ZPJa: Reinvent the Wheel - Transformer Encoder

Bc. Ladislav Ondris
xondri07@fit.vut.cz

Ing. Martin Dočekal
idocekal@fit.vut.cz

Abstract

It has been shown that the new transformer model outperforms traditional neural networks like CNN or LSTM in NLP tasks. This work describes the architecture of the BERT transformer and the classifier made on top of it to show its correctness on a text classification on the AG News dataset. The model is implemented from scratch using only linear operations from the PyTorch library. Pre-trained weights from HuggingFace are used and fine-tuned on the AG News dataset. Evaluation produces an error rate of 14.86%. The score is reasonable given that hyperparameters were not sophisticatedly tuned for the best results.

1 Introduction

A transformer is renowned for its performance in NLP tasks. This work will implement a BERT transformer from scratch and explore how would the transformer perform without special treatments during fine-tuning and training simply as is on a single dataset using freezed pre-trained weights.

2 Transformer Encoder

The transformer encoder is a part of a transformer model (Vaswani et al., 2017), which builds upon an attention mechanism that has become popular in recent models not only for NLP but also for image processing. Before transformers, the attention mechanism was successfully used in recurrent neural networks. The transformer relies only on the attention mechanism.

Attention mechanisms can be divided into several types. The transformer uses a self-attention, which relates different positions of the same input sequence. In other words, it looks at relations between each pair of tokens in that sequence.

The model first transforms input sequences into embeddings, adds positional encoding, and performs self-attention multiple times. The following sections provide the necessary details of the transformer encoder and the operations used as a part of it.

2.1 Softmax

The softmax function is a generalization of the logistic function. It normalizes vector $\mathbf{x}$ into a vector of probabilities. Implementations often subtract a maximum value from the vector before computing softmax to make it computationally stable.

$$\text{softmax}(\mathbf{x}) = \frac{e^{x_i - \max(\mathbf{x})}}{\sum_{j=1} e^{x_j - \max(\mathbf{x})}}$$

2.2 Layer Normalization

Layer normalization (Ba et al., 2016) normalizes the inputs $\mathbf{a}$ of a layer by its mean μ and variance σ^2. It also uses trainable vectors $\mathbf{w}$ and $\mathbf{b}$, which perform an affine transformation of the normalized inputs.

$$\mathbf{h} = \frac{\mathbf{a} - \mu}{\sqrt{\sigma^2 - \epsilon}} \cdot \mathbf{w} + \mathbf{b}$$

$$\mu = \frac{1}{H} \sum_{i=1}^{H} a_i$$

$$\sigma^2 = \frac{1}{H} \sum_{i=1}^{H} (a_i - \mu)^2$$

2.3 Dense Layer

A dense layer, also known as a linear layer, is a layer commonly used in feed-forward neural networks or used for transformations of its inputs $\mathbf{a} \in \mathbb{R}^I$ by a trainable matrix $\mathbf{W} \in \mathbb{R}^{I \times H}$ and a trainable vector $\mathbf{b} \in \mathbb{R}^H$. The parameters are initialized with Xavier uniform distribution for better convergence.

$$\mathbf{h} = \mathbf{a}\mathbf{W} + \mathbf{b}$$

2.4 GELU

GELU (Hendrycks and Gimpel, 2016) is an activation similar to ReLU but showing a better performance at the cost of being more computationally expensive.

$$\text{GELU} = 0.5x(1 + \tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)))$$

2.5 BERT Classifier

The BERT (Devlin et al., 2019) is a transformer whose architecture is similar to the vanilla transformer but with a few architectural changes. Special tokens—[CLS], [MASK], [PAD], [UNK], and [SEP]—can be inserted in the input sequence so it can be used for a wide range of downstream tasks such as text classification, question answering, and natural language inference.

Figure 1 demonstrates an input sequence to which a [CLS] and a [SEP] tokens were appended. The sequence is then tokenized, after which it can be fed into the model.

Image 2 shows a model that was used in this work for text classification. Viewed from above, the BERT model is followed by a linear layer, reducing the output to N classes.

The BERT model itself is composed of three main parts. The first one is an embeddings layer, which creates embeddings from the tokenized sequence. The second part is a sequence of twelve encoder layers, and each of them computes self-attention. The pooling layer is the last layer of BERT and reduces the output of encoder layers.

2.6 Embedding Layer

Language models often use trainable embeddings. Each token in the vocabulary is represented by a vector of fixed size d_e. Thus, the embeddings are represented by a parameter matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d_e}$, where $\mathcal{V}$ is the vocabulary. The layer performs mapping $f : \mathbb{R}^s \to \mathbb{R}^{s \times d_e}$, where s is the number of tokens in a sequence.

Embeddings in BERT are made of three types of embeddings—word embeddings, positional embeddings, and segment embeddings. All of these embeddings are summed together and sent to the next layer of the model.

Each token is converted according to the process described above to a fixed-length vector called word embedding or token embedding. It is a different representation of the token that the model can more easily work with. BERT uses embeddings of length 768.

Positional embeddings ensure that a token's position is known to the model. The vanilla transformer used pre-computed values from the sinus function. BERT, however, trains these embeddings together with the rest of the model. Thus, the embeddings are also represented by a parameter matrix.

2.7 Encoder Layer

The most important part of the model is the encoder layer. The major building block of this layer is a multi-head attention module, which computes scaled dot-product attention, followed by linear layers, residual connection, and layer normalizations. The architecture is shown in Figure 3.

2.7.1 Scaled Dot-Product Attention

The attention mechanism allows the model to learn relations between different tokens in the vocabulary. The input consists of queries, keys, and values of dimension d_k.

The first step is computing the dot product of Q and K matrices. This results in a matrix where each element contains information of relatedness between tokens in the sequence. Next, it is divided by $\sqrt{d_k}$ to decrease the values. Otherwise, too large values would cause the softmax function to return values that are either close to one or too close to zero, which would make training difficult. The softmax function then normalizes the values. Then, multiplying by V matrix results in a matrix where each row represents how each token corresponds to the whole context.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V}$$

2.7.2 Multi-Head Attention

Query, key, and value matrices are split in the multi-head attention module in the last dimension into h parts. Scaled dot-product attention is performed on each split in parallel, producing several heads $\mathbf{h}_i \in \mathbb{R}^{s \times d_k}$ where $d_k = \frac{d_e}{h}$:

$$\mathbf{h}_i = \text{Attention}(\mathbf{Q}_i\mathbf{W}_i^Q, \mathbf{K}_i\mathbf{W}_i^K, \mathbf{V}_i\mathbf{W}_i^V).$$

$\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d_e \times d_k}$ are parameter matrices.

All heads are then concatenated back together and multiplied by a parameter matrix $\mathbf{W}^O \in \mathbb{R}^{d_e \times d_e}$.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{concat}(\mathbf{h}_1, \ldots, \mathbf{h}_h)\mathbf{W}^O$$

Input Sequence	[CLS]	The	computer	age	is	just	beginning	.	[SEP]
Tokenized Sequence	101	1996	3274	2287	2003	2074	2927	1012	102

Figure 1: First, two special tokens, [CLS] and [SEP], are added to the sequence. Then, the sequence is tokenized before it is fed into BERT.

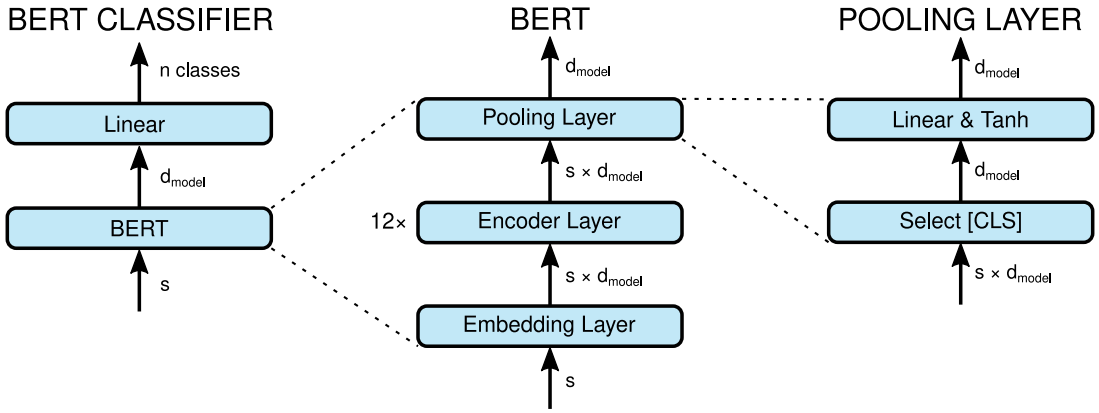

Figure 2: The architecture of a classification model using BERT. For a classification task, a linear layer is added atop BERT. The BERT itself consists of three parts—an embedding layer, a stack of encoder layers, and a pooling layer.

2.8 Pooling Layer

The length of each sequence is different. One way to handle this during classification is to take only the first token—at the position of [CLS]—of the encoded sequence and transform it using a linear layer. Thus, the linear layer does not deal with an arbitrary sequence length.

3 Experimental Setup

The correctness of the implemented BERT model was verified by applying it on a downstream task. A classifier was built on top of BERT for text classification. The model was fine-tuned and evaluated on the AG News dataset.

3.1 AG News Dataset

The dataset consists of four types of text sequences—World, Sports, Business, and Science/Technology. The training set contains $120,000$ samples and the test set $7,600$. All labels have an equal number of samples, so accuracy is a good measure of performance. The verage sequence length is 44 and the maximum length is 221.

An example of such a sequence from the training set from the World category: *US bill aims to jail film pirates Using video cameras to record cinema films would become a state-wide crime under a proposed US law.*

3.2 Fine-tuning for Text Classification

The model described in Section 2.5 has 109.5 million parameters. Pre-trained weights of the BERT model were loaded from the HuggingFace library (without the last linear layer). These weights were frozen during fine-tuning on the AG News dataset. Thus, only 600 thousand parameters were trainable.

The model parameters were set to the same values as in HuggingFace in order to preserve weights compatibility. For example, the number of heads in a multi-head attention module is 12, $d_{model} = 768$, $d_{ff} = 3072$, $|\mathcal{V}| = 30522$. Due to the short length of sequences in the dataset, there was no need for special treatment of sequences longer than 768 tokens.

The Adam optimizer was used during fine-tuning with a learning rate of $5 \cdot 10^{-6}$. The model was trained in Metacentrum[1] on a single GPU for a single epoch, taking about 15 minutes per epoch. Further training did not improve the final score.

4 Results and Analysis

Only a single epoch of fine-tuning on the AG News dataset the model reaches an accuracy of 85.14 %. Table 1 compares our model to current state of the art models on AG News dataset (Pap, 2021).

[1] https://metavo.metacentrum.cz/en/index.html

Model	Type	Error [%]	Year
XLNet (Yang et al., 2019)	Transformer	4.45	2019
BERT-ITPT-FiT (Sun et al., 2019)	Transformer	4.80	2019
L MIXED (Sachan et al., 2020)	LSTM	4.95	2020
ULMFiT (Howard and Ruder, 2018)	LSTM	5.01	2018
DRNN (Wang, 2018)	LSTM	5.53	2018
CNN (Johnson and Zhang, 2016)	CNN	6.57	2016
DPCNN (Johnson and Zhang, 2017)	CNN	6.87	2017
BERT	Transformer	**14.86**	2021

Table 1: Comparison of our BERT transformer model to state-of-the-art models evaluated on the AG News dataset. The error says what percentage of the test samples was classified incorrectly.

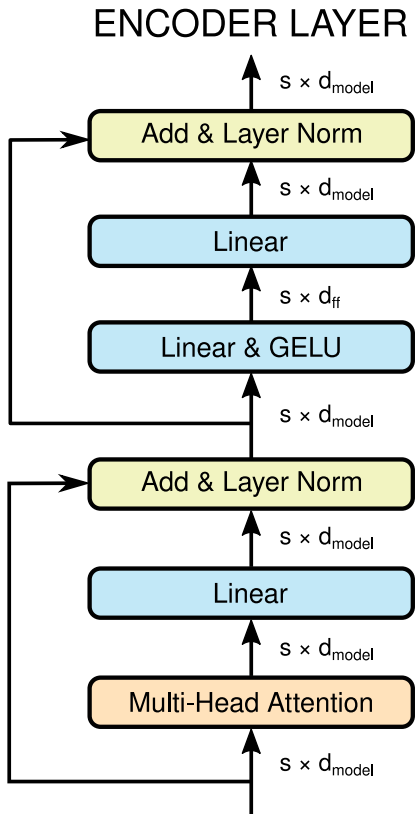

Figure 3: The encoder layer is the building block of a transformer. The first layer is a multi-head attention module whose core function is the scaled dot-product attention. The computed attention is transformed using a linear layer. Then, a residual connection adds the transformed output to the input and normalizes it. After that, two more linear layers follow. The first one increases dimension from d_{model} to d_{ff} and the second one reduces the dimension back to d_{model}. Finally, skip connection and normalization compose the final layer. s denotes the sequence's length.

An interesting trend can be noticed in the table—first places belong to transformer models, followed by LSTM-based ones with CNNs in pursuit. BERT-ITPT-FiT (Sun et al., 2019) is a model whose authors conducted exhaustive experiments on BERT to find out how to best fine-tune BERT for text classification. First, they further pre-trained it specifically for text classification. Only then, they fine-tuned it on concrete datasets. Among other things, they investigated which layers are most important, or effects of learning rate in different layers. They showed that BERT achieves high performance if sufficient attention is paid to fine-tuning and its intricacies.

An important discovery is that our fine-tuned BERT did not outperform state-of-the-art CNNs and LSTMs most likely since only the last linear layer was trained while the whole BERT was frozen. BERT model is pre-trained on multiple tasks, and, thus, freezing layers during fine-tuning results in sub-optimal performance.

Table 2 displays the confusion matrix for the test dataset. The model is confused mostly between Business and Sci/Tech categories, producing 6.36% of erroneous predictions.

Inference time was compared between HuggingFace's and our implementation of BERT. The comparison was performed on a random sequence of 221 tokens, which is the maximum sequence length in the AG News dataset. The HuggingFace's model takes on average about 335 milliseconds, while our implementation takes about 426 milliseconds. The time was measured on a machine with Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz.

5 Conclusion

This work implemented a transformer, specifically BERT, using only linear operations from the PyTorch library. Several details such as computational stability had to be considered during the implemen-

		Predicted label			
		World	Sports	Business	Sci/Tech
True label	World	21.43	0.99	1.58	1.00
	Sports	0.58	23.51	0.54	0.37
	Business	1.53	0.46	19.66	3.36
	Sci/Tech	1.12	0.34	3.00	20.54

Table 2: The correctness of implemented BERT transformer was verified on text classification task. The model was fine-tuned on the AG News dataset. The table shows a confusion matrix in percentages for an evaluation on the test part of the dataset. Each category has the same number of samples.

tation. The implementation was verified by creating a classifier on top of the transformer and performing text classification on the AG News dataset.

Overall, the evaluation of the classifier's performance showed satisfying results and most importantly that the transformer was implemented correctly. However, for more precise text classification, it is necessary to fine-tune the whole model and not only the last layer. It is worth paying attention to the intricacies of fine-tuning, including experimentation with learning rates for different layers, or pre-training on the specific downstream task before the actual fine-tuning.

References

2021. Text classification on ag news. https://paperswithcode.com/sota/text-classification-on-ag-news. Accessed: 2021-12-15.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 328–339. Association for Computational Linguistics.

Rie Johnson and Tong Zhang. 2016. Supervised and semi-supervised text categorization using LSTM for region embeddings. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 526–534. JMLR.org.

Rie Johnson and Tong Zhang. 2017. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 562–570. Association for Computational Linguistics.

Devendra Singh Sachan, Manzil Zaheer, and Ruslan Salakhutdinov. 2020. Revisiting LSTM networks for semi-supervised text classification via mixed objective function. *CoRR*, abs/2009.04007.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Baoxin Wang. 2018. Disconnected recurrent neural networks for text categorization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2311–2320. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.