it striking a balance between the irregularity of the inverse cosine similarity and the regularity of the L2 norm.

7.5.2 Setting of the Smoothing Weight

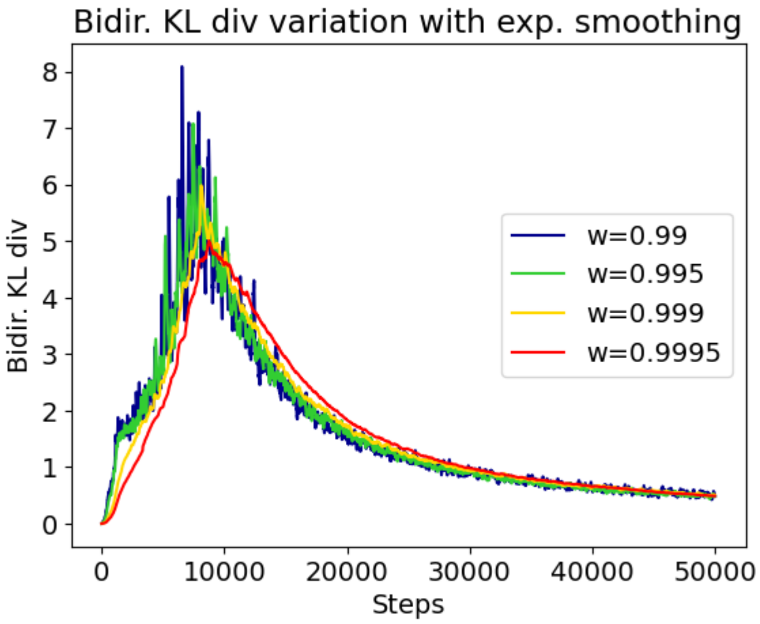

Figure 7.2: Evolution of the bidirectional Kullback-Leibler divergence for different values of the exponential smoothing coefficient w in an experiment on GL-PT→EN with dynamic sampling.

#	System	w	Steps	BLEU
	shuffled	-	44k	**27.49**
	alternation	-	44k	25.64
❙	*self-paced*	0.99	48k	25.76
❙	*self-paced*	<u>0.995</u>	<u>48k</u>	<u>25.92</u>
❙	*self-paced*	0.999	40k	**26.28**
❙	*self-paced*	0.9995	28k	25.42

Table 7.2: BLEU scores on the GL test set of our method with several values of the smoothing coefficient, w. We denote in bold the best result among our methods, and among the baseline ones, and we underline our chosen value for w.

In Figure 7.2 we show the average weight variation between all weight matrices when experimenting on a 2-to-1 setup (GL-PT-to-EN) using default hyper-parameters, with various values of the smoothing weight w. In Table 7.2 we present the corresponding scores, which we compare to *shuffled* and *alternation*. We can see that increasing w produces a more regular weight-variation curve, and also accelerates training without much loss in test score (see "Updates" column). Nonetheless, although some of the smoothing values produce better scores than a simple *alternation* of monolingual batches, none of them improve over the multilingual *shuffled* batches. We select a $w = 0.995$ for our main experiments as a balance between translation quality and training speed, while still maintaining a regular weight variation curve.

7.5.3 Importance of Previous Weight Variation

System	α	Steps	BLEU
shuffled	-	44k	**27.49**
alternation	-	44k	25.64
self-paced	0.9	48k	11.12
self-paced	0.95	48k	11.91
self-paced	1.0	48k	**25.92**
self-paced	1.1	40k	25.04
self-paced	1.2	44k	25.37

Table 7.3: BLEU scores on the LRL test set of our method with several values of the importance hyper-parameter α, with $w = 0.995$. We denote in bold the best result among our methods, and among the baseline ones, and we underline our chosen value for α.

Similarly, we also experiment with the value of α, a hyper-parameter to weight the importance of the previous weight variation when comparing steps t and $t-1$ (line 8 in Algorithm 1). In Table 7.3 we show the results of training with our selected value of $w = 0.995$ and various values for α. We observe best results without any additional weighting of the previous variation, and therefore we select for our main experiments $\alpha = 1$.

7.5.4 Training with HRL Warmup Steps

System	HRL warmup	Steps	BLEU
alternation	no	36k	24.92
alternation	yes	44k	**26.04**
self-paced	no	48k	25.92
self-paced	yes	48k	**26.16**

Table 7.4: BLEU scores on the LRL test set of our method when observing the role of HRL warmup, with $\alpha = 1$ and $w = 0.995$. We denote in bold the best result in the comparison methods, as well as in our method, and underline our chosen final technique.

Due to the effect of the learning rate warmup steps on weight variation during the first 8k steps of training, we also consider starting training the two methods with monolingual batches (*alternation* and *self-paced*) exclusively on the HRL, which simulates a pre-training regime. We show in Table 7.4 the effects of HRL warmup between each of these two methods. We observe *alternation* benefits significantly (+1 BLEU points) from HRL warmup, but our *self-paced* method much less noticeably. We do not consider HRL warmup to produce a positive balance between complexity and score improvement for our method, so we do not perform it in our main experiments.

7.5.5 Amount of Task Switches and Balancing

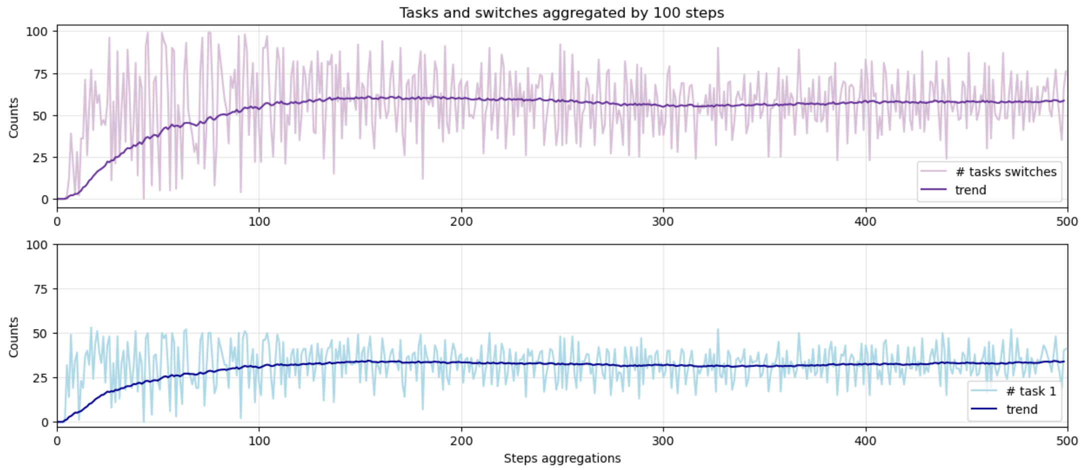

Figure 7.3: Amount of task switches and percentage of training on the LRL (*task 1*).

In Figure 7.3 we show the amount of task switches in training, averaged over 100 steps, where *task 1* is the LRL. We can see that on this GL-PT→EN model, after initial learning rate warmup steps, our method settles on one third of the training batches consisting of the LRL and two thirds on the HRL.

7.5.6 Hyper-Parameter Search

Finally, we search for the appropriate level of regularization to apply to our approach. We consider all four 2-to-1 MNMT systems described in Section 7.4.1. For each of the methods compared (*shuffled*, *alternation*, and *self-paced*) we train a model with default hyper-parameters and another model with more regularized ones (Chapter 3), which we described in Section 7.4.3, and present the average scores of all HRL-LRL-to-EN systems.

System	BLEU	COMET	Steps
shuffled	22.1	64.7	**36.0k**
+ regularization	**25.3**	**69.2**	38.5k
alternation	20.9	63.2	**35.0k**
+ regularization	**24.4**	**68.0**	39.5k
self-paced	21.2	63.8	49.0k
+ regularization	**24.6**	**68.5**	**36.0k**

Table 7.5: Average BLEU scores on the test sets of the four LRLs on 2-to-1 setups for the three methods for multilingual training, with standard and increased regularization (best scores in bold).

The average BLEU scores over the four LRL tests sets of each model are shown in Table 7.5.

Training with more regularization improves all three methods by 3 to 3.5 BLEU points. On this 2-to-1 setup we obtain better results when training with multilingual *shuffled* batches, and a small improvement of *self-paced* versus an *alternation* of monolingual batches. All regularized models methods reach their highest BLEU score at a very similar number of steps, although when training with more aggressive regularization, we observe a noticeable improvement in speed in the *self-paced* method. Finally, we note that the more regularized models, using the hyper-parameters that we proposed in Chapter 3, improve over the scores of previous studies on the same data (Neubig and Hu, 2018; Aharoni et al., 2019; Wang et al., 2020c).

7.5.7 Comparison of 2-to-1 and 8-to-1 Training

Previously we have empirically determined the best hyper-parameters for our system and observed that increased regularization improves scores. In Table 7.6 we present the scores of models trained on the four 2-to-1 setups (presented in Section 7.4.1) by order of increasing absolute size of the LRL-EN dataset, as well as an 8-to-1 setup, which includes all the tasks. As in Section 7.5.6, we compare three methods: *shuffled*, *alternation*, and our *self-paced* method.

Task	Method	2-to-1 Training			8-to-1 Training		
		BLEU	COMET	Step	BLEU	COMET	Step
BE→ EN	*shuffled*	**21.7** (±1.3)	**63.8**	48k	**20.0** (±1.4)	61.4	**64k**
	alternation	19.8 (±1.2)	61.5	44k	18.7 (±1.3)	61.3	120k
	self-paced	20.5 (±1.3)	62.8	**32k**	19.7 (±1.3)	**61.8**	128k
AZ→ EN	*shuffled*	**15.6** (±1.0)	**66.0**	**32k**	14.3 (±1.0)	**64.4**	144k
	alternation	14.4 (±1.0)	63.9	44k	**16.6** (±1.0)	62.9	**140k**
	self-paced	14.5 (±1.0)	64.9	48k	14.4 (±1.0)	64.0	150k
GL→ EN	*shuffled*	**30.2** (±1.2)	70.9	50k	**31.9** (±1.3)	**72.8**	**92k**
	alternation	30.0 (±1.2)	71.0	50k	30.4 (±1.2)	71.3	136k
	self-paced	**30.2** (±1.1)	**71.2**	**44k**	30.7 (±1.2)	72.0	150k
SK→ EN	*shuffled*	**33.6** (±0.8)	**76.2**	24k	**33.9** (±0.9)	**75.4**	**32k**
	alternation	33.4 (±0.8)	75.3	**20k**	31.8 (±0.9)	73.5	144k
	self-paced	33.3 (±0.9)	75.3	**20k**	31.9 (±0.9)	74.0	128k
Avg. of the four LRLs	*shuffled*	**25.3**	**69.2**	39k	**25.0**	**69.0**	**83k**
	alternation	24.4	68.0	40k	24.4	67.3	135k
	self-paced	24.6	69.0	**36k**	24.2	68	139k

Table 7.6: Results of the three methods compared on four 2-to-1 setups and an 8-to-1 setup, as well as the number of steps (Step) necessary to obtain the highest scores. We use our stronger regularization hyper-parameters (as in Table 7.5), and denote in bold the best score in each metric for each task.

Introducing more source languages does not improve scores in either of the three methods for MNMT training, although we observe a small negative effect on higher-resourced LRLs, which has been reported previously (Neubig and Hu, 2018; Aharoni et al., 2019). More specifically, the average of all three methods decreases or remains the same with going from a 2-to-1 setting to an 8-to-1 setting, although for *shuffled* we can see that the two higher-resourced LRL (GL and SK) benefit from more source languages, +1.7 and +0.3 BLEU points respectively. *Self-paced* tends to perform better than *alternation* on 2-to-1, but is more severely affected on an 8-to-1 setup: *alternation* improves on two languages when training on an 8-to-1 setup (AZ and GL, +2.2 and +0.4 BLEU points), while *self-paced* only on one (GL, +0.5 BLEU points), and the change from a 2-to-1 setting to an 8-to-1 setting on the average across the four languages for the *alternation* method is 0 and -0.7 , as measured by BLEU and COMET respectively, and for *self-paced* it is -0.4 and -1. Both of these methods, which rely on monolingual steps, clearly underperform with respect to *shuffled*, which is trained with multilingual batches.

Additionally, in the 2-to-1 case, the difference between *shuffled* and *self-paced* decreases as the absolute size of the LRL→EN dataset increases, but in the 8-to-1 case, the difference increases as the dataset size also increases (1.2, 1.1, 0.0, 0.3 BLEU points and 0.3, -0.1, 1.2, 2, respectively, with the following LRL sizes: 4, 6, 10, and 31k lines).

This indicates that with a small amount of training tasks, the more available data, the less important the sampling method is, but with many training tasks a *self-paced* selection of the balancing of the data becomes more important for lower-resourced datasets.

Regarding convergence speed, we measure the amount of steps that each model requires in order to reach its highest BLEU scores. We observe that all three methods train in nearly the same speed on the 2-to-1 case, but on the 8-to-1 case we observe that training with monolingual batches, regardless of the balancing of the tasks, results in a much slower training. This is likely due to either the model forgetting what it learned the last time it trained on a given task, or to the monolingual steps resulting in weights that are less useful to the other tasks.

7.6 Analysis

In this section we will analyze why our *self-paced* method does not outperform training with multilingual batches uniformly sampled from all datasets (*shuffled*).

In Table 7.7 we provide the percentage of the total amount of steps that an 8-to-1 model should train on the HRL and LRL (rows 1, 2, and 3), and only on each LRL (rows 4, 5, and 6), depending on three different data balancing criteria. Firstly, we compute the amount of steps if we did not perform balancing of the data (rows 1 and 4), which is equivalent to the percentage of the total training data that each dataset accounts for. Secondly, when performing a uniform balancing, where all the datasets are balanced to be the same size, which is the balancing used by *shuffled* and *alternation* (rows 2 and 5). Thirdly, the balancing of the data that our *self-paced* method actually does, obtaining the results of the 8-to-1 column in Table 7.6.

We can observe that our method may not result in sufficient amount of training for neither of the LRLs (rows 4-6): the model trains on all of them for a smaller amount of steps than uniform scaling would reach (12.5% of the steps). Since 12.5% is the is the exact proportion of data on which *shuffled* is trained, obtaining the best results, our method likely should train for more on the LRLs. Overall, our *self-paced* 8-to-1 model dedicates 43% of its training to LRLs: this indicates that our competence measure is too strongly dependent on the amount of training data. In other words, if a task contains a small amount of data, then our model learns it easily, which according to our measure means that the smoother KL divergence between steps flattens out, and therefore our method considers that the task is well learned.

Data Balancing	**Be-Ru→En** E. Slavic	**Az-Tr→En** C. Turkic	**Gl-Pt→En** W. Romance	**Sk-Cs→En** W. Slavic
Hrl + Lrl				
no	33.9	30.0	9.9	26.2
uniform	25.0	25.0	25.0	25.0
self-paced	28.5	27.9	19.0	24.5
Lrl				
no	0.7	0.9	1.6	9.8
uniform	12.5	12.5	12.5	12.5
self-paced	12.2	10.2	9.5	10.9

Table 7.7: Percentage of the total amount of steps that an 8-to-1 model should train on each HRL-LRL pair and each LRL, according to three data balancing methods: i) the true proportion of the data (that is, no balancing), ii) a uniform proportion of the data, and iii) what our *self-paced* method actually does.

Furthermore, if we observe the first three rows of Table 7.7, we see that our method does not result in the training of the model being equally distributed between all the HRL-LRL pairs: an equal proportion would be 25% of the steps to train on each pair. Instead, our method trains for more on the Hrl + Lrl pairs that contain more data, and less on the lower-resourced ones (compare row 1 and 3). This further confirms our explanation that our competence measure relies too much on the size of the dataset of a given task, and is not an optimal metric to measure the generalization capabilities in a task of a model.

7.7 Conclusion

In this chapter we have presented a self-paced method to balance tasks in a many-to-one MNMT system by monitoring the average per-task weight variation across steps, with the objective of not over-training on tasks in which the model is competent, and better allocating resources to tasks in which the model is less competent. Our method carries only a very modest computational overhead. However, we have observed that a multilingual, uniform balancing of all tasks outperforms our method both on 2-to-1 and 8-to-1 setups. The effectiveness of a temperature-based data balancing strategy is yet to be explored.

A limitation of our method may be that measuring the weight variation between two consecutive steps might result in too small a value, with too many oscillations even with the use of smoothing. Additionally, we have shown that as the amount of training tasks increases, performing single-task steps is counter-productive, both in quality and in speed. In the future, we hope to extend our method to assemble multilingual batches based on the per-task weight variation in order to solve this issue.

8 Many-to-many Multilingual Low-Resource Training

In Chapter 6 we have discussed the use of various tasks as auxiliary tasks in a multitask setting. In Chapter 7 we have presented a self-paced many-to-one multilingual system, making use of translation of higher-resourced languages as an auxiliary task to improve the quality of low-resource systems. In this appendix we will present many-to-many multilingual systems, and show in particular that they outperform both default and stronger, regularized unidirectional baselines.

8.1 Data and Systems

We compare models in four directions: HSB↔DE and DE↔EN using two low-resource parallel datasets of 60k (HSB-DE) and 120k (DE-EN) lines respectively[1]. For each direction we train a baseline unidirectional system with recommended hyper-parameters per OpenNMT-py for middle and high-resource translation, as well as a unidirectional system that has more aggressive regularization, which we proved to be beneficial on very low-resource datasets (Chapter 3). The hyper-parameters for our optimized systems are the ones obtained for our best-performing models in Chapter 3[2].

Additionally, as we show in Table 8.1, for each of the two datasets we train a bilingual (bidirectional) system for each of the two sets of hyper-parameters. Finally, we also train trilingual systems in all four directions: HSB↔DE and DE↔EN. This means that the trilingual models are capable of HSB↔EN translation, but only zero-shot, since they have not been trained for it.

We train all models until convergence: the unidirectional models train for a maximum of 16 hours, the bidirectional models for 22, and the trilingual for 56 hours. For the trilingual model, we upsample by two the HSB-DE data, but since all models are trained until convergence,

[1] This data was presented in Chapter 3.

[2] We make public the configuration files that create these systems in the OpenNMT-py framework (github.com/AlexRAtrio/many-to-many).

Model Type	Data	Lines
HSB↔DE	WMT20 Low-res	60k
DE↔EN	NewsComm. v13	120k
HSB↔DE↔EN	WMT20 Low-res (upsampled) &	60k (120k)
	NewsComm. v13	120k

Table 8.1: The three types of multilingual models trained in this section (two bilinguals and a multilingual type) and the data used to train each of them. For each of these types of models, we train two, with different levels of regularization through their hyper-parameters as shown in Table 8.2.

scores remain comparable. For all cases, batches are sampled uniformly from all translation directions.

We use Google's SentencePiece [3], to learn a joint trilingual vocabulary for all cases, from the two datasets in Table 8.1. We train the tokenizer with character_coverage=1. We add language tags in the vocabulary for each target language (`<HSB>, <DE>, <EN>`). Although for training multilingual systems both the source and target sentences can be prepended with a language token (Tang et al., 2021), in our system only the source is tagged (in both training and testing), with a target language tag prepended to each sentence plus a whitespace, while the target always remains as untagged, following previous research in multilingual NMT (Johnson et al., 2017; Aharoni et al., 2019; Arivazhagan et al., 2019), and also more broadly in multitasking (Raffel et al., 2020).

8.2 Results

In Table 8.2 we provide the scores for all directions (evaluated on the WMT20 `devel` set for HSB-DE, and a sample of 1k lines of NC v.13 for DE-EN) as well as the comparable unidirectional baselines and optimized scores. For each model we evaluate the checkpoint that obtains the highest BLEU score on the dev set, and evaluate BLEU score and chrF on the test set and cross-entropy loss on the train set.

In all four directions we obtain best scores with the regularized hyper-parameters. In the lower-resourced directions (DE↔HSB) increasing the amount of multilingual parallel data without increasing also the regularization does not produce consistent improvements, and on the more resourced directions (DE↔EN) the improvements are small.

Specifically focusing on the more regularized models: for the HSB→DE direction we observe that the bilingual model is not able to reach the score obtained by the unidirectional model (although it still improves over the baseline), and the trilingual model reaches the highest score. We see very similar results on the opposite DE→HSB direction, and in this case the bilingual

[3] https://github.com/google/sentencepiece

Direction	HPs	Model	BLEU↑ (test)	chrF↑ (test)	Xent↓ (train)
HSB→DE	bl	`unidirectional`	42.19	68.27	**0.01**
		`bilingual-hsb↔de`	40.62	67.65	0.05
		`trilingual-hsb↔de↔en`	41.64	68.46	0.33
	reg	`unidirectional`	47.20	71.73	0.07
		`bilingual-hsb↔de`	45.71	70.77	0.17
		`trilingual-hsb↔de↔en`	**49.42**	**73.59**	0.67
DE→HSB	bl	`unidirectional`	40.47	66.87	**0.02**
		`bilingual-hsb↔de`	40.92	67.09	0.05
		`trilingual-hsb↔de↔en`	42.03	68.67	0.33
	reg	`unidirectional`	46.31	70.47	0.08
		`bilingual-hsb↔de`	46.81	70.61	0.16
		`trilingual-hsb↔de↔en`	**49.81**	**73.71**	0.67
DE→EN	bl	`unidirectional`	29.94	56.72	**0.16**
		`bilingual-en↔de`	32.16	58.63	0.92
		`trilingual-hsb↔de↔en`	31.02	57.96	0.39
	reg	`unidirectional`	35.77	61.27	0.67
		`bilingual-en↔de`	**36.06**	**61.83**	0.91
		`trilingual-hsb↔de↔en`	34.57	60.99	0.66
EN→DE	bl	`unidirectional`	21.94	53.93	**0.15**
		`bilingual-en↔de`	23.63	55.34	0.78
		`trilingual-hsb↔de↔en`	22.75	54.41	0.57
	reg	`unidirectional`	26.61	57.72	0.85
		`bilingual-en↔de`	**27.10**	**58.00**	0.87
		`trilingual-hsb↔de↔en`	26.07	57.40	0.66

Table 8.2: Results of the unidirectional and multilingual systems. In each of the four directions, for each model setup we compare two sets of hyper-parameters: a baseline, less regularized set (`bl`), and an optimized, more regularized one (`reg`). We show in **bold** the best scores in each metric for each of the four directions.

model reaches similar scores as the unidirectional one. We also observe that the trilingual model reaches very similar scores in both directions HSB↔DE as well, which shows that the regularization introduced by the third language and two additional directions is slightly more beneficial to the DE→HSB direction than the other one, since the baseline scores in DE→HSB are lower. This is unexpected, as having had more training in generating DE would reasonably result in higher scores on the HSB→DE direction.

Regarding the DE→EN and EN→DE directions, we observe a big difference between the baselines (8 BLEU points), which also holds between the two optimized models, the two directions in the bilingual model, and the two directions in the trilingual model. The improvements provided by each of the three non-baseline models are comparable in both directions. We observe that the trilingual model does not improve over the regularized unidirectional model in either of the two cases. We conclude that the addition of a third language (HSB, in this case) benefits more if it is a higher-resource one.

8.3 Discussion

The bilingual models and the trilingual model have remarkably higher loss values than their unidirectional counterparts. We hypothesize that the cause is the same as in Section 3.8: a more regularized model is more likely to optimize into flatter regions of the loss landscape, even if the loss of the weights is higher. Settling into flatter regions allows the model to generalize better, particularly when there is more difference between its training data and the unseen test data. In this specific case, we argue that the introduction of a multilingual task serves as a kind of regularization, as would any auxiliary task, which explains the increase of BLEU in the multilingual model. Similarly, the increased loss also indicates that the model is not wasting as much of its learning in modeling dataset-specific patterns. This also maintains our previous explanation that the smaller amount of training there is, the more regularization is needed, which explains why the trilingual model outperforms the optimized model for HSB→DE but not DE→EN.

As we can see, in each direction, the highest score is always achieved by a multilingual model, despite the already well-optimized hyper-parameters of the regularized unidirectional models. Experiments in Chapter 3 show that the optimized hyper-parameters used in the optimized models are close to the highest amount of regularization that the model can be trained with without lowering translation quality. The scores obtained here show that an additional source of regularization, in the form of multitask multilingual training, can introduce even more regularization than what can be obtained from the regularization factors (dropout, learning rate, warmup steps, gradient clipping), without the generalization ability of the model degrading. This shows that even if regularization coming from one source may be at its maximum (before model quality starts decreasing), overall regularization may still be possible if coming from an additional source.

8.4 Conclusion

In this chapter we have trained unidirectional, bilingual, and trilingual models for two low-resource datasets, in four translation directions. For each of them, we observed improved results by training with more regularized hyper-parameters (see Chapter 3). We show that, particularly when coupled with increased regularization, very low-resource systems benefit greatly from increasing the amount of languages they translate to, at least up to three (+7 and +9 BLEU points). More resourced (but still low-resource) systems also benefit from translating into several languages, although the best results are obtained when translating into two and not three languages (+6 and +5 BLEU points).

Future work should clarify the cause for this drop of improvements. We posit two options: i) additional training on a much lower-resourced dataset does not improve results (but translating into a third language would improve them, if dataset size was higher), ii) as the main dataset size increases, benefits from many-to-many training decrease.

Part III

Constrained Text Generation in a Low-Resource Genre: Regularization with Artificial Data

9 Constraining at Inference-Time for Targeted Generation[1]

This chapter describes a system for interactive poem generation, which combines neural language models (LMs) for poem generation with explicit constraints that can be set by users on form, topic, emotion, and rhyming scheme. LMs cannot learn such constraints from the data, which is scarce with respect to their needs even for a well-resourced language such as French. We propose a method to generate verses and stanzas by combining LMs with rule-based algorithms, and compare several approaches for adjusting the words of a poem to a desired combination of topics or emotions. An approach to automatic rhyme setting using a phonetic dictionary is proposed as well. Our system has been demonstrated at public events, and log analysis shows that users found it engaging.

9.1 Introduction

LMs, which are probability distributions over sequences of words or characters, have recently enabled the generation of fluent sentences and texts. However, controlling such models in order to generate specific text structures remains difficult. We propose solutions for constrained language modeling with external features such as form, topics, emotions and rhymes, and integrate them into a system for interactive poem generation, which enables the joint writing of a poem by a human and a computer. This presents an additional difficulty: the lack of available data that has been annotated with a variety of these features, which places this work in a low-resource setting. Our CR-PO system[2] leverages neural LMs to generate the initial draft of a poem in a form selected by the user, and then enters a cycle of joint human-computer co-editing, in which the user can set various parameters, according to which the computer modifies the current creation. Manual editing is also possible at any stage. The CR-PO system has been demonstrated at a public exhibition and other events at our institutions. The system

[1]This work was done in collaboration with Valentin Minder, Aris Xanthos, Gabriel Luthier, Simon Mattei and Antonio Rodriguez, and published in Popescu-Belis et al. (2022). I designed and implemented the first generation stage (combining a general neural LM with rules in order to constrain its generation to a desired form) and the module applying the rhyming scheme at the last stage of the process.

[2]CR-PO stands for *Création Poétique Assistée par Ordinateur*, i.e. computer assisted poetical creation in French.

runs autonomously on an small form factor PC with a 32” touchscreen, with robust and explicit graphical interfaces.

The goal of this chapter is to present the poem generation system and our approaches to constrain neural LMs according to different dimensions that characterize poetry. We discuss previous related work in Section 9.2 . We present a functional view of CR-PO in Section 9.3, along with the setting and hardware. We introduce the third party software and the data used to build LMs in Section 9.4. Our solutions for constrained text generation are presented in Section 9.5, regarding the form of the poem (stanzas and lines), the adaptation of the poem to given topics or emotions, and the rhyming scheme. We present some sample outputs in Section 9.6. We present in Section 9.7 several actions we took towards the evaluation of CR-PO.

9.2 Related Work

As we saw in Section 2.6, the challenges of training a LM for NLG in a low-resource setting are similar to those of low-resource NMT. Poetry generation is a specific area of NLG, where training data is scarce. We will now present some other studies related to our study in this current chapter and in Chapter 10.

Hopkins and Kiela (2017) train a large recurrent neural LM with LSTM units directly on the phonetic encoding of poems (1.5 MB of text from https://www.sonnets.org), and constrain the form with a finite state machine. In their study, human judges were not able to distinguish machine-generated poems from human-authored ones. “Deep-speare” is a system for sonnet generation in Shakespearean style, which captures especially rhythm and rhyme, with a fixed form (Lau et al., 2018). The model also uses a bidirectional RNN with LSTM units. The results of evaluation by experts show that while constraints on form can be quite easily satisfied, readability and substance still hamper machine-generated poems. More recently, Wöckener et al. (2021) used conditioned RNNs in a system that is able to learn stylistic features from the data, such as length and sentiment, although not rhymes. Moreover, the authors show that models such as GPT-2 struggle with rhymes as well.

For Chinese, one of the earliest poetry generation systems using RNNs was proposed by Zhang and Lapata (2014), starting from user-provided keywords and generating a quatrain line-by-line, with a convolutional sentence model and pre-defined line lengths and tonal patterns. Previously, as done e.g. by Yan et al. (2013), the numerous constraints of Chinese poetry were solved through numeric optimization algorithms. The task of poem generation can be additionally constrained: for instance, Yang et al. (2019) study the problem of generating a poem from prose, an approach that allows users to convey more precise meanings than when seeding the poem with keywords only. Using a variational encoder and adversarial training, the system designed by Li et al. (2018) generates a Chinese poem given the title as a representation of its topic. They evaluate their poems in terms of topic consistency, fluency, meaningfulness and “poeticness.” Additional constraints can be imposed: for instance, to generate an acrostic poem where the first letters of the lines spell a given word, Agarwal and

Kann (2020) use a left-to-right 3-layer RNN with LSTM units and a separate rhyming model.

Turning now to interactive systems, ASPERA is an expert system for prose-to-poetry conversion in Spanish (Gervás, 2001). The rule-based system gathers information from the user about the style and the content, and uses NLG techniques and an example-based approach to generate a poem, although collaborative creation does not seem to be possible.

PoeTryMe is a rule-based interactive poem generation system initially designed for Portuguese and later extended to Spanish and English (Gonçalo Oliveira, 2012; Gonçalo Oliveira et al., 2019). Developed over a long period of time, and also available through a web-based interface,[3] the system leverages its grammatical and semantic knowledge to offer capabilities for word-level or line-level modifications in terms of number of syllables, "surprise" level, or keywords. Although many default forms are possible, there are no direct ways to express topics or emotions, and the fluency in English appears to be limited by the smaller amount of knowledge available to the system.

Poem Machine (Hämäläinen, 2018b,a) was developed for Finnish and was focused from the start on assisting primary school children in creating poetry as a follow up to previous experiments (Kantosalo et al., 2015). To cope with the complex morphology of Finnish, the system uses finite state transducers and a semantic network. Predefined forms and topics are proposed, and an assistance with rhythm is provided too. Similar to what we observed when children used CR-PO, Poem Machine helps to teach the constraints of poetic forms and makes experiments with poem creation more entertaining for children.

Hafez was one of the first systems to combine interaction with deep neural LMs (Ghazvininejad et al., 2016, 2017). The system proceeds in the opposite order of CR-PO: it first asks the user for a number of parameters, including sample words, sentiment, and repetitiveness, then it formulates internally the constraints as a set of transducers, and uses a word-based RNN LM to generate a poem (a quatrain) satisfying these constraints. User satisfaction was shown to increase when learning the initial parameters from previous interactions. Our system makes a different use of the LM: we use a character-based LM which better captures the grammatical inflections of French regardless of the forms seen during training, and we use also LMs to replace mismatching words. Moreover, CR-PO can be run with acceptable speed (less than 5 seconds) on a small computer witout a GPU, which makes our hardware easy to set up.

Van de Cruys (2019, 2020) proposed a RNN encoder-decoder architecture with attention, with GRUs, for English and French poems. The model is trained to generate a line of poetry given the preceding one, with a decoder part that models the new line in reverse order. The advantage of starting from the last word, as for Hafez, is that it can be sampled with a probability distribution that incorporates rhyming constraints, using a rhyming dictionary similar to ours, with an additional bias to avoid repeating the consonant group preceding the final vowel [+ consonant]. In the experiments, the ABAB CDCD pattern is always used. Human judges ranked a set of 40

[3] https://poetryme.dei.uc.pt/ – see also `@poetartificial` on Twitter.

generated poems almost as high as human ones on several parameters. No scores are provided for rhyming alone, likely because it is nearly perfect given the architecture, but the rhyming component improved scores of ‘poeticness’ and human-likeliness.

PoeLM (Ormazabal et al., 2022) uses a decoder (GPT-style with 350M parameters) to learn rhythm and syllables from a large corpus of prose in Spanish and Basque. Input text is segmented into phrases, and for each set of phrases of a sentence a set of tags is prepended to the sentence, e.g. `<LEN:11><END:ura>` for an `11`-syllable phrase finishing with ‘-ura’. PoeLM learns these control tags and can leverage them to generate lines of poetry of desired length and endings. However, as the model does not learn rhyming rules (i.e. identity of syllables) but only identifies actual syllables, poem generation must start by specifying exactly the ending syllables of each line. Evaluation is done by completing the initial line of human poems with PoeLM, and then asking human judges which version they prefer.

ByGPT5 (Belouadi and Eger, 2022) is a character-level Transformer-based decoder, with generation conditioned on rhyme, meter, and alliteration. The model is initialized on the decoder of ByT5 (Xue et al., 2022), trained on large amounts of data, and then fine-tuned on a machine-labeled corpus of pseudo-quatrains in English and German, separately. Meter and rhyme are evaluated with classifiers trained on labeled data. Overall, according to automatic and human measures, ByGPT5 produces better results than ByT5 and subword-level models such as GPT-2 and mT5.

9.3 Overview of the System

9.3.1 Functional Description

CR-PO addresses the following research questions regarding text generation using LMs:

1. While powerful LMs are now available for text completion tasks in well-resourced languages, how can a LM be trained to generate a specific text genre if resources are scarce?
2. How can constraints on form (lines and stanzas) be applied to the unstructured output of auto-regressive LMs? For instance, given the relative scarcity of sonnets, [4] hence the difficulty of learning such a form from data, how can it be imposed on LM-generated text?
3. How can the problem of cold start be addressed? I.e. text generation ex-nihilo, not text completion.
4. As poems convey topics and emotions, how can a human steer a LM to include one or more topics into a poem, without writing explicitly the beginning of the poem?
5. How to design a system that can function autonomously for a long time in a public exhibition?

[4] A poem with two quatrains followed by two tercets.

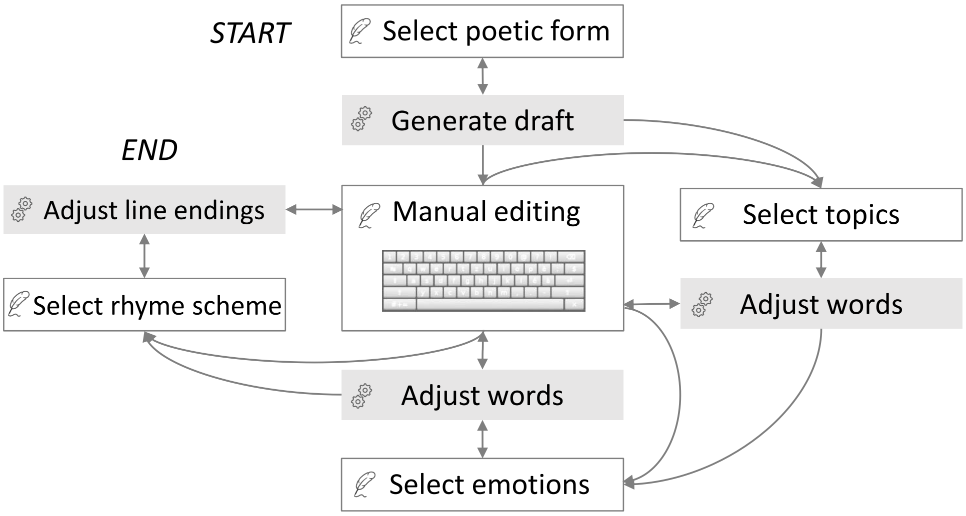

Figure 9.1: Functional schema of the CR-PO system. The creation of a poem proceeds clockwise. The quills in the white boxes indicate actions of the human user, and the cogwheels in the gray boxes indicate actions of the system's back-end. Manual editing is possible at any stage. Figure reproduced from Popescu-Belis et al. (2022).

The CR-PO system attempts to answer these questions, mostly through its poem generator presented in Section 9.5 below. To understand the entire system, a functional description is shown in Figure 9.1. The stages of the creation of a poem are the following ones:

1. The user selects the intended form of the poem, among four predefined options: a quatrain (four lines), a sonnet, a haiku, or free form (3–5 lines of 32–52 characters). Although the generator adapts to any number of stanzas and lines, we found it simpler for the user to choose among a small number of fixed options.
2. Using a general-domain LM trained on French poems, the system generates a first draft respecting the selected form (see Section 9.5.1).
3. The user can edit the poem using a keyboard, for instance to correct mistakes, improve readability, or express their own creativity. Manual editing is possible after each of the system's contributions, as shown in the central box of Figure 9.1. The editor is shown in the Appendix of the publication (Popescu-Belis et al., 2022).
4. The user can select one or more topics using sliders, from a list of five topics labeled as love, art, nature, spirituality or life-and-death. The system then modifies the poem by adjusting some words to fit the desired topics (see Section 9.5.2).
5. Similarly, the user can select one or more emotions among happiness, sadness, or aversion. The system then modifies the poem accordingly.
6. Finally, the user selects a rhyming scheme among three possibilities offered in the interface (e.g. AABB, ABBA, or ABAB for a quatrain), and the system changes line endings

accordingly (see Section 9.5.3). This is the last stage, and the delivery of the final poem depends on the technical setup which we now present.

9.3.2 Implementation and Public Displays

As the focus of this chapter is the poem generator and the constraints on LMs, we list here only briefly the main implementation decisions that make the demonstrator operational. Currently, the generator and the GUI are in French, and are being ported to English.

1. CR-PO is implemented in Python, and the GUI uses the Kivy framework.[5]
2. The system runs autonomously on a small-form-factor PC with modest computational requirements.[6] Training of LMs is done separately on a workstation with two GPUs.[7]
3. Users interact with CR-PO through a 32” touchscreen, which is fixed on a stand together with the computer. The system cannot be stopped as long as the physical keyboard and mouse are hidden.
4. The stages of each poem are logged into a JSON file for further analyses, but we do not record all interactions with the interface.
5. For users to keep a memory of their poems, solutions vary according to the presentation setting, but all the following solutions preserve the privacy of the users:
    - The poems can be automatically uploaded to a website, and when users conclude their creation, its URL and an access code for their specific poem are displayed.
    - In the exhibition mentioned below, each completed poem was printed on a large plotter, displayed as a work of art in the center of the room.[8] Users could select parameters of the fonts used for printing.
    - The poems can be printed on a regular printer, if available.
    - As suggested on the final screen, users can take a picture of their poem using their smartphones.

The system has been shown at the following events:

- The Digital Lyric exhibition in Morges, Switzerland, in spring 2020, which showcased art works and devices demonstrating novel relations between poetry and technology.[9] Poems created by visitors were made available online.

[5] https://kivy.org/
[6] A Dell Optiplex 7060 with an Intel Core i5 3 GHz processor, 8 GB RAM and 128 GB SSD.
[7] nVidia GeForce RTX 2080 Ti 11 GB.
[8] Designed by Nicolas Baldran and David Héritier from the Center for Future Publishing in Geneva, https://www.centerforfuturepublishing.org.
[9] The exhibition was held at the Château de Morges, from February 14 to May 10, 2020. Called *Code/Poésie* in French, it was curated by Antonio Rodriguez (University of Lausanne) and Sarah Kenderdine (EPFL). For more information, see https://lyricalvalley.org/digital-lyric-exposition/.

- Open doors of the HEIG-VD in November 2021 and 2022, where CR-PO was presented at two workshops to 10–13 years old visitors.
- Internally, CR-PO is available to visitors of the Institute for ICT at HEIG-VD in its showroom.
- As a demonstration system in the 13th Edition of Language Resources and Evaluation Conference (LREC) in June 2022.

9.4 Language Models and Data

9.4.1 Neural Language Models

LMs are the key technology that enables the generation of raw text, which our system transforms into poems based on the user's constraints. For the initial generation, we selected character-level LMs in order to increase the variability of generated text, which potentially includes non-existing but plausible French words, but also to ensure better grammatical agreement in sentences, as not all word forms are seen during training. This is in contrast with word-based LMs, which have a limited vocabulary, but also with subword based ones, for which it is not entirely clear how to set the vocabulary size (actually, the number of merge operations) when training data is rather limited (corpora of poems presented below).

The LM toolkit we used for autoregressive (left-to-right) generation of the first draft of the poem is TextGenRNN,[10] an open-source implementation by Max Woolf of a character-based LM using recurrent neural networks (RNNs) with attention. Written in Python using TensorFlow, following an approach proposed by Andrej Karpathy,[11] TextGenRNN uses LSTM layers according to the early principles of sequence modeling with RNNs (Sutskever et al., 2011; Graves, 2013).

To generate text in a poetic style, we trained a LM on our entire collection of poems (14.45 MB of text, presented below) without additional data from prose. Indeed, on the one hand, we observed that the text generated by this general model is nearly always grammatical without the need for more data, and on the other hand, it is not clear what prose genres are suitable to train a model that generates poetry. We also trained topic-specific and emotion-specific models based on smaller datasets presented below. The advantage of character-based RNNs over more recent Transformer-based decoders such as GPT-2 (Radford et al., 2019) is the lower amount of training data needed to reach acceptable quality.

For adjusting words to topics and emotions, the replacement of non-matching words is better achieved by taking into account the left and right contexts of these words, and not by left-to-right generation (as shown by the results in Section 9.7). We experimented with a general LM

[10] https://github.com/minimaxir/textgenrnn
[11] https://github.com/karpathy/char-rnn

for French, namely CamemBERT (Martin et al., 2019),[12] which is an encoder model trained on a masked language modeling task with 138 GB of French text using the RoBERTa architecture (Liu et al., 2019)[13] itself an improvement of BERT (Devlin et al., 2019). We adapted CamemBERT for 20 epochs to the topic-specific datasets presented below, following the documentation from HuggingFace[14].

9.4.2 Data Collection

We obtained about 7.72 MB of French poems from https://www.poesie-francaise.fr/, with poems in the public domain, only used for training our LMs. We gathered more than 5,000 poems, mostly from the 19th century and before. We obtained a similar amount of French poems (7.73 MB of text) from Project Gutenberg[15] by downloading text-only versions of works by a set of authors who have mostly written poetry, again in the public domain (about 50 authors and 76 books). The overlap between the two corpora is smaller than 4%. In order to train TextGenRNN with both datasets, we cleaned the texts by removing characters outside the ISO-8859-1 set, any material not part of the poems (such as Project Gutenberg metadata, titles, author names, etc.), and any blank lines, as we do not expect the LM to learn forms from the data.

In addition, to train topic-specific and emotion-specific LMs, we gathered annotated corpora of poems. We used topic and emotions labels from https://www.poesie-francaise.fr/ – assigned by the creators of the collection – to obtain small datasets for five topics and three emotions, using a correspondence between these eight coarse-grained categories and the observed labels.[16] We augmented these datasets with a second collection of 19th century French poems with labels, from the University of Lausanne.[17] The data sizes for each topic and emotion are the following ones:

- Topics: *amour* (love, 136,557 words, 776 KB of text), *art* (97,522 words, 561 KB), *nature* (130,923 words, 775 KB), *spiritualité* (spirituality and religion, 95,169 words, 542 KB), *vie et mort* (life and death, 153,554 words, 886 KB).
- Emotions: *joie* (joy, 41,126 words, 229 KB), *tristesse* (sadness, 62,248 words, 352 KB), *aversion* (hate, 44,004 words, 253 KB).

These datasets are too small for training or adapting topic- or emotion-specific LMs. To augment them, we used them as training sets for classifiers, which we then applied to our

[12] CamemBERT-Base model from https://huggingface.co/camembert-base/.
[13] https://huggingface.co/docs/transformers/model_doc/roberta
[14] https://huggingface.co/docs/transformers/model_doc/camembert
[15] https://www.gutenberg.org/
[16] Annotating emotions in poems requires in reality a broader range of categories (Haider et al., 2020) but we were constrained here by the available labels.
[17] A 1,001 poem collection put together by Mélina Marchetti under the supervision of Antonio Rodriguez for the Digital Lyric exhibition.

entire collection of poems (15.45 MB). The classifiers are word-based Naive Bayes models: 5-way for topics and 3-way for emotions. Although they are quite imperfect, the benefits of larger data sets thus obtained exceed their drawbacks. The augmented datasets have the following sizes in MB:

- Topics: *amour* (3.3), *art* (1.0), *nature* (2.0), *spiritualité* (5.3), *vie et mort* (5.8).
- Emotions: *joie* (4.3), *tristesse* (8.2), *aversion* (4.9).

The quality of topic-specific and emotion-specific corpus augmentation has been evaluated by holding out a small portion (10%) of the training data (labeled poems) and testing classifiers on it.[18] We experimented with three different options: the type of model (naive Bayes, decision tree, or logistic regression), the representation of lexical features (Bernoulli, i.e. 'present' vs. 'absent', or multinomial, i.e. 'number of occurrences'), and the chunks used for classification (fixed number of lines, whole stanzas, or whole poems). Since the emotion-specific sub-corpora are not divided in stanzas, we consider chunks of at least seven lines stopping at the first punctuation mark. Specifically, we obtain the following F1 scores: *amour* - 0.45, *vie et mort* - 0.36, *nature* - 0.36, *art* - 0.42, *spiritualité* - 0.32, with a macro-average F1 score of 0.38. Overall, our experiments led us to select a naive Bayes classifier, with lexical features represented using the Bernoulli model, and splitting the data into stanzas.

9.5 Constrained Autoregressive Generation of Poems

9.5.1 Setting the Poetic Form

In the first stage, the character-based TextGenRNN LM generates a poem with the form chosen by the user. Any form can be generated, with any number of stanzas, lines per stanza, and lengths of lines, although in the current interface only four fixed possibilities are offered. The following parameters have been set empirically.

Sampling Probability of the LM

The general LM is made of an input layer of 40 units, an embedding layer of 100 units, and two concatenated bidirectional LSTM layers of 256 units each (128 in each direction). Attention is applied to the concatenation of the embedding layer and the two recurrent layers, and the softmax dense output layer has 89 units, which is the size of our character set V. This covers largely the French character set, with common punctuation signs. We use the default values on TextGenRNN, except for our use of bidirectional layers.

Once trained, the general LM provides a probability distribution over the character vocabulary V, conditioned on context, i.e. on the $N = 40$ previous characters noted c_{n-N+1}^{n-1}, with c_n being

[18] The sizes of test sets are : *amour* - 257, *vie et mort* - 226, *nature* - 206, *art* - 133, *spiritualité* - 132.

the character to generate. To sample character c_n from V using this distribution, we use the temperature parameter t available in TextGenRNN, which we set to $t = 0.4$. Such a value augments the highest probability values in the distribution, and thus makes the model more "prudent" when sampling. Formally, we sample from V with the distribution:

$$P'(c_n) = \frac{P(c_n|c_{n-N+1}^{n-1})^{1/t}}{\sum_{c \in V} P(c|c_{n-N+1}^{n-1})^{1/t}} \tag{9.1}$$

where $P(c|c_{n-N+1}^{n-1})$ are the probabilities of the LM. Furthermore, in our implementation, we consider a set F of forbidden characters from V that we disallow our system to generate at a given stage. The use of F will be explained below.

Due to the lack of context, the initial parts of most generated sequences are often ungrammatical, contain numerous repeated words, and lack variety – a phenomenon known as "cold start". To address this, we generate four lines that are not shown to the user, and only start displaying the generated poem after them, i.e. after feeding the LM these four lines as initial context.

Adjusting the Length of Lines

The next goal is to control the length of the generated lines, in characters. This can be set at any value, but in our implementation, we approximate the number of characters per syllable as 4 for French, and, for instance, for a 12-syllable line, we aim for 48 characters per line on average. We note that existing tools allow for labeling of French text into syllables effectively. For instance, Beaudouin and Yvon (1996) is a rules- and dictionary-based tool that transcribes a French poem into phonemes, in order to then be split into syllables, with which they annotate 80k verses. We choose to approximate syllables with just the number of characters, based on considerations of simplicity, and after empirically observing satisfactory results with it.

For each line, we first generate 85% of it, disallowing end-of-sentence punctuation to avoid sentence splits in the middle of the line. Then, we set the minimum length of the remaining part of the line to 2 characters, and the maximum length to 1.8 times the remaining length. For instance, for a line with 48 characters, we first generate 41 characters (85% of 48), and then allow ending sequences between 2 and 13 characters ($1.8 \times 15\%$ of 48).

The algorithm makes several attempts to generate a series of characters *ending with a punctuation mark* within the desired character limit. The attempts are made in the following order, and when one succeeds, the algorithm stops and appends the result to the initial part of the line.

1. Generate a sequence until the maximal ending length is reached.
2. Retry, disallowing the first character of the previous try to encourage diversity.
3. Retry, relaxing the constraint on minimal length.

4. Retry, accepting whitespace as punctuation.
5. Retry, relaxing the constraint on maximal length.

A newline character is appended to the finished line, and the line is appended to the context of the LM. The generation of the next line thus also takes into account the newline character, which drives generation towards a sequence resembling a line start as learned during training. Therefore, our lines are genuinely distinct poem lines, and not just sequences divided manually into lines. The result is post-processed as follows:

1. Remove duplicate whitespaces.
2. Fix whitespaces before and after punctuation.
3. Uppercase line starts.
4. Delete 25% of all punctuation marks at line ends, to avoid too many lines ending with punctuation.
5. Ensure stanzas end with hard punctuation.

Although the creativity of a character-based LM sometimes leads to interesting new words, we decided that for a public exhibition it was preferable to spell check the output before display. We use a French dictionary of 142,541 words[19] (New et al., 2004) and replace unknown words with their closest correct match using Python's SequenceMatcher from the Difflib library. We use NLTK[20] for tokenization.

9.5.2 Adjusting Poems to Topics and Emotions

The next two stages enable the user to adjust the words of the poem towards one or more desired topics, and then emotions. The principles and interfaces for topic and emotion adjustment are similar, and we present them together. As stated in Section 9.3, when designing the CR-PO system, we settled on five topics (love, art, nature, spirituality, life-and-death) and three emotions (happiness, sadness, aversion) that appear frequently in poems. The user selects the desired proportion of each topic in the poem (and then emotion) using the sliders of the interface. The values for the m topics (or emotions) are coded as a vector $\mathbf{w} = (w_1, \ldots, w_m)$ of m weights between 0 and 1.

Poem adjustment requires two operations: select the words to be replaced (9.5.2), i.e. those that do not match well with the desired topics or emotions, and replace them with words that match better (9.5.2). For each operation, topic- or emotion-specific LMs (Section 9.4.2) provide an obvious solution: words that have higher perplexities for these LMs than for the general LM should be replaced with words generated using these specific LMs. However, we

19 http://www.lexique.org/
20 https://www.nltk.org/

propose for each operation an alternative solution, which human evaluators have found to perform better (as presented in Section 9.7).

Word Selection

Using topic- or emotion-specific LMs, the baseline criterion for word selection is the likelihood of each word according to these specific models: words with the lowest values are good candidates for replacement (e.g., "happy" would likely score low with the "sadness" LM). In our implementation, we compute the difference between the weighted average (by **w**) of the likelihoods given by the specific LMs and the likelihood of the general LM, then rank all words by decreasing values, and select about 8% of the words that are at least 3 characters long.[21]

The second method for word selection uses *independence quotients (IQs)* following the approach of Egloff and Bavaud (2018). The IQ value Q_{ik} for word i and category k is the ratio of the observed count of word i in poems belonging to category k to its expected count assuming independence of words and categories. Formally:

$$Q_{ik} = \frac{C_{ik} \cdot C_{\bullet\bullet}}{C_{i\bullet} \cdot C_{\bullet k}} \tag{9.2}$$

where C_{ik} is the count of word i in the poems of category k and the '•' sign denotes summation over the corresponding index. The IQ values are non-negative, smaller than 1 if word i is under-represented in category k, greater than 1 if i is over-represented in k, and 1 if the count of i in k equals its expected count assuming independence. The IQ matrix $\mathbf{Q} = (Q_{ik})_{1 \le i \le n, 1 \le k \le m}$ for n words and m categories can be pre-computed.

The dot product $\mathbf{Q} \cdot \mathbf{w}^\mathbf{T}$ represents the fitness values of all words given a choice of topics (or emotions), which shows how closely the profile of IQs of each word i matches the used-defined weights **w** of the categories. A fraction of the words with the lowest fitness are then replaced.

We built the IQ matrices using labeled poems from https://www.poesie-francaise.fr/ with a mapping of their labels into our five topics or three emotions. We only considered nouns, verbs and adjectives as found by TreeTagger[22] (Schmid, 1994) and we excluded stopwords[23] and words appearing fewer than 5 times. This resulted in 8,146 word types for topics and 2,621 for emotions in the respective IQ matrices.

Word Replacement

We have tested two methods for generating word replacements: either with the character-level LM from TextGenRNN, trained from scratch with topic or emotion specific data, or with a word-level CamemBERT model fine-tuned using the topic or emotion specific data. When

[21] As TextGenRNN is a character-based LM, the likelihood of a word is the average of the character probabilities.
[22] https://www.cis.lmu.de/~schmid/tools/TreeTagger/
[23] From the list available at http://members.unine.ch/jacques.savoy/clef/frenchST.txt.

several topics (or emotions) have non-zero weights in **w**, we decide randomly which model to use to generate each character or word, based on probabilities derived from **w**.

As the first method uses a left-to-right RNN, only the left context of the word to be replaced can be considered. The replacement is generated character by character, forbidding punctuation for the first three characters, then allowing it (including whitespace) and stopping when a punctuation mark is generated. For the second method, once the specific CamemBERT model is drawn, we give it the left and right contexts of the word to replace and a mask token in the respective position, and we select randomly among the five words receiving the highest probabilities.

The 2×2 options for word selection (TextGenRNN vs. IQs) and replacement (TextGenRNN vs. CamemBERT) were evaluated by human judges, and results are summarized in Section 9.7 below.

9.5.3 Setting the Rhyming Scheme

In the final stage, the user can select a rhyming scheme to apply, represented using letters, e.g., 'AABB', 'ABAB' or 'ABBA' for a quatrain. The system selects which line endings must be changed, retrieves a list of candidate words from a dictionary, scores them with the general LM and selects the highest-scoring one with the same POS tag as a replacement. We use the same dictionary as in Section 9.5.1, as it includes the phonetic representation and POS tag of each word.

The main challenge is to exploit the phonetic forms of words to identify the rhymes, i.e. the ending sounds which must match across words. We formulate the following stages using regular expressions: (1) identify the final vowel of the word; (2) extract either the following consonant, if any, or the immediately previous ones, if any; (3) otherwise, extract the immediately previous vowels plus the previous consonant. All the rhymes are two or three sounds long, as exemplified in Table 9.1.

Although these rules produce imperfect rhymes, stricter ones were also tested, but they either did not find rhyming words, or found words that were too similar, e.g., singular vs. plural of the same words – perhaps due to the reduced dictionary size (150k words).

To apply a rhyming scheme to a poem, we keep the final word of a line if it is the first one in the rhyming scheme: e.g., with 'ABBA', the first two lines are not changed. We also store their endings in order not to repeat them below. Then, for each line ending to change, we search for at most five candidate words that have the same rhymes, with priority to words that have the same POS tag as the word to be changed. We score the candidates in context with the general LM, which is better than the specific ones, and choose the candidate with the highest score. If the user edits the poem, the same process is run based on the edited version.

Word	Rhyme	Word	Rhyme
endormant	m@	raisonnassent	as
chipez	pe	perturbent	yRb
plaisantions	tj§	béatement	m@
guet-apens	p@	soumettions	tj§
brouillaient	ujE	misait	zE
vallonnés	ne	observées	ve
fumant	m@	pugilistes	ist
envole	Ol	hydrophiles	il

Table 9.1: Examples of rhymes extracted from the phonetic representations of some words in the dictionary.

9.6 Sample Results

Three sample outputs of our system, respectively prompted internally with the strings "*Je rêve*", "*Être distrait*", and "*Ma tête dans les nuages*" (I dream, being distracted, and my head in the clouds, in French), are provided hereafter to illustrate its results. Put together, these outputs constitute CR-PO's participation to a poetry contest, on the same topic as the prompt strings. To improve quality, the three outputs were selected from a total of six. In these cases, the prompts provide enough context and diverse starting points to continue generation without the problem of a cold start, so we provide the text as it is generated directly after each prompt.

Je rêve des couleurs de la forêt en silence.
La terre s'accroche aux premiers pas de l'homme.
Le soir, dans tout ce qu'il trouve dans ces tomes
Sous les champs d'or de la fleur de la naissance !

Être distrait pour la chair sourde de son âme
L'espoir d'un chant sous le cinname ;
La fille du ciel s'abat, le soir, le temps est éclatant,
Et le printemps, et le soleil se clapotant
Et ses doigts d'or s'en vont au fond du soir.

Ma tête dans les nuages, les arbres noirs, en haut,
Et les parfums pleurants s'en vont en lui rendant les cieux.
Les fleurs de la saison descend de la terre,
A l'entour des vagues du vent, les ombres de l'enfance.

9.7 Evaluation

We performed several experiments with human users of the CR-PO system. First, we summarize an experiment comparing the 2×2 adjustment methods from Section 9.5.2. Then, we present statistics from the two public events where CR-PO was used.

9.7.1 Results from A/B Testing

We tested all four combinations of word selection and replacement methods presented in Section 9.5.2 with 15 automatically-generated poems presented to 13 users. Each user saw the initial poem and had to compare two outputs of topic or emotion adaptation, differing on one of the two stages, either word selection or word replacement. Users were asked to choose the best of the two outputs, with ties allowed, on the dimensions of topic relevance and fluency, by answering the following questions:

- Which output is closer to the indicated topic?
- Which is the most understandable output, i.e. the one that makes the most sense in French?

We present the answers in Table 9.2. The use of IQ values for word selection leads to poems that are clearly perceived as more topically-relevant than when using the LM probability difference. These poems are also perceived as more fluent, although the difference is lower. The smaller effect on fluency can be explained by the fact that the IQ score does not consider the context of words, which may lead to replace words that are particularly important for fluency. For word replacement, CamemBERT clearly outperforms TextGenRNN. This can be explained by the larger amount of training data in comparison with the RNN character-level models, which were trained from scratch, and by the bidirectional nature of CamemBERT. As a result, we chose IQ as a criterion for word selection, and CamemBERT for word replacement.

Question	Answer	Topic	Fluency
Word selection	TextGenRNN is better	17.3	24.4
	IQ values are better	**67.7**	**41.7**
	The two are similar	14.9	33.9
Word replacement	TextGenRNN is better	14.3	3.1
	CamemBERT is better	**68.3**	**79.2**
	The two are similar	17.5	17.7

Table 9.2: Answers of human judges (%) for each method of word selection and replacement.

9.7.2 Results from Use by the General Public

As stated at the end of Section 9.3, CR-PO was presented to the general public on two occasions, and is also available in the ICT Showroom at HEIG-VD. Upon the first occasion, at the Digital Lyric exhibition (see footnote 9), we collected about 100 poems in 13 days, before the exhibition was closed due to the Covid-19 pandemic. The interactions with CR-PO were logged in a central database, which will be analyzed in the future, when more poems are collected.

CR-PO was presented at a workshop for young visitors, aged 10–13, at the HEIG-VD Open Doors in November 2021. After a discussion about artificial creativity and an overview of CR-PO, each visitor could experience the co-creation of a poem. We gathered 42 poems from 25 visitors, who all felt quite engaged by CR-PO and tried all its functions. Table 9.3 shows the average number of interactions and calls to the editing window for each stage. An average of 1.62 interactions at the first stage means that some users started over and asked for a new first draft, while 0.31 manual editing at this stage means that on average, 1 user out of 3 modified the first draft using the editor. The topic and emotion adjustments were each tried once per poem, on average, with manual edits in 1/4–1/3 of the times. Despite coming at the end, the automatic generation of rhyming schemes also raised interest from users.

	AVG	STD
1. Generation of 1st draft	1.62	1.10
Manual editing	0.31	0.47
2. Topic adjustment	1.05	1.23
Manual editing	0.36	0.48
3. Emotion adjustment	0.95	0.91
Manual editing	0.26	0.45
4. Rhyming scheme	1.26	1.67
Manual editing	0.38	0.49

Table 9.3: Average number of interactions with the CR-PO system, for each stage, at the 2021 Open Doors of HEIG-VD (25 visitors, 42 poems).

9.8 Conclusion

We presented CR-PO, a system for interactive poetry generation in French, putting forward solutions that combine neural LMs and rule-based constraints on form, topic, emotion, and rhyming scheme. Together with the hardware and the graphical interfaces, we achieved a fully functional, robust system, which was left without supervision in a public exhibition. The system also represents a platform which can be extended through future developments, such as porting it to English, improving the management of rhythm, and allowing users to provide seed words.

Overall, the main observation made when combining LMs and explicit constraints is that the poems generated by such an approach lack a high-level meaning conveyed by several lines or even stanzas, for instance as complex visual scenes or short narratives. This is a shortcoming of most poem generation systems based on deep neural LMs, and only extremely large LMs such as GPT-3 seem to be able to overcome it for plain text. Therefore, finding solutions that increase the coherence of texts generated by smaller LMs is a promising research question.

9.9 Perspectives

The quality of our system could be improved by leveraging additional models, or by extending our system as a multilingual model: language models benefit from their representation of several languages, and the already existing poetic datasets in several languages allows for training such a model. Following this, a direct evaluation of the system should be performed, in terms of perplexity, lexical diversity, syntactic analysis, and other aspects of quality.

10 Domain-Specific Data Augmentation[1]

Poem generation with language models requires the modeling of rhyming patterns. In this chapter we propose a novel solution for learning to rhyme, based on synthetic data generated with the rule-based algorithm presented in the previous chapter (see Section 9.5.3). The algorithm and an evaluation metric use a phonetic dictionary and the definitions of perfect and assonant rhymes. A GPT-2 English model with 124M parameters was fine-tuned on 142 MB of natural poems and find that this model generates consecutive rhymes infrequently (11%). Then the model is fine-tuned on 6 MB of synthetic quatrains with consecutive rhymes (AABB) and obtain nearly 60% of rhyming lines in samples generated by the model. Alternating rhymes (ABAB) are more difficult to model because of longer-range dependencies, but they are still learnable from synthetic data, reaching 45% of rhyming lines in generated samples.

10.1 Introduction

Although in recent years the quality of texts generated by language models (LM) as improved dramatically, LMs are not yet able to regularly and correctly generate text constrained by explicit rhyming patterns. In this chapter, we focus on the first property and propose a method to adapt an LM so that it generates rhyming verses, with modest computing requirements. We start from an unconstrained autoregressive LM, in our case GPT-2, which we fine-tune first on a poetry corpus of about 120 MB to improve its style (Section 10.4). In Section 10.2 we show how unconstrained text generation with this system does not result in a good quality of text. We design a rule-based system which modifies text generated by the LM so that it obeys a given rhyming pattern while retaining acceptable fluency, and we generate two datasets of 160k lines (6 MB) each with the AABB and ABAB patterns (Section 10.5). We further fine-tune the LM on these synthetic datasets in order to generate rhyming verses with the respective patterns, thus showing that they can be learned by a moderately-sized LM (Section 10.6).

[1]This work was done in collaboration with Bastien Bernath, Étienne Boisson, Teo Ferrari, Xavier Theimer-Lienhard, and Giorgos Vernikos, and published in Popescu-Belis et al. (2023). I contributed to the design of the experiment and the design and implementation of an automatic measure of rhyming, while the implementation of the learning experiment was done by three EPFL MSc students (3rd, 4th and 5th authors).

We also introduce a rhyming metric (see Section 10.3) based on an English rhyming dictionary, and use it throughout the study to count the proportion of perfect and assonant rhymes generated by a model. We find that this is very low (11%) for the LM fine-tuned on natural poetry with variable rhyming patterns, but increases to around 60% when the LM learns only the AABB pattern from synthetic data. The ABAB pattern is more challenging, but can still be learned, reaching around 45% rhyming lines. In the conclusion (Section 10.7), we discuss some issues related to the integration of the rhyming LMs into an existing, operational system for interactive poetry generation.[2]

The contributions of this chapter are the following (my own are stated explicitly):

- a metric computing how many lines have perfect or assonant rhymes that conform to a given pattern in English;
- I show the negative effects that result from fine-tuning a LM on small amounts of data, which justify our main strategy in this chapter;
- I designed a rule-based algorithm to generate rhyming lines of a given pattern, based on a GPT-2 LM fine-tuned on poetry;
- it is demonstrated that even medium-scale LMs can be fine-tuned to learn a rhyming pattern from machine-generated poems;
- evidence is provided that local rhyming patterns are more easily learned than those implying longer-range dependencies.

10.2 Effects of Fine-Tuning a LM on Small Amounts of Data

In this section we study the negative effect that fine-tuning on a small amount of data has on language models in terms of the form diversity of the generated text, and how this motivates our research on this chapter to generate poetry outputs with a better form. Similarly, we also present an analysis on the problems on textual diversity, although we will not tackle them in this work.

In particular, we compare the texts generated by *GPT-2* and GPT-2 when fine-tuned (*GPT-2 ft*) on the Gutenberg Poetry Corpus[3] (with around 3 million lines). To these two machine-generated texts we also compare the human-generated Gutenberg Poetry Corpus (*H-Gutenberg*), and the English side of the English-German News-Commentary v.13 (*H-News*, as an example of non-poetic text)[4].

To obtain each of the two machine-generated texts, we start text generation from scratch 300

[2] Source code available at github.com/heig-iict-ida/crpo.

[3] github.com/aparrish/gutenberg-poetry-corpus

[4] We also compare a sample of the full Gutenberg Corpus, which is mainly comprised of non-poetic data, but we only observe a significant difference with *H-Gutenberg* in regards to its lexical diversity, which is more similar to non-poetic data like *H-News*.

times, in order to minimize the effect of the lack of initial context and successive consequences from this. For each of the 300 samples we generate around 1k tokens. In total, we obtain slightly less than 1MB of data. For the two human-generated texts, we mimic this process by sampling randomly 300 chunks of text, obtaining a similar size of text afterwards.

10.2.1 Effect on Form

Text	**Bytes**	**Lines**	**Sentences**	**Average sent. length**	**Average word length**
H-News	926k	6'600	6'426	25.80 (±12.82)	4.62 (±3.04)
H-Gutenberg	785k	19'200	6'101	28.16 (±23.62)	3.73 (±2.24)
GPT-2	811k	3'667	6'403	25.81 (±21.92)	4.02 (±2.90)
GPT-2 ft	868k	21'975	8'845	18.89 (±16.69)	3.21 (±1.70)

Table 10.1: Number of generated lines, sentences, average sentence length, and average word length of the four samples of corpora: human-generated non-poetic data (*H-News*), human-generated poetic data (*H-Gutenberg*), text generated by GPT-2 (*GPT-2*), and text generated by GPT-2 after it has been fine-tuned on human-generated poetic data (*GPT-2 ft*).

Firstly, in Table 10.1 we present size measures of the generated text. Column "Lines" is the result of splitting the text on naturally-occurring newline characters, and "Sentences" is the result of segmenting the entire text into actual sentences, regardless of newline characters present in the text.[5] We observe that the non-poetic human-generated text (*H-News*) obtains similar values in both cases, since it is formatted for sentence-level translation, with one sentence per line. The human-generated poems (*H-Gutenberg*) has a much larger number of lines, since every verse ends in a newline character, however these verses amount to a very similar number of sentences, showing that there is a syntactic continuity between verses. *GPT-2* generates more paragraphs, hence the larger number of sentences compared to lines. *GPT-2 ft* learns well to generate verses, like we see in *H-Gutenberg*. However, it generates much shorter sentences overall: while the human-generated poetic text has sentences that span for 3.15 lines, the machine-generated ones only span 2.48 verses.

GPT-2 ft also has problems generating longer sentences. Its average sentence length is 18.89 words, whereas its fine-tuning data (*H-Gutenberg*) averages 28.16 words. Non-poetic data, both human- and machine-generated has an average sentence length of ~25.8 words, and does not reach the average length of true poetic data. Likewise, *GPT-2 ft* does not learn the correct average word length (in characters) of its training data (*H-Gutenberg*): although it also seems to learn in its fine-tuning to generate shorter words, it reduces too much their length. Interestingly, we also observe the same behavior between the human- and machine-generated non-poetic data.

In Figures 10.1 and 10.2 we show the distribution of sentence lengths and word lengths

[5] We use `sent_tokenize` from the NLTK library.

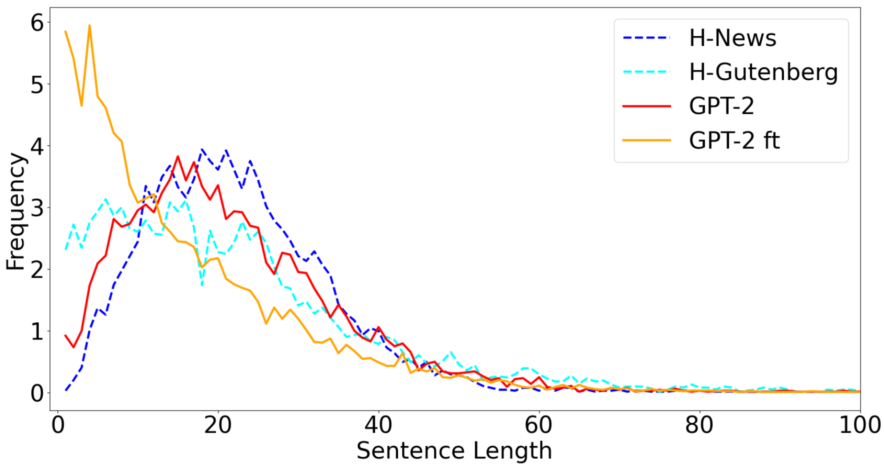

Figure 10.1: Distribution of sentence lengths (in words) for the four text samples compared in this section. Human-generated texts are in dashed lines, and machine-generated texts in continuous lines[6].

between the four texts, to further highlight the previously discussed issues. In the case of sentence length we observe that the system is biased towards very short sentences, and in the case of word lengths, although the human-generated poetic data (*H-Gutenberg*) does feature shorter words than human-generated non-poetic data (*H-News*), we can observe that text generated by *GPT-2 ft* increases this gap. In particular, regarding word length, *GPT-2 ft* generates a total of 27 types that are one character long, whereas *GPT-2* generates 75. The frequency of punctuation signs is higher in the samples generated by *GPT-2 ft*: for instance, "," is 3.15 percentage points higher, ";" is 1.07, "!" is 0.86, and "?" is 0.31. One-character alphabetic types like "I" and "A" are also more frequent (1.19 and 0.19 percentage points, respectively).

10.2.2 Effect on Textual Diversity

Although in this work we will only focus on improving the form generation of a LM fine-tuned on a small amount of data, we also observe a negative impact in its lexical diversity. In Table 10.2 we present textual diversity scores, in order to observe the lexical effects of fine-tuning on small amounts of data. We computed the numbers of tokens and types, Type-to-Token ratio (Templin, 1957), Yule's I[7] (Yule, 1944), and Jaccard-Diversity (JD)[8] (Wang and Wan, 2018).

We can see that the human-generated data features a larger TTR, and in particular the poetic data. While *GPT-2* shows a smaller TTR, *GPT-2 ft* obtains even worse results, although it has been fine-tuned on poetic data, which has the highest TTR.

[7]Yule's I is the inverse of Yule's Characteristic Constant, which measures lexical consistency by considering vocabulary repetition.

[8]We compute the average of the JD of all sentences, which is defined as the inverse maximal Jaccard similarity between each sentence in the text and all the rest.

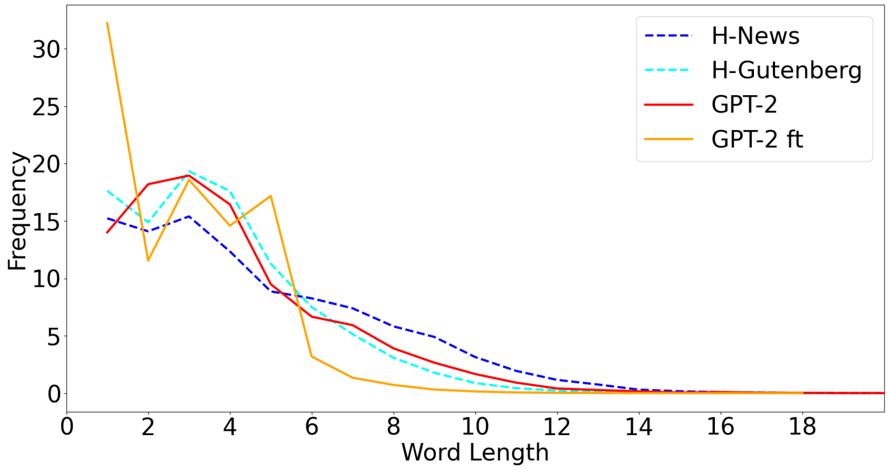

Figure 10.2: Distribution of word lengths (in characters) for the four text samples compared in this section. Human-generated texts are in dashed lines, and machine-generated texts in continuous lines.

Text	**Tokens**	**Types**	**TTR↑**	**Yule's I↑**	**JD↑**
H-News	165'789	16'359	0.10	0.09	0.78
H-Gutenberg	171'815	21'654	0.13	0.07	0.78
GPT-2	165'291	11'849	0.07	0.10	0.74
GPT-2 ft	167'115	5'412	0.03	0.06	0.66

Table 10.2: Textual diversity scores (TTR, Yule's I, and Jaccard-Diversity (JD)) of the four text samples compared. For all three metrics, higher values indicate more diverse text.

Between the two human-generated text samples we observe some discrepancy between TTR and Yule's I, but they have a similar lexical diversity, as JD confirms. JD shows a decrease of lexical diversity on *GPT-2*, but only moderately, whereas Yule's I assigns it a larger value. Both TTR and JD agree on a drastic loss of lexical diversity of the text generated by *GPT-2* of the same model after is has been fine-tuned on poetic data (*GPT-2 ft*), while Yule's I shows a more moderate loss. Since we know that the data used for fine-tuning the model (*H-Gutenberg*) features a "normal" lexical diversity, we may conclude that fine-tuning a LM on a comparatively small amount of data has a negative effect on its lexical diversity.

Overall, we have shown that fine-tuning a LM allows it to learn some features of the data: specifically, the splitting of true sentences into many lines (i.e., verses), and a shortening of the average word length. However, the fine-tuning appears to bias the model towards much shorter sentences and words, in comparison to either human-generated text or machine-generated text when trained with large amounts of data. Although it is possible that this is due to the model overfitting to the fine-tuning dataset, we do not have enough data to definitively conclude this. Similar observations have been made regarding the quality of generative tasks on literary text, like translation of literary texts, where machine-generated text lags behind human text on several dimensions, such as creativity (Guerberof-Arenas and Toral, 2022). Finally, fine-tuning on a small amount of data has strong negative effects on the lexical diversity of generated text. We conclude that in order to ensure that features that would be highly unlikely to appear naturally do appear in the generated text, we require either to augment the amount of data synthetically, or to constrain the generation of the text.

10.3 Measuring the Number of Rhymes

A criterion for measuring the number of rhyming verses is key for the present study. We present a metric that distinguishes between perfect rhymes, assonant rhymes, and no rhymes, using a rhyming dictionary derived from an English pronunciation dictionary. We test it on a corpus of human poetry annotated for rhyme and show that its accuracy is sufficient for use in this study.

10.3.1 Definitions of Rhymes

Following a widespread definition,[9] also adopted by Van de Cruys (2020), a perfect rhyme is the identity of the final vowel and consonant sounds of a word, starting with the first vowel of the last stressed syllable. An assonant rhyme is the identity of the final vowels in the last stressed syllable, but not of the ending consonant.

Since the addition of stress information would reduce the amount of available candidates for a rhyme, we simplify the definition of a rhyme between words w_1 and w_2 as follows, using the phonetic representation of each word $phon(w)$.

1. We have a *perfect rhyme* if $phon(w_1)$ and $phon(w_2)$ end with the same vowel followed by the same consonant(s), if any.
2. We have an *assonant rhyme* if $phon(w_1)$ and $phon(w_2)$ end with the same vowel, followed by one or more non-identical consonants.
3. Otherwise, the lines *do not rhyme*.

10.3.2 Construction of the Rhyming Dictionary

To apply the preceding definitions, and to generate rhymes according to them, we build a rhyming dictionary starting from the Carnegie Mellon Pronouncing Dictionary of English.[10] The dictionary contains pronunciations of 123,631 English words. Each word is associated with a series of phonemes coded using ASCII letters only, for example '`K AE M P EY N`' for the word 'campaign'.

[9] See e.g. rhymenow.com/types-of-rhymes.

[10] Freely available from svn.code.sf.net/p/cmusphinx/code/ trunk/cmudict/sphinxdict/cmudict_SPHINX_40.

We distinguish 15 phonemic vowels (e.g., 'AH', 'AW', 'EY', 'OY') and consider all other phonemes as consonants. To each word from the dictionary we associate two strings.

1. The last phonemic vowel and all the consonants following it (if any), to allow testing for perfect rhymes.
2. The last phonemic vowel only, whether it is followed or not by consonants, to allow testing for assonant rhymes.

Examples of entries in our rhyming dictionary are therefore ('campaign' → 'eyn', 'ey'), ('copy-codes' → 'owdz', 'ow'), ('vanilla' → 'ah', 'ah'), ('do' → 'uw', 'uw'), and ('wouldn't' → 'ahnt', 'ah').

To help with rule-based generation of rhymes, we create two dictionaries that invert the first one, for efficiency reasons. One has the strings defining the perfect rhymes as keys and the corresponding words as values – for instance ('eyn' → …, 'campaign', 'overtrain', 'plane', …) – and the other one has the strings defining the assonant rhymes as keys and the corresponding words as values. The first additional dictionary has 1,356 keys (word endings for perfect rhymes) and an average number of 91 words per key, while the second one has only 15 keys (the number of phonemic vowels) and an average of 6,507 words per key, ranging from 576 to 34,037.

		Our metric		
		Perfect rhyme	Assonant rhyme	No rhyme
Human annotation	Rhyming	27,174 (78.8%)	680 (2.0%)	6,628 (19.2%)
	Not rhyming	4,209 (1.3%)	25,163 (7.9%)	290,633 (90.8%)

Table 10.3: Confusion matrix for rhyming detection by our metric vs. human annotation.

10.3.3 Definition of the Metric

The proposed metric for rhymes follows from the definitions above, and makes use of the first dictionary. Given two words – the ending words of two lines of poetry – we compare their entries in the dictionary. If the first strings are identical, then we count a *perfect rhyme*. If they are not, we examine the second strings, and if they are identical, then we count an *assonant rhyme*. If not, then we consider that the words do not rhyme. The order of testing is important, because for words ending with a vowel, such as ('vanilla' → 'ah', 'ah') and ('Godzilla' → 'ah', 'ah'), both entries match, but we want to consider this as a perfect rhyme.

To apply the metric, the lines of the poem are first tokenized using NLTK's `word_tokenize()` function.[11] If a line finishes with punctuation, we discard it and examine the last word of the

[11] From www.nltk.org.

line. If the line ends with a contraction (such as 'wouldn't') we join back the two resulting tokens generated by `word_tokenize()`. If a word does not appear in the pronunciation dictionary, then we search for the most similar one in terms of string edit distance using the `get_close_matches()` function from the 'difflib' Python package (a time-consuming operation). We experimented with restricting the similarity search to the initial parts of words, because changing the end changes the rhyme, but did not observe significant differences when validating the metric.

10.3.4 Validating the Metric

We validated our metric on the Chicago Rhyming Poetry Corpus[12] which includes English poems annotated with their rhymes. For each poem, the annotation marks the last word of each line with an index number, and co-indexes rhyming words. For instance, a three-line stanza could be annotated as "house pain souse" followed by "1 2 1", indicating that its lines end respectively with the words 'house', 'pain' and 'souse' and that the first line rhymes with the third one.

From the corpus, we derive ground-truth pairs of rhyming and non-rhyming words. For each annotated stanza with k line-ending words, we consider all $k(k-1)/2$ pairs of words and separate them using the annotations in rhyming or non-rhyming pairs. During this process, we found a small number of annotation inconsistencies, and we checked how many words are actually present in our pronunciation dictionary. As for some poets the total number of unknown words is quite high, we exclude them from the dataset, on the grounds that their vocabulary or spelling is too different from modern use.[13]

In fact, the human assessment of rhymes may not be 100% reliable, due to the evolution of pronunciation and the imperfections of the annotation process. Additionally, some pairs annotated as non-rhyming may in fact rhyme, but have not been annotated as such since they do not fit the rhyming schema of the poem. The creation of a validation corpus can thus be improved, but the goal is to obtain the most reliable rather than the largest possible dataset, in order to validate the metric. Overall, we obtained 34,482 rhyming word pairs and 320,005 non-rhyming ones.

We assessed if our metric, given each word pair, can correctly label it as rhyming or non-rhyming. As the metric distinguishes perfect from assonant rhymes, we may or not merge these two categories. Results are shown in Table 10.3. If we merge perfect and assonant rhymes, our metric finds 80.8% of the rhymes (most of them perfect) but also labels 9.2% of non-rhyming words as rhyming (F_1 = 0.61). To maximize the F_1-score, it would seem preferable not to count assonant rhymes (then F_1 = 0.83) but in what follows we will count both types of rhymes.

[12] github.com/sravanareddy/rhymedata
[13] These are, by decreasing numbers of unknown words: Spenser, Lovelace, Drayton, Jonson, Kipling, and Byron.

Upon inspection, recall errors are often due to words that are absent from the pronunciation dictionary, and when replaced with similarly-spelled ones, their pronunciations differ. For instance, 'marinere' → 'mariner' no longer rhymes with 'hear', or 'thro" → 'throw' no longer rhymes with 'flew'. In other cases, the pronunciation in our dictionary does not match the one considered by the poet: 'stood' rhymes with 'blood' and 'thus' rhymes with 'albatross' according to the corpus, but not in our dictionary. As for precision errors, a large part of them are assonant rhymes which are not annotated in the corpus. For instance, 'there'-'around'-'howl'd'-'swound' is annotated as ABCB but we detect an assonance because the last three words have the same final vowel. Finally, annotation mistakes in the corpus can lead to both types of errors, e.g. 'close'-'beat'-'sky'-'eye'-'feet' is annotated as ABCCC in the corpus but correctly labeled by us as ABCCB.

10.4 An Auto-Regressive Language Model Fine-Tuned on Poetry

The starting point is GPT-2 (Radford et al., 2019), a general-purpose decoder LM for English. The Python implementation provided by the Huggingface library is used (Wolf et al., 2019).[14] The model is enabled to generate poetry by fine-tuning it first on a corpus of English poetry (10.4.1), and then by designing constraints so that its output has the form of a poem, with lines and stanzas (10.4.2). The frequency of rhymes is evaluated in the output of this model using our metric (10.4.3), before moving on to its specific training for rhyming in the next sections.

10.4.1 Fine-tuning GPT-2 on Poetry

Fine-tuning is on the *Gutenberg Poetry Corpus*[15] composed of approximately 3 million lines of poetry extracted from hundreds of poetry books from Project Gutenberg. Unlike the Chicago Rhyming Poetry Corpus used for validation in Section 10.3.4, no author is filtered out. The corpus is converted from the JSON format it into raw text, with poetry lines separated by newline characters ('\n') and no blank lines. Therefore, all information about stanzas, poems and books is removed, and quotation marks and dashes are also deleted. However, to emphasize the importance of lines, each line is prefixed with a '<start>' tag, which will help generation. The result is a text file with 3,085,063 lines (142 MB). On this data, the smallest GPT-2 model (124M parameters) is fine-tuned for three epochs, which takes ca. 3 hours on a single Nvidia GeForce RTX 3080 GPU.

10.4.2 Setting the Poem's Form

Generating text in a form that is typical of poetry is essential for considering rhyming patterns because without a division into lines (verses) there are no line endings that can rhyme. A

[14] huggingface.co/gpt2
[15] github.com/aparrish/gutenberg-poetry-corpus

general discussion of form constraints is out of the scope of this chapter (see Section 4.1 of Popescu-Belis et al., 2022), and we summarize the approach as follows.

We give the desired structure of the poem – number of stanzas, number of lines in each stanza, and number of syllables in each line – to the following algorithm. The first two parameters are easy to constrain, by inserting one or two newline characters. However, it is harder to constrain GPT-2 to generate a pre-specified number of syllables in a line. We generate the poem line by line, with decoding by sampling according to the word probability generated by GPT-2, modulated by a temperature factor. To generate line k, we provide GPT-2 with lines 1, 2, ..., $k-1$ as context. To obtain the expected number of syllables S_E in line k, we loop through the following steps:

1. Require GPT-2 to generate a line L with a fixed number of tokens, computed from S_E using a ratio of 1.5 syllables per token.[16]
2. Count the actual number of syllables S_L of the line L, using an algorithm for English by Emre Aydin (found at eayd.in/?p=232).
3. Exit the loop with L if $S_L = S_E$, or after 10 iterations.

10.4.3 Number of Rhymes of the Baseline

Using the GPT-2 model fine-tuned on poetry, we evaluate the number of rhyming verses as a term of comparison with further models. As we cannot make any prior assumption on the rhyming pattern, we simply group the generated verses into pairs (or couplets) by inserting a newline every other verse. When applying our metric to a set of 4,000 couplets generated in this way, we find that only 4.3% have perfect rhymes, while 6.6% have assonant rhymes, and the remaining 89.1% do not rhyme at all.

10.5 Synthetic Data with Rhymes: Rule-based Generation

We use a rule-based approach to modify the poems (see Section 9.5.3) generated by the previous model so that they follow a given rhyme scheme, which is specified in conventional form (e.g. AABB, ABAB or ABBA). This builds upon earlier work in Chapter 9, where we created an interactive system for poetry generation which combines LMs with rules governing form, rhymes, topics and emotions.

The rule-based rhyming algorithm parses the scheme, and for every second line of a rhyme (e.g., given AABB, for the second and fourth lines), it modifies the last word so that it rhymes with the last word of the previous line. The inverted rhyming dictionaries presented in Section 10.3.2 and the fine-tuned GPT-2 model are used as follows.

[16] Technically, the decoder is given a *maximum* length, but in practice, we never observed end-of-sequence symbols, so this length is always reached.

The algorithm obtains from the first dictionary the perfect rhyme ending the word to replace, and it searches the second dictionary for all the words that share this perfect rhyme. If none is found, the words sharing the respective assonant rhyme are used instead. Each word is inserted in the entire line and the result is submitted to GPT-2, which generates a likelihood score for each of these sequences. The replacement word leading to the highest score is selected. Therefore, to generate rhyming poems, we first generate a non-rhyming one and then we re-generate the last words so that they rhyme according to the given patters.

Using this strategy, we generate large numbers of poems, first with the AABB rhyming pattern, and later with the more challenging ABAB pattern. For each pattern we generate 20,000 quatrains (four-line stanzas) resulting in about 6 MB of text. Some cleaning of the data is necessary because some lines are made mostly of punctuation or include special characters. About 0.04% of the lines are removed. To simplify training, we insert a blank line after lines AA and then BB of the quatrain, so that the training data is made of rhyming couplets only. Alternatively, to learn ABAB, we insert a blank line after each quatrain. Our metric found that the first dataset has a rhyming accuracy of 97.8%, which is expected because the rhyming algorithm and the metric make use of the same dictionary.

Moreover, as the LM must capture dependencies between words at the end of lines regardless of the punctuation, we hypothesize that if we remove punctuation at the end of the verses in the training dataset, the LM would better learn rhyming patterns. The results below confirm this hypothesis.

10.6 Learning Rhyming Patterns from Synthetic Data

10.6.1 Learning the AABB Pattern

The first experiment with fine-tuning GPT-2 on synthetic data studies the simplest rhyming pattern, where two consecutive lines rhyme. As stated above, the synthetic data is made of couplets, and this is what we expect the fine-tuned model, called GPoeT, to generate as well.

To measure the proportion of rhyming verses, only the couplets are considered and isolated lines excluded, or stanzas with an odd number of lines. This ensures that we always test the rhyming of paired lines in the sample data. During fine-tuning, we generate ca. 50 kB of text every 10 epochs and measure the proportion of rhyming lines on this sample.[17] Cleaning the isolated lines removes ca. 20% of the text, a number which stays quite constant during fine-tuning (red curve in Figures 10.3 and 10.6). In other words, the model produces couplets in 80% of the cases.

The evolution of the rhyming capabilities of GPoeT during fine-tuning is shown in Figure 10.3. The improvement with respect to the baseline (fine-tuned on the Gutenberg Poetry Corpus only) is very substantial, from a proportion of perfectly rhyming couplets of 4.3% to 56.2% (a

[17] On one GPU, 10 epochs take about 25 minutes.

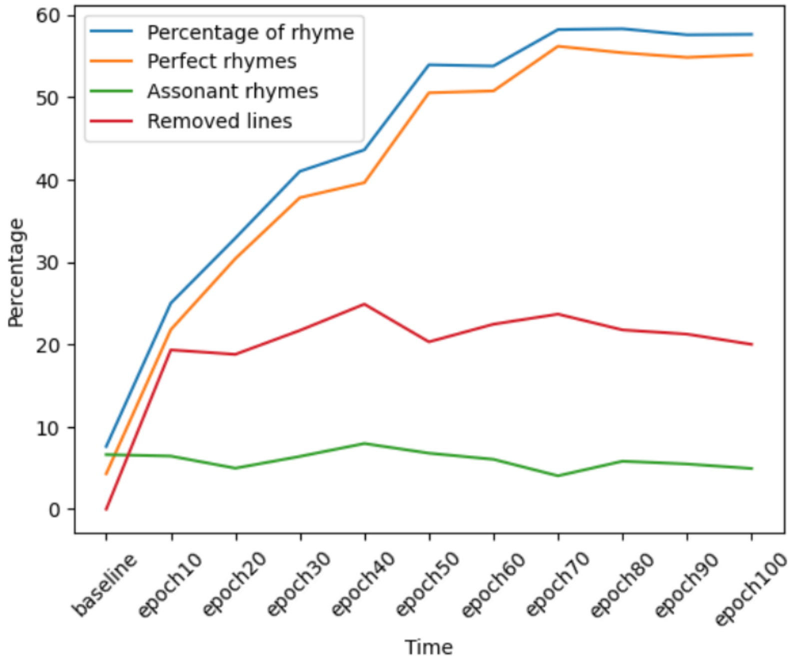

Figure 10.3: Proportion of perfect and assonant rhymes generated during the fine-tuning of GPoeT on AABB synthetic data, for 100 epochs.

factor of 13). When counting both types of rhymes, GPoeT generates 59% of rhyming couplets vs. 7.6% for the baseline (a factor of 7.7). The proportion of perfect rhymes rises quickly and then converges to around 56% after 70 epochs, while the proportion of assonant rhymes remains quite constant, likely because the data used for fine-tuning has only perfect rhymes. From the evolution of the curves, the system has likely reached its maximal performance.

The learning rate decreases linearly with the number of steps, from 5×10^{-5} to 9×10^{-7} along 10 epochs. After 10 epochs we reset the learning rate to the initial value. In this way, we force larger updates of the parameters at regular time intervals, which makes the model more robust, following insights from low-resource machine translation Atrio and Popescu-Belis (2022). This may improve training, as opposed to a learning rate that decreases too quickly. We can see in Figure 10.4 that the validation loss globally decreases over time, with small increases every 10 epochs when the learning rate is reset.

The use of quatrains stripped of the final punctuation for training is validated, hypothesizing that such tokens may hinder the learning of rhymes. We compare the proportion of rhymes generated by GPoeT after fine-tuning for 10 epochs on the synthetic quatrains when the final punctuation is kept *versus* deleted. The results shown in Table 10.4 confirm that deleting the punctuation from the training data is beneficial, and GPoeT was trained beyond 10 epochs on this data only.

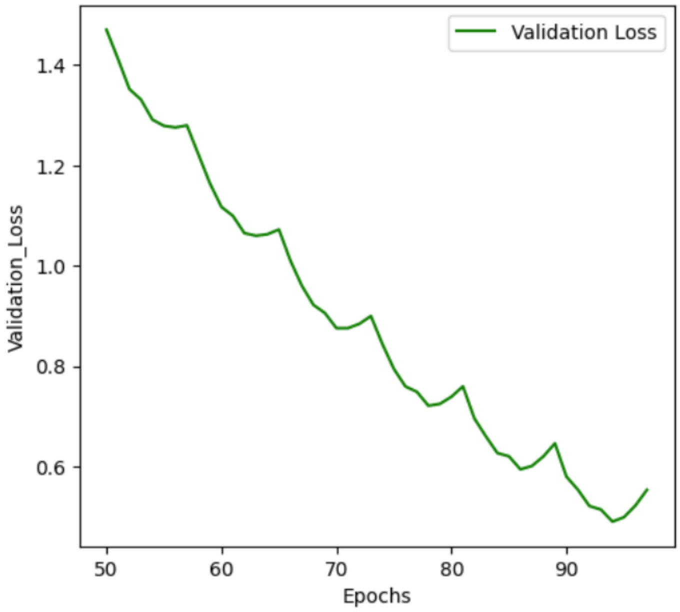

Figure 10.4: Evolution of the validation loss while learning the AABB pattern.

	Final punctuation	
Metric	**kept**	**deleted**
Perfect rhymes	13.8%	18.4%
Assonant rhymes	8.1%	7.2%
No rhyme	78.1%	74.4%

Table 10.4: Scores after 10 epochs on fine-tuning on data with or without punctuation at the end of the lines.

10.6.2 Sample Outputs of GPoeT

We provide below two unedited excerpts selected from the sample generated by the last GPoeT checkpoint.

The prince of men in arms he heard
So bold, so bold the warrior plundered

That she herself in sorrow cried
My God! who made the earth so bide

She sees no other sun above
Nor in that cloudless sky doth dove

My God! who made the earth so fair
And on this cloudless night hath mair

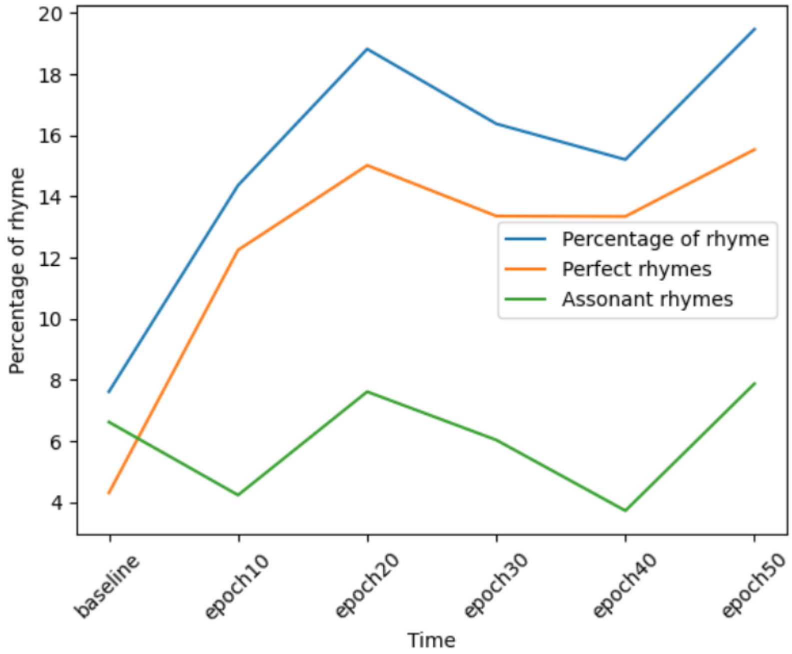

Figure 10.5: Proportion of perfect and assonant rhymes when training on natural AABB data for 50 epochs.

To the sound of your sweet voice
As of a little bird at choice

As in a trance the dreamer hears
At length a voice, so deep, so here's

That in itself it seems a sound
It is as if a great brown ground

10.6.3 Learning from Natural Data

In this experiment, it is attempted to teach GPoeT the AABB rhyming pattern using natural rather than synthetic data. It is extracted from the above-mentioned Chicago Rhyming Poetry Corpus all couplets with consecutive rhyming lines, resulting in a dataset of 2.25 MB of text, mainly with perfect rhymes (75% according to our metric). All other parameters are identical to those of the previous section.

The evolution of the proportions of perfect rhymes and assonant rhymes generated every 10 epochs during training is shown in Figure 10.5. The proportions are significantly smaller than in the previous experiment, and as the total proportion of rhymes never surpassed 20%, we only represent 50 epochs in the figure. While the model still outperforms the baseline (which has only 7.6% of rhyming verses), it is noticeably less successful than the previous one. It is likely that the smaller amount of data (by a factor of 3) and the larger variety of the vocabulary

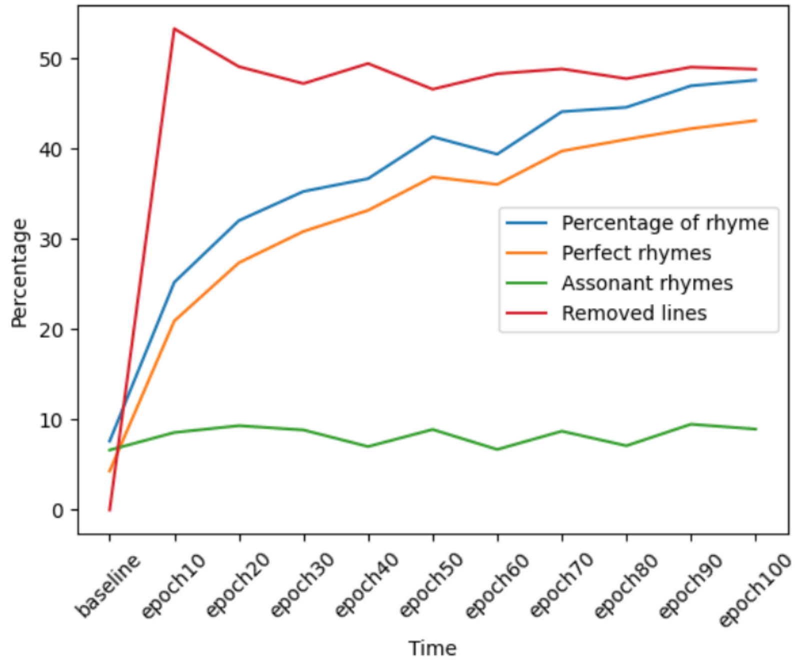

Figure 10.6: Proportion of perfect and assonant rhymes generated every 10 epochs when training on ABAB synthetic data.

used by human poets vs. GPT-2 are the main causes of the lower performance.

10.6.4 Learning the ABAB Pattern

The ABAB rhyming pattern seems more challenging to learn, as line-endings which should rhyme are further apart, separated by one verse. In this experiment, we use our second synthetic dataset, with ABAB quatrains, without separating them into couplets. Quatrains are separated by a blank line. All other parameters are identical to those of the first experiment.

We train the model until the scores stabilize, which is around 80 epochs, as shown in Figure 10.6. The proportion of perfect rhymes rises quickly and converges at around 40%, with a total number of rhyming verses (perfect and assonant) around 45%. Among these, 82.6% are perfect rhymes. As before, to evaluate rhyming, we delete solitary lines, i.e. lines that are not in a quatrain. The proportion of lines retained is 51%, which is much less than above (80%), likely because it is harder to learn to generate a quatrain than a couplet. However, when it generates a full quatrain, the model has clearly learned the ABAB rhyming scheme, although to a lesser extent than the AABB scheme (45% compared to 59%).

10.7 Discussion and Conclusion

The rhyme-generating LM presented here, GPoeT, is intended for integration into an interactive poem generation system, presented in Chapter 9. While the experiments above show that rhyming patterns can be learned from synthetic data, several issues remain to be solved in future studies.

Our rule-based rhyming algorithm operates on a poem already generated by a LM with several other parameters as input, e.g. a title or first verse, a desired theme or emotion, and a poetical form (such as a sonnet). We must now integrate GPoeT in this pipeline, and ensure that the generated rhymes are not altered by the other constrains of the system. Moreover, we must ensure that the lexical diversity of GPoeT is not reduced by its training on synthetic data.

We intend to address the problem of generating a desired form using the rule-based algorithm presented in Section 10.4.2, which takes advantage of a maximum length for the LM decoder. It may seem straightforward to replace GPT-2 with GPoeT in this algorithm, in order to obtain rhyming lines of a desired length, but our initial experiments have shown that rhymes are less satisfactory when the desired length is very different from the synthetic data GPoeT was trained on.

Moreover, while our rule-based rhyme generator can be easily adapted to any rhyming pattern, this is not yet the case for GPoeT, which is trained on one pattern at a time in our proof-of-concept. The solution lies probably in using a labeling system to indicate which lines must rhyme, and then training a GPoeT model to learn the effects of labels rather than a single rhyming pattern, in the style of the CTRL model (Keskar et al., 2019).

In this chapter, we demonstrated that rhyming is learnable with LMs that can be efficiently fine-tuned and queried with very moderate computing requirements. The key to effective fine-tuning is the use of synthetic data, which we showed how to generate in much larger amounts than what human poets have ever written. However, not all rhyming patterns are learned equally well: a pattern that exhibits longer-term dependencies such as ABAB is harder to learn than a more local one such as AABB. Overall, we have demonstrated that the rhyming and form of text generated by LMs that are fine-tuned on small amounts of data can be improved through regularization in the form of synthetic data.

10.8 Perspectives

In future work, our synthetic data could be extended to a mixture of synthetic data and authentic, silver-labeled data (following common practice in low-resource machine translation regarding back-translated data and authentic parallel data). Poetic and non-poetic human-generated data, as well as synthetic data, could be labeled so that our model is able to differentiate them during training. Additionally, the evaluation of generated poems can be improved as follows. First, our evaluation on fluency and length can be extended to the data generated by the final system, as well as dedicated rhyming evaluation. Second, in order to evaluate the poetic quality of a text, we may train a classifier to discriminate between human-generated poetic text and human-generated prosaic text, and observed its response on machine-generated poems.

11 Conclusion and Perspectives

11.1 Conclusion

This thesis deals with some problems that arise when training neural machine translation and text generation systems in low-resource settings. Due to the scarcity of data, these systems have limited generalization abilities, since it is harder to differentiate between patterns in the data that the model should learn because they are generalizable, and local patterns that should not be learned because they are artifacts due to the small training set.

We have presented solutions to this problem that rely on the use of regularization techniques. In Part I, we explored the benefits of regularization through hyper-parameters for low-resource NMT and the relationship between regularization and the shape of the parameter space, as well as an estimation of a models' neighborhood of the loss landscape. In Part II, we studied the methods to improve the use of auxiliary data, in the form of multitasking and multilingual systems, for low-resource NMT. Finally, in Part III, we examined how to improve text generation with LMs that are fine-tuned on small amounts of data, by making use of synthetic data and rule-based constraints.

More specifically, we contributed in the following ways to the improvement of translation and text generation training in low-resource settings.

- We showed that stronger regularization hyper-parameters improve scores for a variety of low-resource NMT datasets (Chapter 3). We showed that standard hyper-parameters are not suited to train Transformer networks in low-resource settings. Considering batch size, learning rate, dropout rate, and gradient clipping, we showed that a less accurate gradient – as the result of increasing the values of these hyper-parameters – produces better results, in particular if several of them are applied at the same time. We explained the improvements brought by regularization in relation to the generalization abilities of flat regions in the parameter space.

- We showed that linear interpolation with random perturbations is an efficient method to estimate the neighborhood in the loss landscape around a model (Chapter 4). We proposed various measures of flatness for an estimated neighborhood, and showed a significant correlation with BLEU score on the development set during training, which indicates that flatter regions generalize better. We also propose a solution to integrate landscape flatness information on standard SDG, in order to train the model directly on flatter areas.

- We showed that when some of the most commonly used techniques for low-resource and unsupervised machine translation are assembled into complex multi-stage pipelines, their usage brings minimal or no benefits to the final scores (Chapter 5). We proposed a simplified pipeline consisting of parent-child transfer learning, one round of back-translation for low-resource training, and two rounds for unsupervised training, with no use of common techniques such as multiple initializations or multitask and multilingual training. We compared our simplified pipeline with the best-scoring systems in the WMT 2021 Shared Task on Unsupervised MT and Very Low Resource Supervised MT. Of the four compared directions, our simpler pipeline obtains comparable scores to the highest-scoring team in one unsupervised and the two supervised ones, and improves over the other unsupervised direction.

- We studied the improvements provided by back-translation and transfer learning from various parent models, and presented systems for translating from Upper Sorbian into German and back, which rely on fixed-scheduled multitask training, for the WMT 2021 Unsupervised MT and Very Low Resource Supervised MT Task (Chapter 6). We introduced two new document-level auxiliary monolingual tasks which encourage the model to pay more attention to inter-sentential context. We observe that with a schedule of various tasks, our final systems improve over our baseline, particularly when the low-resource language is on the target side.

- We designed a self-paced many-to-one multilingual NMT system for low-resource language pairs (Chapter 7). In particular, we studied whether the variation of weights during training can be used as a measure of competence, and whether this can be used to dynamically select on which task to train, in order to better allocate training resources for tasks depending on model competence. Although we obtain better results with multilingual shuffled batches after uniform upsampling, our system sheds light on the role of many-to-one training for low-resource translation.

- We showed that the addition of multiple target languages during training of low-resource NMT systems improves their scores, but only if paired with increased regularization (Chapter 8). Additionally, less-resourced models benefit from multiple (more than two) target languages, but better-resourced models do not benefit as much from learning to translate into multiple languages.

- We contributed to a system for targeted generation of French and English poetry (Chapter 9). We designed a hybrid system combining various language models, a phonetic dictionary, and algorithms for generating rhyming poetic data that can be adapted by users to various topics and emotions, across several stages of generation. Since the very small amount of training data makes it impossible for a neural system to learn these objectives via training, we devised several strategies to alleviate this problem. We constrained the generation of lengths of verses and stanzas with rules, trained domain- and emotion-specific language models for textual adaptation, created a rhyming dictionary from a phonetic dictionary, and combined a rule-based algorithm with language model scoring for rhyming adaptation.

- We contributed to a solution for fine-tuning a LM in a low-resource setting that uses synthetic data (Chapter 10). The LM is fine-tuned with data generated by our system, which is enabled to rhyme thanks to the above-mentioned rule-based algorithm. We evaluated the rhyming quality of the resulting system with our own metric, and found that the LM trained on synthetic data was able to correctly rhyme in both couplets and quatrains. Our approach improves over the scores of the same LM when fine-tuned with a larger set of naturally occurring data featuring general poetry.

To sum up, in this thesis we have proposed improvements for low-resource NMT and text generation that rely on various regularization techniques, including obtaining noisier gradients through the hyper-parameters of the model, the scheduled use of auxiliary data in the form of denoising tasks and multilingual tasks, the combination of rules with neural systems, and the use of synthetic data for training. We have thus shown that opportunities for knowledge transfer across tasks are numerous, and that several hyper-parameters can be optimized by studying how the training evolves.

11.2 Perspectives

In this thesis, we considered that the paradigm of training a model to perform a single task with mostly fixed hyper-parameters (except dynamic ones such as the learning rate) may not be the best paradigm that we can reach. We propose here new avenues of research to continue improving over the paradigm of "fixed training for one task". We hope that these will contribute to the already existing shift towards new training paradigms that in some domains of NLP are becoming popular, like transfer learning from pre-trained models or massively multilingual systems.

When a neural network is trained on multiple tasks simultaneously, it can either leverage its learning from each task to improve overall performance, but sometimes experiences performance degradation. This difference may come from the choices of tasks, from the architecture of the network, or from various hyper-parameters related to multitask training, like the sampling between the various tasks. It is thus necessary to understand the difference in the system's behavior between the cases where multitask training is beneficial and the cases where it is not. As an example, current research on distillation and pruning methods also helps to highlight inefficient and poorly understood network architectural choices. As an additional example, as we have shown, low-resource multilingual NMT systems benefit more from decoding into many languages than encoding many languages. By studying how network weights are updated when successfully learning and leveraging multiple tasks, we should aim to select better-suited architectures for a given setting and optimize the limited representational capacity of models.

A narrow expert model excels in a specific task, whereas a competent generalist model can handle numerous tasks, albeit often lacking the quality achieved by narrow experts. (An example would be a well-trained unidirectional translation system compared to a foundational model like GPT-3.) However, training competent generalists necessitates vast amounts of unlabeled data, which is difficult for most researchers and practitioners due to limited resources. We propose as future work an approach that bridges the gap between narrow experts and competent generalists by employing meta-learning systems. This could involve the design of a monitoring system, which would be trained on different features from many datasets, similarities between various training objectives or tasks, the impact of regularization factors on gradient noise, and different network architectures. At inference time, the monitoring system's role is to continuously assess a narrow expert model during training by analyzing these static and dynamic properties of the narrow expert and its data. Subsequently, it makes regular adjustments to enhance the training process. These adjustments may include hyper-parameter tuning, data augmentation, or multi-task and multilingual training, all aimed at improving the narrow expert model's primary objective.

In the process of conducting our research for this thesis, we observed multiple rediscoveries of the same findings by different researchers. In particular, many researchers adopt their own settings of hyper-parameters, and therefore it is uncertain how our proposals will impact the research community, as many practitioners may independently rediscover them. To address the high volume of applied research, we suggest employing meta-learning on scientific articles. By extracting article structure and metadata (e.g., datasets, frameworks, architecture), systems could be trained to provide optimal recommendations for new model training (task, data, resources). These recommendations may include optimal architecture, hyper-parameters, or regularization techniques like data augmentation, and should be related to dataset features such as linguistic variation, topic, genre, or similarity to other datasets in the literature.

Bibliography

Agarwal, R. and Kann, K. (2020). Acrostic poem generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1230–1240, Online. Association for Computational Linguistics.

Agić, Ž. and Vulić, I. (2019). JW300: A wide-coverage parallel corpus for low-resource languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3204–3210, Florence, Italy. Association for Computational Linguistics.

Aharoni, R., Johnson, M., and Firat, O. (2019). Massively multilingual neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3874–3884, Minneapolis, Minnesota. Association for Computational Linguistics.

Aji, A. F., Bogoychev, N., Heafield, K., and Sennrich, R. (2020). In neural machine translation, what does transfer learning transfer? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7701–7710, Online. Association for Computational Linguistics.

Araabi, A. and Monz, C. (2020). Optimizing transformer for low-resource neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3429–3435, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Arivazhagan, N., Bapna, A., Firat, O., Lepikhin, D., Johnson, M., Krikun, M., Chen, M. X., Cao, Y., Foster, G., Cherry, C., et al. (2019). Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint arXiv:1907.05019*.

Artetxe, M., Labaka, G., and Agirre, E. (2018). A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 789–798, Melbourne, Australia. Association for Computational Linguistics.

Artetxe, M., Labaka, G., Casas, N., and Agirre, E. (2020a). Do all roads lead to Rome? understanding the role of initialization in iterative back-translation. *Knowledge-Based Systems*, 206:106401.

Artetxe, M., Ruder, S., and Yogatama, D. (2020b). On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4623–4637, Online. Association for Computational Linguistics.

Atrio, À. R., Allemann, A., Dolamic, L., and Popescu-Belis, A. (2023). A simplified training pipeline for low-resource and unsupervised machine translation. In *Proceedings of the Sixth Workshop on Technologies for Machine Translation of Low-Resource Languages (LoResMT 2023)*.

Atrio, À. R., Luthier, G., Fahy, A., Vernikos, G., Popescu-Belis, A., and Dolamic, L. (2021). The IICT-yverdon system for the WMT 2021 unsupervised MT and very low resource supervised MT task. In *Proceedings of the Sixth Conference on Machine Translation*, pages 973–981, Online. Association for Computational Linguistics.

Atrio, À. R. and Popescu-Belis, A. (2021). Small batch sizes improve training of low-resource neural MT. In *Proceedings of the 18th International Conference on Natural Language Processing (ICON)*, Patna, India.

Atrio, À. R. and Popescu-Belis, A. (2022). On the interaction of regularization factors in low-resource neural machine translation. In *Proceedings of the 23rd Annual Conference of the European Association for Machine Translation (EAMT)*, pages 111–120.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). NMT by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*, San Diego, CA, USA.

Bahri, D., Mobahi, H., and Tay, Y. (2022). Sharpness-aware minimization improves language model generalization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7360–7371, Dublin, Ireland. Association for Computational Linguistics.

Barrault, L., Biesialska, M., Bojar, O., Costa-jussà, M. R., Federmann, C., Graham, Y., Grundkiewicz, R., Haddow, B., Huck, M., Joanis, E., Kocmi, T., Koehn, P., Lo, C.-k., Ljubešić, N., Monz, C., Morishita, M., Nagata, M., Nakazawa, T., Pal, S., Post, M., and Zampieri, M. (2020). Findings of the 2020 conference on machine translation (WMT20). In *Proceedings of the Fifth Conference on Machine Translation*, pages 1–55, Online. Association for Computational Linguistics.

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., and Wood, F. (2018). Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*.

Baziotis, C., Titov, I., Birch, A., and Haddow, B. (2021). Exploring unsupervised pretraining objectives for machine translation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2956–2971, Online. Association for Computational Linguistics.

Beaudouin, V. and Yvon, F. (1996). The metrometer: a tool for analysing french verse. *Literary and Linguistic Computing*, 11(1):23–31.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

Belouadi, J. and Eger, S. (2022). ByGPT5: End-to-end style-conditioned poetry generation with token-free language models. *arXiv preprint arXiv:2212.10474*.

Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Ben-Tal, A., Den Hertog, D., De Waegenaere, A., Melenberg, B., and Rennen, G. (2013). Robust solutions of optimization problems affected by uncertain probabilities. *Management Science*, 59(2):341–357.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Bojar, O., Federmann, C., Fishel, M., Graham, Y., Haddow, B., Huck, M., Koehn, P., and Monz, C. (2018). Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 272–303, Belgium, Brussels. Association for Computational Linguistics.

Bottou, L. and Bousquet, O. (2011). The tradeoffs of large scale learning. In Sra, S., Nowozin, S., and Wright, S. J., editors, *Optimization for Machine Learning*, pages 351–368. MIT Press.

Branwen, G. and Presser, S. (2019). GPT-2 neural network poetry. *Demo Tutorial*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Burlot, F. and Yvon, F. (2018). Using monolingual data in neural machine translation: a systematic study. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 144–155, Brussels, Belgium. Association for Computational Linguistics.

Caruana, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In *ICML*.

Caswell, I., Chelba, C., and Grangier, D. (2019). Tagged back-translation. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 53–63, Florence, Italy. Association for Computational Linguistics.

Chatterjee, S. and Zielinski, P. (2022). On the generalization mystery in deep learning. *arXiv preprint arXiv:2203.10036*.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2017). Entropy-SGD: Biasing gradient descent into wide valleys. In *International Conference on Learning Representations*.

Chen, L., Ma, S., Zhang, D., Wei, F., and Chang, B. (2023). On the Pareto Front of multilingual neural machine translation. *arXiv preprint arXiv:2304.03216*.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. (2018). Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2020). Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Conneau, A. and Lample, G. (2019). Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Costa-jussà, M. R., Escolano, C., and Fonollosa, J. A. R. (2017). Byte-based neural machine translation. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 154–158, Copenhagen, Denmark. Association for Computational Linguistics.

Davies, X., Langosco, L., and Krueger, D. (2023). Unifying grokking and double descent. *arXiv preprint arXiv:2303.06173*.

Devarakonda, A., Naumov, M., and Garland, M. (2017). Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dhar, P., Bisazza, A., and van Noord, G. (2022). Evaluating pre-training objectives for low-resource translation into morphologically rich languages. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 4933–4943, Marseille, France. European Language Resources Association.

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. (2017). Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028.

Du, J., Yan, H., Feng, J., Zhou, J. T., Zhen, L., Goh, R. S. M., and Tan, V. (2022). Efficient sharpness-aware minimization for improved training of neural networks. In *International Conference on Learning Representations*.

Duchi, J., Glynn, P., and Namkoong, H. (2016). Statistics of robust optimization: A generalized empirical likelihood approach. *arXiv preprint arXiv:1610.03425*.

Dziugaite, G. K. and Roy, D. (2018). Entropy-sgd optimizes the prior of a pac-bayes bound: Generalization properties of entropy-sgd and data-dependent priors. In *International Conference on Machine Learning*, pages 1377–1386. PMLR.

Edman, L., Üstün, A., Toral, A., and van Noord, G. (2021). Unsupervised translation of German–Lower Sorbian: Exploring training and novel transfer methods on a low-resource language. In *Proceedings of the Sixth Conference on Machine Translation*, pages 982–988, Online. Association for Computational Linguistics.

Edunov, S., Ott, M., Auli, M., and Grangier, D. (2018). Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium. Association for Computational Linguistics.

Egloff, M. and Bavaud, F. (2018). Taking into account semantic similarities in correspondence analysis. In *Proceedings of the Workshop on Computational Methods in the Humanities 2018 (COMHUM 2018)*, volume 2314, pages 45–51. CEUR Workshop Proceedings.

Fan, A., Bhosale, S., Schwenk, H., Ma, Z., El-Kishky, A., Goyal, S., Baines, M., Celebi, O., Wenzek, G., Chaudhary, V., et al. (2021). Beyond English-centric multilingual machine translation. *The Journal of Machine Learning Research*, 22(1):4839–4886.

Fernandes, P., Ghorbani, B., Garcia, X., Freitag, M., and Firat, O. (2023). Scaling laws for multilingual neural machine translation. *arXiv preprint arXiv:2302.09650*.

Firat, O., Cho, K., and Bengio, Y. (2016). Multi-way, multilingual neural machine translation with a shared attention mechanism. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 866–875, San Diego, California. Association for Computational Linguistics.

Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. (2020). Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*.

Fraser, A. (2020). Findings of the WMT 2020 shared tasks in unsupervised MT and very low resource supervised MT. In *Proceedings of the Fifth Conference on Machine Translation*, pages 765–771, Online. Association for Computational Linguistics.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA. PMLR.

Gan, Z., Xu, H., and Zan, H. (2021). Self-supervised curriculum learning for spelling error correction. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3487–3494, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Sebastopol, CA.

Gervás, P. (2001). An expert system for the composition of formal spanish poetry. In *Applications and Innovations in Intelligent Systems VIII*, pages 19–32, London. Springer.

Ghazvininejad, M., Shi, X., Choi, Y., and Knight, K. (2016). Generating topical poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191, Austin, Texas. Association for Computational Linguistics.

Ghazvininejad, M., Shi, X., Priyadarshi, J., and Knight, K. (2017). Hafez: an interactive poetry generation system. In *Proceedings of ACL 2017, System Demonstrations*, pages 43–48, Vancouver, Canada. Association for Computational Linguistics.

Gogoulou, E., Ekgren, A., Isbister, T., and Sahlgren, M. (2022). Cross-lingual transfer of monolingual models. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 948–955, Marseille, France. European Language Resources Association.

Golmant, N., Vemuri, N., Yao, Z., Feinberg, V., Gholami, A., Rothauge, K., Mahoney, M. W., and Gonzalez, J. (2018). On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941.*

Gonçalo Oliveira, H. (2012). PoeTryMe: a versatile platform for poetry generation. In *Proceedings of the ECAI 2012 Workshop on Computational Creativity, Concept Invention, and General Intelligence (C3GI)*, Montpellier, France.

Gonçalo Oliveira, H., Mendes, T., Boavida, A., Nakamura, A., and Ackerman, M. (2019). Co-PoeTryMe: interactive poetry generation. *Cognitive Systems Research*, 54:199–216.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning.* MIT Press.

Goodfellow, I. J., Vinyals, O., and Saxe, A. M. (2015). Qualitatively characterizing neural network optimization problems. In *Proceedings of the International Conference on Learning Representations (ICLR 2015)*. arXiv:1904.09675.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch SGD: Training Imagenet in 1 hour. *arXiv preprint arXiv:1706.02677.*

Graves, A. (2012). Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711.*

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Gu, J., Hassan, H., Devlin, J., and Li, V. O. (2018). Universal neural machine translation for extremely low resource languages. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 344–354, New Orleans, Louisiana. Association for Computational Linguistics.

Guerberof-Arenas, A. and Toral, A. (2022). Creativity in translation: Machine translation as a constraint for literary texts. *Translation Spaces*, 11(2):184–212.

Guyon, I., Vapnik, V., Boser, B., Bottou, L., and Solla, S. A. (1991). Structural risk minimization for character recognition. *Advances in neural information processing systems*, 4.

Haddow, B., Bawden, R., Miceli Barone, A. V., Helcl, J., and Birch, A. (2022). Survey of low-resource machine translation. *Computational Linguistics*, 48(3):673–732.

Haider, T., Eger, S., Kim, E., Klinger, R., and Menninghaus, W. (2020). PO-EMO: Conceptualization, annotation, and modeling of aesthetic emotions in German and English poetry. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1652–1663, Marseille, France. European Language Resources Association.

Hämäläinen, M. (2018a). Harnessing NLG to create Finnish poetry automatically. In *Proceedings of the ninth international conference on computational creativity*. Association for Computational Creativity (ACC).

Hämäläinen, M. (2018b). Poem machine - a co-creative NLG web application for poem writing. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 195–196, Tilburg University, The Netherlands. Association for Computational Linguistics.

Hangya, V., Saadi, H. S., and Fraser, A. (2022). Improving low-resource languages in pre-trained multilingual language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11993–12006, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Hassan, H., Aue, A., Chen, C., Chowdhary, V., Clark, J., Federmann, C., Huang, X., Junczys-Dowmunt, M., Lewis, W., Li, M., et al. (2018). Achieving human parity on automatic chinese to english news translation. *arXiv preprint arXiv:1803.05567*.

Hoang, V. C. D., Koehn, P., Haffari, G., and Cohn, T. (2018). Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24, Melbourne, Australia. Association for Computational Linguistics.

Hochreiter, S. and Schmidhuber, J. (1994). Simplifying neural nets by discovering flat minima. *Advances in neural information processing systems*, 7.

Hoffer, E., Hubara, I., and Soudry, D. (2017). Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1729–1739.

Honnet, P.-E., Popescu-Belis, A., Musat, C., and Baeriswyl, M. (2018). Machine translation of low-resource spoken dialects: Strategies for normalizing Swiss German. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Hopkins, J. and Kiela, D. (2017). Automatically generating rhythmic verse with neural networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 168–178, Vancouver, Canada. Association for Computational Linguistics.

Hupkes, D., Giulianelli, M., Dankers, V., Artetxe, M., Elazar, Y., Pimentel, T., Christodoulopoulos, C., Lasri, K., Saphra, N., Sinclair, A., et al. (2022). State-of-the-art generalisation research in NLP: a taxonomy and review. *arXiv preprint arXiv:2210.03050*.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D. P., and Wilson, A. G. (2018). Averaging weights leads to wider optima and better generalization. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 876–885. AUAI Press.

Jastrzębski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. (2018). Width of minima reached by stochastic gradient descent is influenced by learning rate to batch size ratio. In *Proceedings of 27th International Conference on Artificial Neural Networks*, Lecture Notes in Computer Science, pages 392–402. Springer, Cham.

Jean, S., Firat, O., and Johnson, M. (2019). Adaptive scheduling for multi-task learning. *arXiv preprint arXiv:1909.06434*.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Jurafsky, D. and Martin, J. H. (2023). *Speech and Language Processing (3rd Edition Draft)*. https://web.stanford.edu/ jurafsky/slp3/.

Kaddour, J., Liu, L., Silva, R., and Kusner, M. (2022). When do flat minima optimizers work? In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Kantosalo, A. A., Toivanen, J. M., and Toivonen, H. T. T. (2015). Interaction evaluation for human-computer co-creativity: A case study. In *Proceedings of the 6th International Conference on Computational Creativity*. Brigham Young University.

Karimi Mahabadi, R. (2023). *Improving Generalization of Pretrained Language Models.* PhD thesis, EPFL, Lausanne, Switzerland.

Karimi Mahabadi, R., Belinkov, Y., and Henderson, J. (2020). End-to-end bias mitigation by modelling biases in corpora. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8706–8716, Online. Association for Computational Linguistics.

Karimi Mahabadi, R., Belinkov, Y., and Henderson, J. (2021a). Variational information bottleneck for effective low-resource fine-tuning. *arXiv preprint arXiv:2106.05469*.

Karimi Mahabadi, R., Ruder, S., Dehghani, M., and Henderson, J. (2021b). Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 565–576, Online. Association for Computational Linguistics.

Keshari, R., Singh, R., and Vatsa, M. (2019). Guided dropout. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4065–4072.

Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., and Socher, R. (2019). Ctrl: A conditional transformer language model for controllable generation. *ArXiv*, abs/1909.05858.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

Khatri, J., Murthy, R., and Bhattacharyya, P. (2021). Language model pretraining and transfer learning for very low resource languages. In *Proceedings of the Sixth Conference on Machine Translation*, pages 995–998, Online. Association for Computational Linguistics.

Khayrallah, H. and Koehn, P. (2018). On the impact of various types of noise on neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 74–83, Melbourne, Australia. Association for Computational Linguistics.

Khayrallah, H., Thompson, B., Post, M., and Koehn, P. (2020). Simulated multiple reference training improves low-resource machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 82–89, Online. Association for Computational Linguistics.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. In *arXiv:1412.6980*.

Klein, G., Hernandez, F., Nguyen, V., and Senellart, J. (2020). The OpenNMT neural machine translation toolkit: 2020 edition. In *Proceedings of the 14th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pages 102–109, Virtual. Association for Machine Translation in the Americas.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. (2017). OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

Knowles, R. and Larkin, S. (2021). NRC-CNRC systems for Upper Sorbian-German and Lower Sorbian-German machine translation 2021. In *Proceedings of the Sixth Conference on Machine Translation*, pages 999–1008, Online. Association for Computational Linguistics.

Knowles, R., Larkin, S., Stewart, D., and Littell, P. (2020). NRC systems for low resource German-Upper Sorbian machine translation 2020: Transfer learning with lexical modifications. In *Proceedings of the Fifth Conference on Machine Translation*, pages 1112–1122, Online. Association for Computational Linguistics.

Kocmi, T. and Bojar, O. (2017). Curriculum learning and minibatch bucketing in neural machine translation. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, pages 379–386, Varna, Bulgaria. INCOMA Ltd.

Kocmi, T. and Bojar, O. (2018). Trivial transfer learning for low-resource neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 244–252, Brussels, Belgium. Association for Computational Linguistics.

Kocmi, T. and Bojar, O. (2020). Efficiently reusing old models across languages via transfer learning. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 19–28, Lisboa, Portugal. European Association for Machine Translation.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit X: Papers*, pages 79–86, Phuket, Thailand.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Koehn, P. and Knowles, R. (2017). Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver. Association for Computational Linguistics.

Korpelevich, G. M. (1976). The extragradient method for finding saddle points and other problems. *Matecon*, 12:747–756.

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 66–75.

Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Kukačka, J., Golkov, V., and Cremers, D. (2017). Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*.

Kumar, G., Koehn, P., and Khudanpur, S. (2021). Learning curricula for multilingual neural machine translation training. In *Proceedings of Machine Translation Summit XVIII: Research Track*, pages 1–9, Virtual. Association for Machine Translation in the Americas.

Kumar, M., Packer, B., and Koller, D. (2010). Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23.

Kvapilíková, I., Kocmi, T., and Bojar, O. (2020). CUNI systems for the unsupervised and very low resource translation task in WMT20. In *Proceedings of the Fifth Conference on Machine Translation*, pages 1123–1128, Online. Association for Computational Linguistics.

Lample, G., Conneau, A., Denoyer, L., and Ranzato, M. (2017). Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*.

Lample, G., Ott, M., Conneau, A., Denoyer, L., and Ranzato, M. (2018). Phrase-based & neural unsupervised machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5039–5049, Brussels, Belgium. Association for Computational Linguistics.

Lau, J. H., Cohn, T., Baldwin, T., Brooke, J., and Hammond, A. (2018). Deep-speare: A joint neural model of poetic language, meter and rhyme. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1948–1958, Melbourne, Australia. Association for Computational Linguistics.

Läubli, S., Castilho, S., Neubig, G., Sennrich, R., Shen, Q., and Toral, A. (2020). A set of recommendations for assessing human–machine parity in language translation. *Journal of Artificial Intelligence Research*, 67:653–672.

Läubli, S., Sennrich, R., and Volk, M. (2018). Has machine translation achieved human parity? a case for document-level evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4791–4796. Association for Computational Linguistics.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2017). Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*.

Li, J., Song, Y., Zhang, H., Chen, D., Shi, S., Zhao, D., and Yan, R. (2018). Generating classical Chinese poems via conditional variational autoencoder and adversarial training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3890–3900, Brussels, Belgium. Association for Computational Linguistics.

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., and Gonzalez, J. (2020). Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on machine learning*, pages 5958–5968. PMLR.

Liang, X., Wu, L., Li, J., Wang, Y., Meng, Q., Qin, T., Chen, W., Zhang, M., and Liu, T.-Y. (2021). R-drop: Regularized dropout for neural networks. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 10890–10905. Curran Associates, Inc.

Libovický, J. and Fraser, A. (2021a). Findings of the WMT 2021 shared tasks in unsupervised MT and very low resource supervised MT. In *Proceedings of the Sixth Conference on Machine Translation*, pages 726–732, Online. Association for Computational Linguistics.

Libovický, J. and Fraser, A. (2021b). The LMU Munich systems for the WMT21 unsupervised and very low-resource translation task. In *Proceedings of the Sixth Conference on Machine Translation*, pages 989–994, Online. Association for Computational Linguistics.

Libovický, J., Hangya, V., Schmid, H., and Fraser, A. (2020). The LMU Munich system for the WMT20 very low resource supervised MT task. In *Proceedings of the Fifth Conference on Machine Translation*, pages 1104–1111, Online. Association for Computational Linguistics.

Libovický, J., Schmid, H., and Fraser, A. (2022). Why don't people use character-level machine translation? In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2470–2485, Dublin, Ireland. Association for Computational Linguistics.

Libovický, J. and Fraser, A. (2021). Findings of the WMT 2021 shared tasks in unsupervised MT and very low resource supervised MT. In *Proceedings of the Sixth Conference on Machine Translation*.

Lin, T., Kong, L., Stich, S., and Jaggi, M. (2020). Extrapolation for large-batch training in deep learning. In *International Conference on Machine Learning*, pages 6094–6104. PMLR.

Lison, P. and Tiedemann, J. (2016). OpenSubtitles2016: Extracting large parallel corpora from movie and TV subtitles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 923–929, Portorož, Slovenia. European Language Resources Association (ELRA).

Liu, X., Lai, H., Wong, D. F., and Chao, L. S. (2020a). Norm-based curriculum learning for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 427–436, Online. Association for Computational Linguistics.

Liu, Y., Gu, J., Goyal, N., Li, X., Edunov, S., Ghazvininejad, M., Lewis, M., and Zettlemoyer, L. (2020b). Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

Liu, Y. and Lapata, M. (2019). Hierarchical transformers for multi-document summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5070–5081, Florence, Italy. Association for Computational Linguistics.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). RoBERTa: a robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Lu, J. and Zhang, J. (2021). Exploiting curriculum learning in unsupervised neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 924–934.

Lu, P., Kobyzev, I., Rezagholizadeh, M., Rashid, A., Ghodsi, A., and Langlais, P. (2022). Improving generalization of pre-trained language models via stochastic weight averaging. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 4948–4954, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Martin, L., Muller, B., Suárez, P. J. O., Dupont, Y., Romary, L., de La Clergerie, É. V., Seddah, D., and Sagot, B. (2019). CamemBERT: a tasty French language model. *arXiv preprint arXiv:1911.03894*.

McCandlish, S., Kaplan, J., Amodei, D., and Team, O. D. (2018). An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*.

McGregor, S., Purver, M., and Wiggins, G. (2016). Process based evaluation of computer generated poetry. In *Proceedings of the INLG 2016 Workshop on Computational Creativity in Natural Language Generation*, pages 51–60, Edinburgh, UK. Association for Computational Linguistics.

Mikolov, T. (2012). *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, Faculty of Information Technology, Brno, Czechia.

Moore, R. C. and Lewis, W. (2010). Intelligent selection of language model training data. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 220–224, Uppsala, Sweden. Association for Computational Linguistics.

Moradi, R., Berangi, R., and Minaei, B. (2020). A survey of regularization strategies for deep models. *Artificial Intelligence Review*, 53(6):3947–3986.

Neubig, G. and Hu, J. (2018). Rapid adaptation of neural machine translation to new languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 875–880, Brussels, Belgium. Association for Computational Linguistics.

New, B., Pallier, C., Brysbaert, M., and Ferrand, L. (2004). Lexique 2 : A new French lexical database. *Behavior Research Methods, Instruments, & Computers*, 36:516–524.

Ormazabal, A., Artetxe, M., Agirrezabal, M., Soroa, A., and Agirre, E. (2022). PoeLM: A meter- and rhyme-controllable language model for unsupervised poetry generation. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3655–3670, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Peng, H., Mou, L., Li, G., Chen, Y., Lu, Y., and Jin, Z. (2015). A comparative study on regularization strategies for embedding-based neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2106–2111, Lisbon, Portugal. Association for Computational Linguistics.

Pham, H. and Le, Q. (2021). Autodropout: Learning dropout patterns to regularize deep networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9351–9359.

Pham, M.-Q., Crego, J., and Yvon, F. (2022a). Multi-domain adaptation in neural machine translation with dynamic sampling strategies. In *Proceedings of the 23rd Annual Conference of the European Association for Machine Translation*, pages 13–22, Ghent, Belgium. European Association for Machine Translation.

Pham, M.-Q., Yvon, F., and Crego, J. (2022b). Latent group dropout for multilingual and multidomain machine translation. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2469–2481, Seattle, United States. Association for Computational Linguistics.

Platanios, E. A., Stretcu, O., Neubig, G., Póczos, B., and Mitchell, T. (2019). Competence-based curriculum learning for neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1162–1172.

Popel, M. and Bojar, O. (2018). Training tips for the Transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70.

Popescu-Belis, A., Atrio, À., Minder, V., Xanthos, A., Luthier, G., Mattei, S., and Rodriguez, A. (2022). Constrained language models for interactive poem generation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 3519–3529, Marseille, France. European Language Resources Association.

Popescu-Belis, A., Atrio, À. R., Bernath, B., Boisson, E., Ferrari, T., Theimer-Liemard, X., and Vernikos, G. (2023). Gpoet: a language model trained for rhyme generation on synthetic data. In *Proceedings of the 7th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 10–20. Association for Computational Linguistics.

Popović, M. (2015). chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Post, M. (2018). A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Power, A., Burda, Y., Edwards, H., Babuschkin, I., and Misra, V. (2022). Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*.

Provilkov, I., Emelianenko, D., and Voita, E. (2020). BPE-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online. Association for Computational Linguistics.

Qi, Y., Sachan, D., Felix, M., Padmanabhan, S., and Neubig, G. (2018). When and why are pre-trained word embeddings useful for neural machine translation? In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 529–535, New Orleans, Louisiana. Association for Computational Linguistics.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding with unsupervised learning. *Technical report, OpenAI*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P. J., et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(140):1–67.

Rei, R., Stewart, C., Farinha, A. C., and Lavie, A. (2020). COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

Sankar, A. R., Khasbage, Y., Vigneswaran, R., and Balasubramanian, V. N. (2021). A deeper look at the hessian eigenspectrum of deep neural networks and its applications to regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9481–9488.

Scherrer, Y., Grönroos, S.-A., and Virpioja, S. (2020). The University of Helsinki and aalto university submissions to the WMT 2020 news and low-resource translation tasks. In *Proceedings* *of the Fifth Conference on Machine Translation*, pages 1129–1138, Online. Association for Computational Linguistics.

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing*, Manchester, UK.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

Sennrich, R., Haddow, B., and Birch, A. (2016a). Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Sennrich, R., Haddow, B., and Birch, A. (2016b). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Sennrich, R. and Zhang, B. (2019). Revisiting low-resource neural machine translation: A case study. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 211–221, Florence, Italy. Association for Computational Linguistics.

Shaham, U., Elbayad, M., Goswami, V., Levy, O., and Bhosale, S. (2023). Causes and cures for interference in multilingual translation. *arXiv preprint arXiv:2212.07530*.

Shapiro, A., Salama, M., Abdelhakim, O., Fayed, M., Khalafallah, A., and Adly, N. (2022). The AIC system for the WMT 2022 unsupervised MT and very low resource supervised MT task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 1117–1121, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. (2017). Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*.

Smith, S. L. and Le, Q. V. (2017). A bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451*.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2019). Mass: Masked sequence to sequence pre-training for language generation. In *International Conference on Machine Learning*, pages 5926–5936. PMLR.

Soviany, P., Ionescu, R. T., Rota, P., and Sebe, N. (2022). Curriculum learning: A survey. *International Journal of Computer Vision*, 130(6):1526–1565.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Steinberger, R., Eisele, A., Klocek, S., Pilos, S., and Schlüter, P. (2012). DGT-TM: A freely available translation memory in 22 languages. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 454–459, Istanbul, Turkey. European Language Resources Association (ELRA).

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, Bellevue, WA, USA.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Tang, Y., Tran, C., Li, X., Chen, P.-J., Goyal, N., Chaudhary, V., Gu, J., and Fan, A. (2021). Multilingual translation from denoising pre-training. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3450–3466, Online. Association for Computational Linguistics.

Tay, Y., Tran, V. Q., Ruder, S., Gupta, J., Chung, H. W., Bahri, D., Qin, Z., Baumgartner, S., Yu, C., and Metzler, D. (2021). Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672*.

Templin, M. C. (1957). *Certain language skills in children; their development and interrelationships.* University of Minnesota Press.

Tiedemann, J. (2012). Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey. European Language Resources Association (ELRA).

Tiedemann, J. and Thottingal, S. (2020). OPUS-MT – building open translation services for the world. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 479–480, Lisboa, Portugal. European Association for Machine Translation.

Toral, A. (2020). Reassessing claims of human parity and super-human performance in machine translation at WMT 2019. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 185–194, Lisboa, Portugal. European Association for Machine Translation.

Van de Cruys, T. (2019). La génération automatique de poésie en français (automatic poetry generation in French). In *Actes de la Conférence sur le Traitement Automatique des Langues Naturelles (TALN) PFIA 2019. Volume I : Articles longs*, pages 113–126, Toulouse, France. ATALA.

Van de Cruys, T. (2020). Automatic poetry generation from prosaic text. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2471–2480, Online. Association for Computational Linguistics.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008.

Wan, Y., Yang, B., Wong, D. F., Zhou, Y., Chao, L. S., Zhang, H., and Chen, B. (2020). Self-paced learning for neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1074–1080, Online. Association for Computational Linguistics.

Wang, C., Venkatesh, S., and Judd, J. (1993). Optimal stopping and effective machine complexity in learning. *Advances in neural information processing systems*, 6.

Wang, K. and Wan, X. (2018). Sentigan: Generating sentimental texts via mixture adversarial networks. In *International Joint Conferences on Artificial Intelligence*, pages 4446–4452.

Wang, W., Tian, Y., Ngiam, J., Yang, Y., Caswell, I., and Parekh, Z. (2020a). Learning a multi-domain curriculum for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7711–7723, Online. Association for Computational Linguistics.

Wang, X., Chen, Y., and Zhu, W. (2021). A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):4555–4576.

Wang, X., Pham, H., Arthur, P., and Neubig, G. (2019). Multilingual neural machine translation with soft decoupled encoding. *arXiv preprint arXiv:1902.03499*.

Wang, X., Pham, H., Michel, P., Anastasopoulos, A., Carbonell, J., and Neubig, G. (2020b). Optimizing data usage via differentiable rewards. In *International Conference on Machine Learning*, pages 9983–9995. PMLR.

Wang, X., Tsvetkov, Y., and Neubig, G. (2020c). Balancing training for multilingual neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8526–8537, Online. Association for Computational Linguistics.

Wang, Y., Zhai, C., and Hassan, H. (2020d). Multi-task learning for multilingual neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1022–1034, Online. Association for Computational Linguistics.

Wei, D., Li, Z., Wu, Z., Yu, Z., Chen, X., Shang, H., Guo, J., Wang, M., Lei, L., Zhang, M., Yang, H., and Qin, Y. (2021). HW-TSC's participation in the WMT 2021 news translation shared task. In *Proceedings of the Sixth Conference on Machine Translation*, pages 225–231, Online. Association for Computational Linguistics.

Weller-Di Marco, M. and Fraser, A. (2022). Findings of the WMT 2022 shared tasks in unsupervised MT and very low resource supervised MT. In *Proceedings of the Seventh Conference on Machine Translation*, pages 801–805, Abu Dhabi. Association for Computational Linguistics.

Wöckener, J., Haider, T., Miller, T., Nguyen, T.-K., Nguyen, T. T. L., Pham, M. V., Belouadi, J., and Eger, S. (2021). End-to-end style-conditioned poetry generation: What does it take to learn from examples alone? In *Proceedings of the 5th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 57–66, Punta Cana, Dominican Republic (online). Association for Computational Linguistics.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2019). HuggingFace's Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Wolleb, B., Silvestri, R., Vernikos, G., Dolamic, L., and Popescu-Belis, A. (2023). Assessing the importance of frequency versus compositionality for subword-based tokenization in nmt. In *Proceedings of the 24th Annual Conference of the European Association for Machine Translation (EAMT)*, pages 137–146.

Wortsman, M., Horton, M. C., Guestrin, C., Farhadi, A., and Rastegari, M. (2021). Learning neural network subspaces. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11217–11227. PMLR.

Wu, M., Li, Y., Zhang, M., Li, L., Haffari, G., and Liu, Q. (2021). Uncertainty-aware balancing for multilingual and multi-domain neural machine translation training. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7291–7305, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xu, C., Hu, B., Jiang, Y., Feng, K., Wang, Z., Huang, S., Ju, Q., Xiao, T., and Zhu, J. (2020a). Dynamic curriculum learning for low-resource neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3977–3989, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Xu, H., van Genabith, J., Xiong, D., and Liu, Q. (2020b). Dynamically adjusting transformer batch size by monitoring gradient direction change. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3519–3524, Online. Association for Computational Linguistics.

Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., and Raffel, C. (2022). ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., and Raffel, C. (2021). mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Yan, R., Jiang, H., Lapata, M., Lin, S.-D., Lv, X., and Li, X. (2013). i, Poet: Automatic Chinese poetry composition through a generative summarization framework under constrained optimization. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2197–2203. AAAI Press.

Yang, Z., Cai, P., Feng, Y., Li, F., Feng, W., Chiu, E. S.-Y., and Yu, H. (2019). Generating classical Chinese poems from vernacular Chinese. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6155–6164, Hong Kong, China. Association for Computational Linguistics.

Yule, G. U. (1944). *The Statistical Study of Literary Vocabulary*. Cambridge University Press.

Zaremoodi, P. and Haffari, G. (2019). Adaptively scheduled multitask learning: The case of low-resource neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 177–186, Hong Kong. Association for Computational Linguistics.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2021a). Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115.

Zhang, M., Meng, F., Tong, Y., and Zhou, J. (2021b). Competence-based curriculum learning for multilingual machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2481–2493, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhang, M., Wu, M., Li, P., Li, L., and Liu, Q. (2021c). NoahNMT at WMT 2021: Dual transfer for very low resource supervised machine translation. In *Proceedings of the Sixth Conference on Machine Translation*, pages 1009–1013, Online. Association for Computational Linguistics.

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2019a). BERTScore: evaluating text generation with bert. In *Proceedings of the International Conference on Learning Representations (ICLR 2020)*. arXiv:1904.09675.

Zhang, X., Kumar, G., Khayrallah, H., Murray, K., Gwinnup, J., Martindale, M. J., McNamee, P., Duh, K., and Carpuat, M. (2018). An empirical exploration of curriculum learning for neural machine translation. *arXiv preprint arXiv:1811.00739*.

Zhang, X. and Lapata, M. (2014). Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680, Doha, Qatar. Association for Computational Linguistics.

Zhang, X., Shapiro, P., Kumar, G., McNamee, P., Carpuat, M., and Duh, K. (2019b). Curriculum learning for domain adaptation in neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1903–1915, Minneapolis, Minnesota. Association for Computational Linguistics.

Zhou, C., Levy, D., Li, X., Ghazvininejad, M., and Neubig, G. (2021). Distributionally robust multilingual machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5664–5674, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zoph, B., Yuret, D., May, J., and Knight, K. (2016). Transfer learning for low-resource neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1568–1575, Austin, Texas. Association for Computational Linguistics.

HEIG-VD
Route de Cheseaux 1, Office 40
CH-1401 Yverdon-les-Bains
Vaud, Switzerland
alexratrio@gmail.com
https://alexratrio.github.io/

Àlex R. Atrio

Research Interests

Natural Language Processing, Machine Translation, Language Generation, Machine Learning, Deep Learning.

Education

2019 – **Ph.D. Electrical Engineering Doctoral Program (EDEE).** *Exams: 4.75/6*
École Polytechnique Fédérale de Lausanne (EPFL) & Haute École d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD), Switzerland.
- *Thesis*: Regularization Techniques for Low-Resource Machine Translation.
- *Supervisors*: Andrei Popescu-Belis & Jean-Marc Odobez

2017 – 2018 **Master in Cognitive Science and Language (CCiL).** *GPA: 8.2/10*
University Pompeu Fabra (UPF) & University of Barcelona (UB).
- *Thesis*: Machine Translation Inspired Word Reordering as Preprocessing for Cross-Lingual Sentiment Analysis.
- *Supervisors*: Toni Badia & Jeremy Barnes

2011 – 2016 **Bachelor in Philosophy with Specialization in Logic, Philosophy of Language and Philosophy of Science.** *GPA: 7.6/10*
University of Barcelona (UB).

Publications

Papers in Conference Proceedings

2023 **Àlex R. Atrio**, Alexis Allemann, Ljiljana Dolamic, and Andrei Popescu-Belis. A simplified training pipeline for low-resource and unsupervised machine translation. In *Proceedings of the Sixth Workshop on Technologies for Machine Translation of Low-Resource Languages (LoResMT 2023)*, 2023.

2023 Andrei Popescu-Belis, **Àlex R. Atrio**, Bastien Bernath, Etienne Boisson, Xavier Theimer-Liemard, Teo Ferrari, and Giorgos Vernikos. Gpoet: a language model trained for rhyme generation on synthetic data. In *Proceedings of the 7th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, 2023.

2022 **Àlex R. Atrio** and Andrei Popescu-Belis. On the interaction of regularization factors in low-resource neural machine translation. In *Proceedings of the 23rd Annual Conference of the European Association for Machine Translation*, pages 111–120, Ghent, Belgium, June 2022.

2022 Andrei Popescu-Belis, **Atrio, Àlex R.**, Valentin Minder, Aris Xanthos, Gabriel Luthier, Simon Mattei, and Antonio Rodriguez. Constrained language models for interactive poem generation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 3519–3529, Marseille, France, June 2022.

2021 **Atrio, Àlex R.** and Andrei Popescu-Belis. Small Batch Sizes Improve Training of Low-Resource Neural MT. In *Proceedings of the 18th International Conference on Natural Language Processing (ICON)*, pages 18–24, National Institute of Technology Silchar, Silchar, India, December 2021.

2021 **Àlex R. Atrio**, Gabriel Luthier, Axel Fahy, Giorgos Vernikos, Andrei Popescu-Belis, and Ljiljana Dolamic. The iict-yverdon system for the WMT 2021 unsupervised MT and very low resource supervised MT task. In *Proceedings of the 6th Conference on Machine Translation (WMT)*, 2021.

2019 **Àlex R. Atrio**, Toni Badia, and Jeremy Barnes. On the effect of word order on cross-lingual sentiment analysis. *Procesamiento del Lenguaje Natural (SEPLN)*, volume 63, pages 23–30, 2019.

Presentations at Reviewed Conferences without Proceedings

2020 Andrei Popescu-Belis, Aris Xanthos, Valentin Minder, **Àlex R. Atrio**, Gabriel Luthier, and Antonio Rodriguez. Interactive poem generation: when language models support human creativity. *Presentation at the 5th Swiss Text Analytics and 16th KONVENS Conference (SwissText-KONVENS 2020)*, 2020.

Presentations

2020 **Àlex R. Atrio**, Valentin Minder, Gabriel Luthier, and Andrei Popescu-Belis. Création poétique assistée par ordinateur (cpao). *Presentation at the Data Science Seminar, IICT / HEIG-VD*, 2020.

Participation in Research Projects

2019 – **DOMAT - On-demand Knowledge for Document-level Machine Translation**
DOMAT (n. 175693) is a Swiss National Science Foundation project which aims at designing a novel approach for providing on-demand linguistic knowledge to neural machine translation systems.

2022 – **UNISUB - Unsupervised NMT with Innovative Multilingual Subword Models**
UNISUB is a project funded by Armasuisse. Its main goal is to improve unsupervised neural machine translation through an adaptive scheduling of the training tasks and through improved tokenization models, either using subword aligment across the source and target languages, or more advanced subword construction methods.

2020 – 2021 **FamilyMT - Multilingual Neural MT for Families of Languages and Dialects**
FamilyMT is an Armasuisse-funded project which aims at designing systems for closely related source languages of the same family, and show that this relatedness can be leveraged to improve translation of low-resource languages in multilingual neural machine translation.

2019 – 2020 **Digital Lyric - Création Poétique Assistée par Ordinateur**
Digital Lyric (n. 184330) is a project supported by the SNSF Agora program (main applicant: University of Lausanne). Its main goals are to design a system for interactive poem generation, which combines language models with explicit constraints that can be set by users on form, topic, emotion, and rhyming scheme. The system was showcased at a public exhibition at the Château de Morges (Vaud, Switzerland), from February 14 to May 10, 2020.

Awards and Distinctions

2019 **Best Paper Award.** *"On the Effect of Word Order on Cross-lingual Sentiment Analysis"*, SEPLN (Conference of the Spanish Society for Natural Language Processing), Bilbao.

Professional Experience

2019 – ... **Research Assistant.** Haute École d'Ingénierie et de Gestion du Canton de Vaud, Switzerland.

Teaching

Teaching assistant for courses

2020 & 2022 Traitement Automatique des Langues (TAL) – BSc in computer science (3rd year)

Languages (Proficiency)

Catalan Native

Spanish Native

English C2 - *Proficiency Exam in English (University of Cambridge)*

French Professional proficiency

Galician Professional proficiency

Computer skills

Programming Python, Prolog

Visualizing Matplotlib, Pandas

OS Linux, Windows

Toolkits PyTorch, TensorFlow

Misc LaTeX, Git