Self-supervised learning for time series classification

EPFL

Author:
Anne-Sophie Van de Velde

Supervisor:
Professor Martin Jaggi
Co-Supervisor:
Jean-Baptiste Cordonnier

Abstract

Time series classification (TSC) is an important and challenging problem in machine learning. In this work, we tackle the problem of TSC by first applying a Bidirectional Encoder Representations from Transformers (BERT) model, and then applying a convolutional neural network (CNN) for the classification. We report suboptimal results compared to the state-of-the-art, as the model is overfitting early in the training process. We believe that this issue might be overcome by tuning the hyperparameters more carefully.

1 Introduction

Time series classification (TSC) has been a challenging problem in machine learning and statistics for many decades. Due to their natural temporal ordering, time series data are present in almost every task that requires some sort of human cognitive process, e.g. bioinformatics, biomedical engineering or even econometrics.

The work of Franceschi et al. [2] aims to classify time series using an unsupervised representation learning with causal Convolutional Neural Network (CNN). Based on the recent success of Transformers [4] on machine translation, we try to combine the unsupervised representation learning of [2] with a Transformer to improve the accuracy on the standardized datasets of the UCR archive[1]. More specifically, we propose a simplified version of the BERT model presented in [1], i.e. we use one hidden layer and two attention heads as

[1] https://www.cs.ucr.edu/~eamonn/time_series_data/

the encoder. Then, we put a CNN on top of the model to assess its capacity at classifying time series.

We benchmark our results with a simple CNN using positional encoding. Our results are consistent across all datasets that we trained on: while the benchmark performed just under the state-of-the-art results of Franceschi et al [2], the Transformer approach failed to leverage the representation learned in the pre-training and is overfitting early in the training process. This might be related to the issue raised in [3], where the authors argue that non-trivial efforts are required in designing learning rates schedulers and optimizers.

This project is organized as follows. In Section 2 we specify the exact architecture of the representation and classification architectures. We also outline the strategy used to assess the performance of our models. In Section 3, we present the experimental results conducted. We present our results on the Worms dataset of the UCR archive, as it is representative of the behavior we observed.

2 Model elaboration

As explained in Section 1 our goal is to classify time series via first learning an encoder based on the BERT architecture. We will *pre-train* our encoder in a self-supervised manner using the so-called *triplet loss* defined in [2]. Then, we apply a CNN on top of our encoder to perform the original classification task.

2.1 Encoder: positional embedding and simplified Bert

Positional encoding In order to learn a representation for any univariate time series in $\mathbb{R}^n$, we first project the input into $\mathbb{R}^h$. We then add a positional embedding to this projection, yielding to an output of the form

$$\text{output} = W \times \text{input} + b,$$

where $W \in \mathbb{R}^{h \times n}$ and $b \in \mathbb{R}^h$. The output is then normalized via the LayerNorm function described in [4] and a dropout is applied.

Encoder We use a simplified version of the Bert encoder described in [1], i.e. we use $L = 1$ hidden layer, a hidden size of $H = 128$ and $A = 2$ attention heads.

Altogether, the positional encoding followed by the encoder yields a representation in $\mathbb{R}^H$ of the input time series of size $\mathbb{R}^n$, with n arbitrary.

2.2 Unsupervised training

In our *pre-training* phase, we seek to train an encoder-only architecture. To this end, we use the *triplet loss* introduced in [2] that follows the word2vec's intuition. The objective to minimize is defined by

$$-\log\left(\sigma\left(f(x^{\text{ref}};\theta)^{\intercal} f(x^{\text{pos}};\theta)\right)\right) - \sum_{k=1}^{K} \log\left(\sigma\left(f(x^{\text{ref}};\theta)^{\intercal} f(x_k^{\text{neg}};\theta)\right)\right),$$

where $f(\cdot;\theta)$ is the encoder, x^{ref} is a random subseries of a given time series y_i, x^{pos} is a subseries of x^{ref} and x_k^{neg} are random subseries of different time series y_j (for several j) and σ is the sigmoid function. This loss pushes the computed representation to distinguish between x^{ref} and x^{neg}, and to assimilate x^{ref} and x^{pos}.

2.3 Classification via CNN

To perform the classification task, we consider two CNNs described in Table 1.

Layer operation	Parameters CNN$_1$	Parameters CNN$_2$
1D Convolution	$C_{\text{out}} = 512$; Kernel 10; Stride 5	$C_{\text{out}} = 128$; Kernel 10; Stride 5
Activation function	ReLU	ReLU
Max Pooling	Kernel 3	Kernel 3
1D Convolution	$C_{\text{out}} = 256$; Kernel 2; Stride 2	$C_{\text{out}} = 32$; Kernel 2; Stride 2
Activation function	ReLU	ReLU
Mean pooling	-	-
Fully Connected	(256, 32)	$(32, 8)$
Activation function	ReLU	ReLU
Fully Connected	(32, 5)	$(8, 5)$

Table 1: CNN architecture for our classification task on the Worms dataset of UCR. CNN$_1$ takes as input a tensor of dimension (B, h, l) and outputs a tensor of dimension $(B, 5)$. CNN$_2$ takes as input a tensor of dimension $(B, 1, H)$ and outputs a tensor of dimension $(B, 5)$.

2.4 Benchmarks

To assess whether the representations learned by our encoder is useful, we consider the following two models:

1. **EmbConv**: positional embedding described in 2.1 with $h = 1024$ followed by CNN_1.
2. **BenchBert**: positional embedding and encoder described in 2.1 followed by CNN_2.

3 Experimental results

We present the results obtained on the Worms dataset, found in the UCR archive. This dataset comprises a training set and a testing set of 181 and 77 time series respectively, each of dimension 900. They are classified in 5 different classes. We choose to show the results of this experiment, as we obtain very similar results on other dataset in the archive and on toy dataset.

3.1 Benchmarks

The results of **EmbConv** and **BenchBert** are presented in Figure 1. We used batch sizes of 16 and 8 for the training set and testing sets respectively. We used the Adam optimizer with adaptive learning rate initialized at 0.001 and decreased by a factor of 0.5 every 200 epochs.

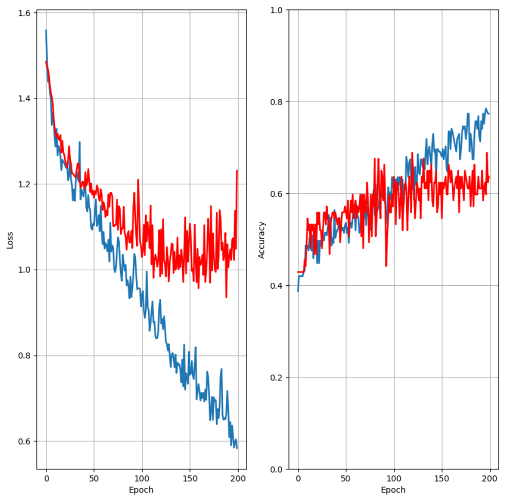
Notice that **EmbConv** reaches an accuracy of 69% on the testing set, where the state-of-the-art method presented in [2] reaches an accuracy of 74%. However, **BenchBert** only reaches 57% of accuracy with clear signs of overfitting. We observe early signs of overfitting on **BenchBert** whereas the training of **EmbConv** behaves more smoothly.

3.2 Classification and fine-tuning

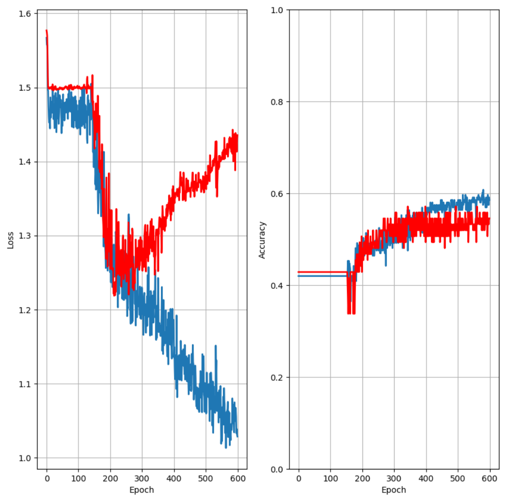
We use the positional embedding and the encoder and the triplet loss described in 2 with $K = 8$ and perform a *pre-training* of 1500 steps on the training set. We used the Adam optimizer with adaptive learning rate initialized at 0.001 and decrease by a factor of 0.5 every 200 epochs.

In order to perform classification, we then re-use our **BenchBert** with the encoder weights initialized at the pre-training weights. Then, we proceed to train our model once by only training the convolutional weights (i.e. we keep our encoder fixed and only train the CNN_2 weights), and once by training all the weights of the network (*fine-tuning*). We used batch sizes of 16 and 8 for the training set and testing sets respectively. We used the Adam optimizer with adaptive learning rate initialized at 0.001 and decreased by a factor of 0.5 every 200 epochs.

Figure 2 shows that both methods produce early signs of overfitting.



(a) **EmbConv**

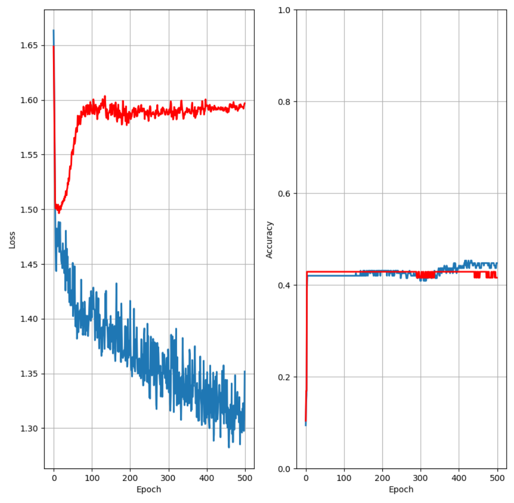


(b) **BenchBert**

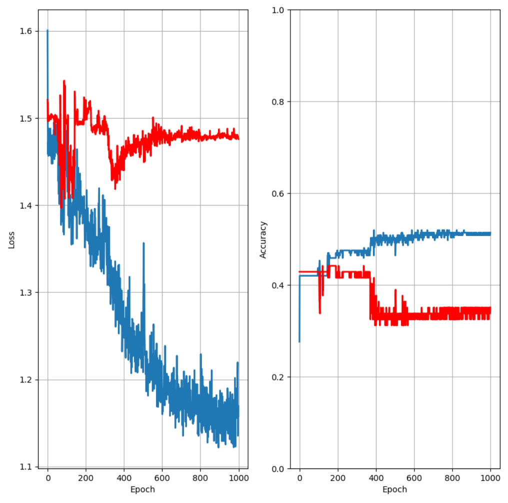
Figure 1: Loss and accuracy of **EmbConv** and **BenchBert** on the Worms dataset on 200 and 600 epochs respectively. Blue line: training set results. Red line: testing set results.

4 Conclusion

In this work, we tried to combine an unsupervised representation of univariate time series with a BERT architecture to assess the performance on time series classification. We observe that while our benchmark **EmbConv** performs just under the state-of-the art results of Franceschi et al. [2], the simplified BERT **BenchBert** fails to produce viable results and is prone to over-fitting, even when performing a pre-training. In that regard, further investigation is needed to better tune the learning rate schedule and carefully choose the optimizer.



(a) CNN$_2$ training



(b) Fine-tuning

Figure 2: Loss and accuracy of **BenchBert** after pre-training on the Worms dataset. Blue line: training set results. Red line: testing set results.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, *BERT: Pre-training of deep bidirectional transformers for language understanding*, 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) (Minneapolis, Minnesota), Association for Computational Linguistics, June 2019, pp. 4171–4186.

[2] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi, *Unsupervised Scalable Representation Learning for Multivariate Time Series*, Thirty-third Conference on Neural Information Processing Systems, December 2019, Poster.

[3] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han, *Understanding the difficulty of training transformers*, 2020.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, 2017.