LaMBERT: Light and Multigranular BERT

Ljupche Milosheski
École Polytechnique Fédérale de Lausanne
ljupche.milosheski@epfl.ch

Abstract

Pre-training complex language models is essential for the success of the recent methods such as BERT or OpenAI GPT. Their size makes not only the pre-training phase, but also consecutive applications to be computationally expensive. BERT-like models excel at token-level tasks as they provide reliable token embeddings, but they fall short when it comes to sentence or higher-level structure embeddings. The reason is that these models do not have a built-in mechanism that explicitly provides such representations. Namely, both training objectives of BERT, masked language modeling, and next sentence prediction consider at most two sentences in a single training instance, which makes it infeasible to learn higher-level structure representations. We introduce Light and Multigranural BERT that has similar complexity to BERT in the number of parameters, but is about 3 times faster by modifying the input representation, which consequently introduces changes to the attention mechanism and at the same time produces reliable segment embeddings as it is one of our training objectives. The model we publish achieves 70.7% on the MNLI task, which is promising bearing in mind there were two major issues with it.

1 Introduction

Many of the NLP tasks have limited training data, thus it is infeasible to achieve state-of-the-art results by only using the task-specific training data. This is the reason why the language models have benefited from pre-training general models on huge external datasets (Devlin et al., 2018; Radford et al., 2019; Howard and Ruder, 2018). Recent work has shown that large models with hundreds of millions or even billions of parameters are necessary for achieving state-of-the-art performance (Devlin et al., 2018; Radford et al., 2019). The operations on these models are computationally expensive due to their complexity. While these models once pre-trained they can be later easily fine-tuned for downstream tasks, the real-world application is questionable. There has been recent work on distilling large pre-trained models down to smaller ones (Sun et al., 2019; Turc et al., 2019).

One can reduce the complexity of a model by speeding it up. Having a faster model will allow us to train it on the same data as the old model faster, or train it on more data for the same amount of time as the old model. The latter approach is especially important if our main concern is the performance, as models such OpenAI GPT (Radford et al., 2019) and RoBERTa (Liu et al., 2019) have greatly benefited from more data. We introduce Light and Multigranural BERT (LaMBERT) that introduces changes that is about 3 times faster than BERT due to changes to the input representation and attention mechanism.

LaMBERT has similar complexity than BERT in the number of parameters, but is faster, thus its training time is shorter for a similar amount of data. The reasons for this are two majors modification: changes in the input representation that allows us to reduce the maximum number of tokens per sentence and a consequent change to the attention mechanism. Having a fixed length of the number tokens per sentence has its benefits and disadvantages. While it is easier to work with fixed length data structures, having a fixed length creates a waste of computation due to the padding, but nonetheless, we must consider them due to the skewed distribution of the data. It is indeed mentioned that BERT was first 90% of the time trained with at most 128 tokens, and then the last 10% for at most 512 tokens. This approach, however, only speeds up the pre-training time, but the complexity of the final model remains unchanged.

We say that our input, which we call a sentence,

is divided into at most S segments. A sentence in this context may not refer to an actual sentence from the text, but rather a contiguous piece of text. A segment is a substring of the sentence that consists of M tokens. In the model we publish, we set $S = M = 23$. This results in at most 529 tokens per training instance, which is the closest perfect square to BERT's 512.

The general idea of LaMBERT is to reduce the maximum segment length and in addition, have partially variable sentence lengths. In general, this would mean that we could not train sentences longer than the maximum length, but we will explain how to mitigate this issue.

The change in the input representation directly impacts the entities over which the attention mechanism is computed. The attention in LaMBERT is computed in two phases. The first is similar to BERT as we compute the attention of the tokens with respect to all tokens, and in addition all segment embeddings. The second is cross-attention, in which we manually compute the [CLS] embedding from the attention of the segment embeddings.

The second improvement over BERT is that LaMBERT explicitly produces segment representations. BERT does not explicitly provide ones, so using naive techniques such as averaging the token embeddings of a sentence often does not work well. Our model is pre-trained on instances of 2 sentences, each of which is split into a few segments. The segments are connected via the attention mechanism to all other segments in the instance. This should intuitively work as it creates deeper dependencies between multiple segments in long paragraphs, compared to BERT where there are only connections between the tokens.

In summary, the contributions of this work are:

1. we introduce modifications to the input representation of BERT results in about 3 times improvement in speed, and

2. we partially change BERT's attention mechanism so that LaMBERT has a built-in mechanism for segment embeddings.

The report is organized as follows. In section 2 we compare our approach to recent work about complexity reduction of language models and learning and improving segment representations. Section 3 discusses the changes we introduced over BERT in detail. In section 4 we explain the fine-tuning procedure of LaMBERT on the downstream tasks and discuss the results when compared to BERT. The report is concluded with section 5.

2 Related work

2.1 Complexity reduction techniques

The technique of reducing the complexity by lowering the number of parameters, and at the same time speed up the training has already been covered by Lan et al. (2019). Their approach, however, is different than ours. They reduce the number of parameters by factorizing the embedding matrix, whereas our reduction is focused on changing the input representation and the attention mechanism. We believe that their reduction of the embedding matrix can be included to further reduce the number of parameters of LaMBERT, but we will not use it in this work. They use cross-layer parameter sharing, whereas we use weight sharing between different segments.

Clark et al. (2020) propose ELECTRA, which speeds up the pre-training time of BERT-like models by changing the MLM task to a task called replaced token detection. They corrupt some of the input tokens with plausible alternatives sampled from a small generator network. Instead of training a model that predicts the corrupted entries, they train a discriminative model that predicts whether each token in the corrupted input was replaced by a generator sample or not. This makes the training faster because at each step the model learns information about every token in the training instance instead of only the masked tokens in BERT. As the new task allows the model to learn information quicker than BERT, the training of ELECTRA is faster than BERT. At the same time, it removes the problem that during the pre-training BERT has [MASK] tokens, but not during the fine-tuning on downstream tasks.

2.2 Learning segment representations

Prior work used different training objectives to learn sentence representations. Kiros et al. (2015) introduce skip-thought vectors in which they build an encoder-decoder model that tries to reconstruct the surrounding sentences of an encoded passage. Hill et al. (2016) compare various methods used to obtain sentence embeddings, argue that the performance of each approach is dependent on the task, and denoise autoencoder derived objectives to learn sentence representations. Jernite et al. (2017) use purely discriminative objective to train the sentence

embeddings, and they obtain them via exploiting the signals from paragraph-level discourse coherence to train these models to understand text. Their task is, for given the first three sentences of a paragraph, choose the next sentence from five sentences later in the paragraph. Logeswaran and Lee (2018) propose a framework in which they reformulate the problem of predicting the context in which a sentence appears as a classification problem. This allows their model to learn different types of encoding functions, and they rank the candidate next sentences.

Our training objective to learn the segment representations is to predict whether the pair of sentences was swapped. During the training, we did not include sentences from random paragraphs, but rather from the same paragraph or similar context. This is inspired by ALBERT's (Lan et al., 2019) argument that replacing sentences randomly chosen from different documents makes the next sentence prediction (NSP) task trivial. They go even further and use a more difficult task which is called sentence ordering prediction, in which the model predicts the exact ordering of the sentences. Stronger claims about the unimportance of BERT's NSP task have been studied by Liu et al. (2019) where they remove completely remove it as a training objective and observe performance gains.

BERT-sentences[1] have previously tried to obtain segment embeddings by introducing slight modifications to BERT's input representation. They add a new special token [EOS] denoting the end of the sentence. The information about one sentence from another goes through this token, and the other undesired information flow is blocked via attention masking, e.g., from tokens of two different sentences. This approach is similar to ours, however, it has still similar complexity to BERT because they could not reduce the number of parameters.

Bai et al. (2020) argue that better contextual representations can be generated from the text encoder with richer positional information. They empirically show that their SegaBERT outperforms BERT on several tasks. Inspired by their idea, we use segment positional embeddings that are dependent on the two sentences, rather than absolute positional embedding like them. We also do not use the paragraph index embeddings as it would not be appropriate in our case. SegaBERT is not trained on NSP, which we suppose is because adding too many positional embeddings would make the task trivial. Since we have to use the NSP task to train segment embeddings, we omit the paragraph index embeddings.

Iter et al. (2020) propose a new training objective that improves the coherence and the distance between sentences. Their training objective is, for a given anchor sentence, the model is trained to predict the text k sentences away by using sampled-softmax to choose among some set of candidate sentences. The candidates are chosen randomly from the neighbouring sentences or different documents from the corpus.

3 LaMBERT

In this section, we introduce LaMBERT, present the details and differences, and provide a direct comparison with BERT.

3.1 Model architecture

In order to make a direct comparison with BERT, we tried to stick as much as possible to BERT's implementation in PyTorch from HuggingFace library [2]. We introduced only the LaMBERT specific differences over BERT as minimal changes to the overall code. The implementation of LaMBERT is published on MLO's GitHub [3].

The model we publish mainly uses the same parameters as $\text{BERT}_{\text{BASE}}$: $L = 12, H = 768, A = 12$, where L represents the number of layers, H is the hidden size, A is the number of self-attention heads. The total number of parameters is 156M.

Input representation: One of the main reasons for the complexity of BERT is the fixed maximum token length, which is 512. Many of the downstream tasks and real-world applications do not require that many tokens. Indeed, the first 90% of BERT's training was only on at most 128 tokens, which allowed them to freeze the weights of the remaining 384 positions and thus train the model faster (Devlin et al., 2018). Nonetheless, once the model is pre-trained, we are required to do computations on the whole model if we fine-tune it for a downstream task or a real-world application. While from one side this sufficiently big length is beneficial as we do not have to worry about extremely long cases, it is also computationally expensive because most of

[1] The code is available at github.com/epfml/bert-sentences.

[2] The implementation of BERT in PyTorch is available in the HuggingFace library at github.com/huggingface/transformers.

[3] The LaMBERT implementation is available at github.com/epfml/LaMBERT.

the time we are wasting resources. The first major difference of LaMBERT is that we have partially variable representation length.

Throughout this report, we refer to an arbitrarily contiguous span of text as a sentence. Input instances consist of two such sentences. We say that the sentence is separated into at most S segments, each of which has M tokens. These parameters are $S = M = 23$ for the model we publish. We can thus have at most $SM = 529$ tokens in a single instance, which is the closest perfect square to BERT's 512.

Separating the input instance into segments allows us to have partially variable input representation. We can indeed represent every sentence with, the lowest multiple of M which is higher than the number of tokens in the sentence, tokens. In other words, any sentence of length n can be represented it with $\lceil \frac{n}{M} \rceil M$ tokens. This is because there are $\lfloor \frac{n}{M} \rfloor$ complete segments — segments of length M, and if n is not divisible by M, then we add padding tokens until it reaches length M. Note that the loss in computational resources in BERT is that we have to add padding in order to reach the fixed number of 512 tokens, whereas in LaMBERT we need to add at most $M - 1$ additional padding tokens for one sentence. It adds additional overhead as we have to reconstruct the sentences from the variable size data structures, but overall it makes all computations on the model faster.

If we only reduced the number of tokens per segment without adding multiple segments, it would be pretty similar to BERT, with a reduced maximum number of tokens in the input. Or we can think of BERT as a special case of LaMBERT with parameters $S = 1$ and $M = 512$. They would not be equivalent as we calculate the [CLS] token manually, and do not use the [SEP] token, but theoretically they would be closely similar.

Since our goal is to make a direct comparison with BERT, we used the same tokenizer that BERT uses — WordPiece tokenizer (Wu et al., 2016). It has a vocabulary of approximately 30,000 tokens. Each sentence is constructed by tokenizing it with the WordPiece tokenizer, and then padding it such that its length is the next multiple of M. LaMBERT was trained on input instances each containing 2 sentences, thus theoretically there are at most $2(M - 1)$ padding tokens.

Aside from splitting the sentence into segments of M tokens, LaMBERT also differs in how the intial embeddings are constructed. The tokens in LaMBERT, similarly to BERT, come from the token embedding matrix. The input token embedding in the model is calculated as sum of the corresponding token embedding, segment embedding of the token (whether it belongs to the first or second sentence) and token positional embedding. In addition to this, we have a segment embedding matrix. The input segment embedding is calculated as sum of the corresponding segment embedding and a segment positional embedding. The idea of adding more positional embeddings has already been studied by Bai et al. (2020) in SegaBERT. Our segment positional embeddings are similar to their sentence index embeddings. However, we do not use paragraph index embeddings. The reason is that SegaBERT is not trained on the NSP task, and we suspect it is because these embeddings would make the task too trivial.

Weight sharing: LaMBERT uses weight sharing for every parameter across different segments. In other words, this means that the first M tokens of one sentence share the weights with the second M tokens of the same sentence, and so on. Different segments of one sentence are linked via the attention mechanism.

A difference with BERT is that we do not use [SEP] token, but we rather separate the input instances ourselves and get the outputs of the model separately. This is possible due to the weight sharing of every parameter for both input sentences.

Attention: Since the input representation of LaMBERT is quite different than BERT, the self-attention mechanism is consequently different. As our aim is to obtain reliable segment embeddings, we want each segment embeddings to influence other segment embeddings and the token embeddings in itself. Or equivalently, for a fixed segment, each of its tokens is influenced by other tokens in that segment as well as the other segments. This intuitively creates deep connections between different parts of the sentences.

This influence is implemented via the attention mechanism. It is calculated in two phases. The first phase is similar to the self-attention is BERT. For every segment, we calculate the attention of the token embeddings with respect to the token embeddings in the segment itself, as well as all other segments. In addition, we calculate the segment embedding with respect to the same vectors. The keys and values are equal, which means that the val-

ues are the same vectors we calculate the attention with respect to. Note that BERT's self-attention computes only the attention of every token with respect to every other token, so there are differences. The second phase is cross-attention. It is used to manually compute the [CLS] embedding by calculating the attention of the [CLS] embedding with respect to itself, and all segment embeddings. Since we use the [CLS] embedding for our NSP task, it directly affects the segment embeddings, which intuitively makes sense and helps the model learn reliable segment embeddings.

We can define these relations between the representations in a more mathematical way. Let:

- M be the number of tokens per segment,
- S be the number of segments in the current training instance,
- w_{ij} be the j-th token in the i-th segment,
- $u_{w_{ij},k}$ denote the k-th layer representation of the j-th token in the i-th segment,
- $v_{s_i,k}$ be the k-th layer representation of the i-th segment,
- $P_{ij}(v_{s_i,k})$ be the position dependent k-th layer representation of the j-th segment with respect to the contents of the i-th segment. P can be anything from a linear map to a deep network or adding some positional embeddings to bias the sentence, and
- H represent our BERT-like model where H_k represent the output of k-th layer.

Using the outputs from the previous layer as inputs and using the function P, we obtain:

$$u_{w_{i1},k+1}, \ldots, u_{w_{iM},k+1}, v_{s_i,k+1} = H_k(u_{w_{i1},k}, \ldots, u_{w_{iM},k}, P_{i1}(v_{s_1,k}), \ldots, P_{iS}(v_{s_S,k})).$$

3.2 Pre-training LaMBERT

The pre-training procedure of LaMBERT is similar to BERT's. We only introduced a few justified changes.

Pre-training data: LaMBERT is pre-trained only on the English Wikipedia dataset. We did not use BooksCorpus (Zhu et al., 2015) because of resource limitations. The preprocessing of the Wikipedia dataset is probably slightly different than the one in BERT. The reason is that the preprocessing code of BERT is not publicly published, so we did our best to replicate the steps they describe. The current model was trained for less than half of an epoch, so our pre-trained model is undertrained. On the final pre-training, we aim to train it on approximately 40 epochs, just like BERT.

Our pre-training also used BERT's trick to reduce the training time. The trick is that for the first 90% of the pre-training, the model is trained only on with pairs of sentences such that their combined length is less than or equal to 6 segments. It results in at most 138 tokens per instance, which is close to BERT's 128 tokens during the first 90% of the pre-training.

Training objectives: We stick to the same objectives that BERT uses in order to make minimal changes for direct comparison. This also gives us the chance to, at least some degree, compare our loss to BERT's.

Our first training objective is masked language modeling, which is identical to BERT's training objective. During the data preprocessing, we replace some of the tokens with [MASK] tag. The objective of the model is to predict the hidden token. This intuitively helps the model learn reliable token embeddings as it fuses information from the left and right contexts. We used the same parameters as BERT: randomly chose 15% of the tokens. Then 80% of them were effectively replaced with the [MASK] token, 10% were replaced with the original token, and the remaining 10% were replaced with a random token.

Our second training objective is NSP. For given two sentences, the task is to predict whether the second sentence comes just after the first sentence in the original data. A sentence in this sense may not represent an actual linguistic sentence, but a contiguous span of text. This is an important objective to help the model learn about cohesion and coherence of the language. It directly affects the model's [CLS] embedding as the model's prediction solely depends on that token. The change in the [CLS] embedding is then directly propagated to the segment embeddings as it is directly calculated from them via the cross-attention mechanism. We introduce a small change over BERT's NSP objective. Inspired by ALBERT's (Lan et al., 2019) analysis on the NSP task, we believe that replacing the second sentence with a randomly chosen sentence from a different document makes the task too trivial. This is why, if we choose to replace

the sentence, we replace it with another sentence from the same paragraph or the same context. This makes the model learn more deeply about cohesion and coherence.

3.3 Fine-tuning LaMBERT

Fine-tuning is relatively simple as the attention mechanism in the Transformer allows us to completely use the information the model has learned during the pre-training phase. It does not matter which specific task we will fine-tune the model for. If we have textual entailment information such as the Multi-Genre Natural Language Inference (MNLI) task (Williams et al., 2017), we put the first sentence of the dataset as the first sentence in the model, and it is followed up by the second sentence from the dataset. Even if we have a multiple-choice question answering task, we do similarly by placing the question as the first sentence followed by a concatenation of the answers.

Once the model is pre-trained, the fine-tuning phase is relatively simple. The fine-tuning time depends on the amount of data for the specific task, but for many downstream tasks, it takes at most a few hours on a single GPU. The pre-training phase, however, according to our estimations, will take between a week and a half to two weeks on 4 GPUs.

4 Evaluation

We did our experiments for the MNLI task (Williams et al., 2017), which is part of the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018).

The MNLI dataset consists of approximately 392,000 human-annotated crowd-sourced pairs of sentences. The task is to predict the relation between the pair of the sentences, among the three possibilities: entailment, neutral and contradiction. The testing dataset consists of *matched* and *mismatcehd* datasets, each of which has 10,000 testing instances. The matched dataset contains instances similar to the ones seen during the training. The examples in the mismatched dataset do not closely resemble any of those seen at training time.

The fine-tuning procedure is pretty straightforward. We prepare the input in the same way as described in the previous section. The first sentence in the dataset is put as the first sentence in the model, which is followed by the second sentence. We add a top-level classifier on top of the [CLS] embedding. This embedding has initially as many dimensions as the hidden size of the model is, which is $H = 768$ in our case. Since the MNLI task has 3 classes, we add a trainable linear mapping that maps this embedding to an embedding of dimension 3. We then use softmax on the three values to predict the final class.

We use a batch size of 32, and fine-tuned the model for 3 epochs over the whole MNLI data. The learning rate was selected as the best in the training dataset among 2e-5, 3e-5, 4e-5 and 5e-5. We carried out an experiment with weight decay rate among 5e-4, 5e-5, 5e-6, and no weight decay at all. We noticed that the higher the weight decay was, the worse the model performed. So in the end we chose not to use any weight decay. We used Adam (Kingma and Ba, 2014) as optimization algorithm with warmup parameter 0.1, and default values for β ($\beta_1 = 0.9, \beta_2 = 0.999$).

We obtained accuracy of 70.7%, combined on both, matched and mismatched datasets. This is less than $\text{BERT}_{\text{BASE}}$'s accuracy of 84.6% and 83.4% on matched and mismatched datasets respectively (Devlin et al., 2018). There are two major reasons for this. First, we noticed an error in the attention masking. More precisely, at some parts the masking was omitted, so the information flow was not blocked, thus the pre-trained model was not working as intended. Second, the model was trained only for the purpose to check if we will get somewhat good results. The total pre-training time was a few hours on a single GPU, and it was trained for less than half of an epoch over the dataset.

During one of the fine-tuning evaluations, by mistake we used different tokenizer, and thus the input representation was different than the pre-training phase. The fine-tuned model's loss, in this case, could not converge to any point and performed poorly with accuracy of about 30%, which is even less than what a majority classifier would achieve. This suggests that even with the attention masking error that there was, and despite the severely undertrained pre-trained model, LaMBERT achieved promising results. Moreover, a simple Continuous Bag of Words model achieves accuracy of 64.7% as reported by Williams et al. (2017). We get better results than it, even though our model has two issues. It additionally suggests promising results once both problems are fixed.

Figure 1 shows how the loss changes over batches in different epochs. We can observe that the

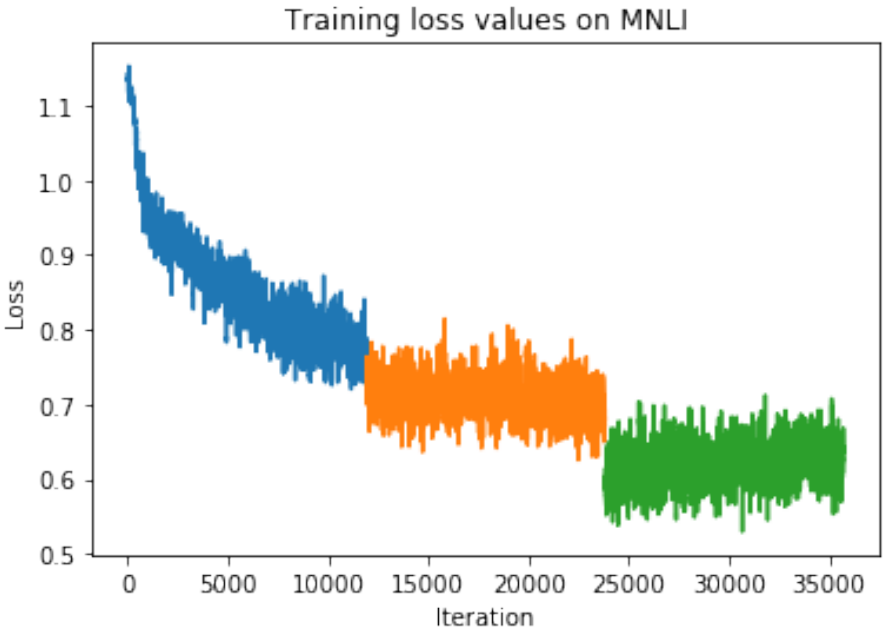

Figure 1: Training loss for three epochs with batch size 32 on MNLI with 25 moving average. The blue, orange and green colors represent the losses for the three epochs respectively.

loss fluctuates, but in general it steadily decreases over time. A similar thing has been reported in BERT (Devlin et al., 2018). The reason for this is that we use a relatively low batch size. In general, the lower the batch size is, the more the loss fluctuates. If we think of the extreme cases when the batch size is only 1 instance, compared to the whole dataset being only 1 batch, there should definitely be many fluctuations in the first case as one training instance may update parameters that have been already updated, and the following instance updates the weights for the first time.

5 Conclusion

We propose modifications over BERT's input representation and consequently changes to the entities for which the attention mechanism is computed. Our approach significantly is faster than BERT by about 3 times, which not only makes the pre-training faster, but also the fine-tuning, and every other consequent application. While there was a major problem with the attention masking for the model we pre-trained, and the model was not even trained for half an epoch, our conclusions about the experiments give promising results.

References

He Bai, Peng Shi, Jimmy Lin, Luchen Tan, Kun Xiong, Wen Gao, and Ming Li. 2020. SegaBERT: Pre-training of Segment-aware BERT for Language Understanding.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning Distributed Representations of Sentences from Unlabelled Data.

Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification.

Dan Iter, Kelvin Guu, Larry Lansing, and Dan Jurafsky. 2020. Pretraining with Contrastive Sentence Objectives Improves Discourse Performance of Language Models.

Yacine Jernite, Samuel R. Bowman, and David Sontag. 2017. Discourse-Based Objectives for Fast Unsupervised Sentence Representation Learning.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. 2015. Skip-Thought Vectors.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach.

Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient Knowledge Distillation for BERT Model Compression.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2017. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.

Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books.