Evaluating Transformers as Embedding layers

Daniel Gedon
Department of Information Technology
Division of Systems and Control
Uppsala University
daniel.gedon@it.uu.se

Abstract

In this course project we study the use of pretrained transformers as embedding layers for NLP tasks. We compare embeddings on word level, character level and combined word-character level for the task of word sense disambiguation (WSD). We pretrain the transformers on a large corpus on data and then finetune it for the WSD task. As a comparison method we use standard embedding layers. The complete code is completely self written for pretraining and finetuning and it is freely available.

1 Introduction and Background

For many NLP tasks, the use of large pretrained language models is the standard method nowadays. Pretrained attention based models have pushed the state of the art from its first use in the NLP area by Bahdanau et al. [2016], which adds a RNN on top of a self-attention mechanism for a machine translation task. Vaswani et al. [2017] presented a new model structure, the transformer, which is only based on attention and does not use an additional RNN or CNN. This model is pretrained in a self-supervised fashion on a large corpus of freely available and unlabeled language data.

Two main pretrained model categories can be distinguished for our purposes. **Autoregressive models** make use of the decoder in the transformer architecture. These models generate a language model by predicting the next token in a sequence with information only from previous tokens. Well known examples are GPT (Radford et al. [2018]), GPT-2 (Radford et al. [2019]) and the most recent GPT-3, which yield impressive results in text generation. The second category are **autoencoding models**. These models rely on the encoder in the transformer architecture. The model is not limited to previous tokens but can look at all tokens. The inputs are corrupted to have a denoising autoencoder, similar to Vincent et al. [2010]. The most famous example is BERT (Devlin et al. [2019]) and its further improvements like RoBERTa (Liu et al. [2019]) or DistilBERT (Sanh et al. [2020]). In practice, a certain percentage of tokens in the input is corrupted by masking them out. The model can use information before and after that token in order to predict the masked tokens and therefore learn a language representation.

In this course project we use transformers on different text abstraction levels (i.e. word level, character level and a combination of both) as pretrained models and compare their performance on a common, simple NLP task. The goal is not to push the limits in terms of overall performance but to compare the different levels of abstraction in the transformer. Our code is freely available [1].

[1] https://github.com/dgedon/nlp_transformer_embeddings

2 Methods

The main focus is to compare the performance of different text abstraction levels within a transformer as embedding layer. In order to evaluate the overall performance, a comparison with model based on standard embedding layers on the same abstraction levels is drawn.

We make use of a BERT-like architecture and learning objective based on the native PyTorch toolkit (Paszke et al. [2019]). This means that we mask out 15% of the input tokens and predict these tokens. Out of the 15% of masked tokens, 80% are replaced with a special *masking* token, 10% are replaced with a random token and 10% remain to original token value.

3 Experiment Setup

3.1 NLP Task

We are considering the task of word sense disambiguation (WSD). A specific word in a sentence can have multiple meanings which can be determined by the context. Transformer models build a token representation within the specific tokens. Therefore the hypothesis is that the transformer model should outperform the simple embeddings.

We treat the WSD task as a document classification task. This means that we do not directly consider the word position as input the model but use a bag of tokens. A higher performing architecture would be to consider the position of the word to disambiguate and take the context into account by e.g. a (bi-)directional RNN.

3.2 Datasets

For pretraining the transformer models we take the WikiText-103 dataset with 103 million training word token and 217 thousand validation word token. For pretraining we limit the word level model to 50 million word tokens and for the character level model to 75 million character tokens. We consider more character level tokens since, each words consists of considerably more characters.

For finetuning the WSD task we consider a provided dataset by the course organizers [2]. This dataset is pre-processed for an easy use.

3.3 Model and Training Details

We choose a model of small size such that we obtain a not too high final performance in order to compare models more reasonably than if the final performance would be in the top range. Therefore for the simple comparison models we use an embedding dimension of 128.

For the transformer model we choose the same embedding dimension for comparison reasons. A total of 4 transformer encoder layers with 8 heads and an inner dimension of 256 is chosen. For word level models we choose a sequence length of 128 token and for character level model of 512. GeLu activation function with a dropout rate of 0.1 is selected. Training is performed with a learning rate of $5 \cdot 10^{-5}$ which is linearly decreased to zero in each of the 40 epochs. The highest performing model in terms of minimum cross-

For finetuning a fully connected 2-layer network with 500 hidden unit and a dropout rate of 0.5 is chosen. For training the transformer a learning rate of $5 \cdot 10^{-5}$ and for the simple comparison models of $1 \cdot 10^{-3}$ is empirically chosen. The learning rate is reduced on the milestones $40, 60, 80$ epochs by a factor of 3. A total of 100 epochs is learned and the highest performing model in terms of validation set accuracy is stored.

As tokenizer we use for word level the hugginface RoBERTa fast tokenizer[3] in order to obtain word pieces. This is the only external piece of code outside of the native PyTorch libraries that we utilize. For character level we use all the characters and simply lowercase them. We obtain a word vocabulary size of 50.265 and a character level vocabulary of 822.

[2]see `http://www.cse.chalmers.se/~richajo/waspnlp2020/a1/a1_data.zip`

[3]see `https://huggingface.co/transformers/model_doc/roberta.html?`

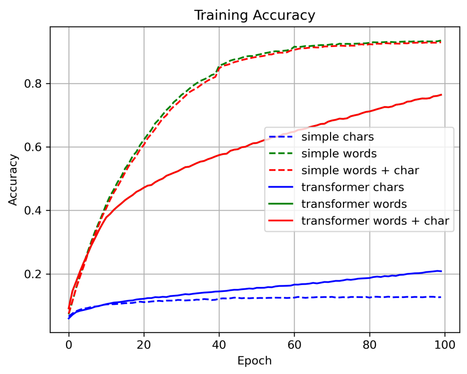
4 Results

We measure the capabilities of the pretrained language model by the percentage of correctly predicted token. Within a small number of epochs the word level model correctly predicts about 58.0% of the tokens and for the character level 47.5%, respectively. This is achieved while only correcting the model for the 15% of masked tokens. A short test with a larger model with 6 transformer layers achieves about 70% accuracy on word level. However, the model is kept small in order to reduce computation times and increase comparability with the simple embeddings.

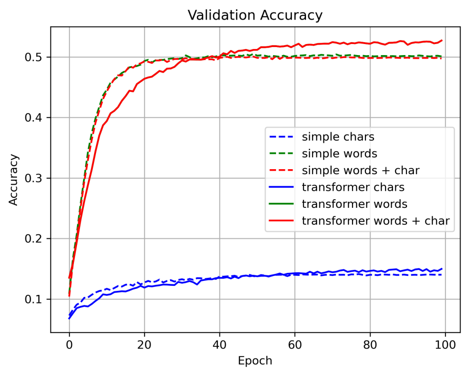
The accuracy on the training sett and on the held out validation set (30% of the training data) is plotted in Fig. 1a and Fig. 1b as a function of the epoch in the training process. The curves for the simple embeddings are dashed, while the ones for the transformer are solid. Same level of text abstraction is shown with the same color for the embeddings and the transformer. Multiple observations are made:

- Observing the training accuracy plot we see that the transformer model has not yet fully converged.
- Comparing both plots we can see that all models overfit to the training data quite heavily despite using a dropout regularization of 0.5.
- In the validation accuracy plot we see that in general the transformer model perform slightly better than the simple embedding models. However, the transformer models converge slower.
- Comparing the different models within the validation accuracy plot, we can see that the character level model performs quite poorly with a final performance of about 15%. This is reasonable since the we are aiming for word disambiguation. Since we are using a word piece tokenizer (from hugginface RoBERTa implementation), we hypothesize that the character level abstraction is not very beneficial. This hypothesis also explains that there is barely any performance gain between the word level and word+character level models.

The last two observations are also backed up by the performance of the models on the separate test set, see Tab. 1. From this table we see that the improvement from the simple embedding models to the transformer models is not very significant. However, we assume that with a larger transformer model (more layers, more heads) this performance improvement increase. This assumption is based on the simple experiment of learning a larger transformer, which yielded a more precise language representation as explained above. Experiments with larger transformer models were not conducted due to temporal constraints.



(a) Accuracy of training



(b) Accuracy of validation

Figure 1: Accuracy curves for all models over the training epochs.

Table 1: WSD test accuracy

Method	Embeddings	Transformers
Char-level	13.96%	14.64%
Word-level	49.85%	52.19%
Char+Word-level	49.96%	TODO

5 Conclusion

This course project compared transformers as embedding layers for the WSD task where different levels of text abstraction (character level, word level and combined) are considered. The project code is completely self written which ensured a complete understanding of the transformer technology and BERT learning objective. We only take the RoBERTa tokenizer as external library for improved word piece tokenization. We observe that the small sized transformer in general outperform simple embeddings and that word level abstraction is sufficient for the chosen WSD task.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, May 2016.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*, July 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. page 12, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. page 24, 2019.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv:1910.01108 [cs]*, February 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, June 2017.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. page 38, 2010.