CS6910 Fundamentals of Deep Learning
Project Report

Vasudev Gupta(ME18B182) **Rishabh Shah**(ME18B029)

January 31, 2021

Contents

1 Introduction **2**
- 1.1 Adapters . . . 2
- 1.2 Mask Predict . . . 2
- 1.3 Model Architecture . . . 2
- 1.4 Training Objective . . . 3

2 Paper Analysis **3**
- 2.1 Simultaneously training BERT and adapters . . . 3
- 2.2 Role of length token . . . 3
- 2.3 Effect of number of iterations . . . 3
- 2.4 Why not autoregressive decoding . . . 3

3 Results and Observations **4**
- 3.1 Number of parameters . . . 4
- 3.2 Training Graphs . . . 4
- 3.3 Inference . . . 4
  - 3.3.1 Predictions from trained model . . . 4
  - 3.3.2 Predictions from untrained model . . . 5

4 Conclusion **5**

5 Appendix **5**

---

1 Introduction

This paper introduced ways to incorporate BERT like models for seq2seq tasks. Their idea is based on algorithm introduced by Microsoft called Mask-predict in order to handle bidirectional nature of BERT in decoder side. Other idea is to use pre-trained models which already posses great understanding of language and adapt its understanding to particular task by tuning just a small set of weights.

1.1 Adapters

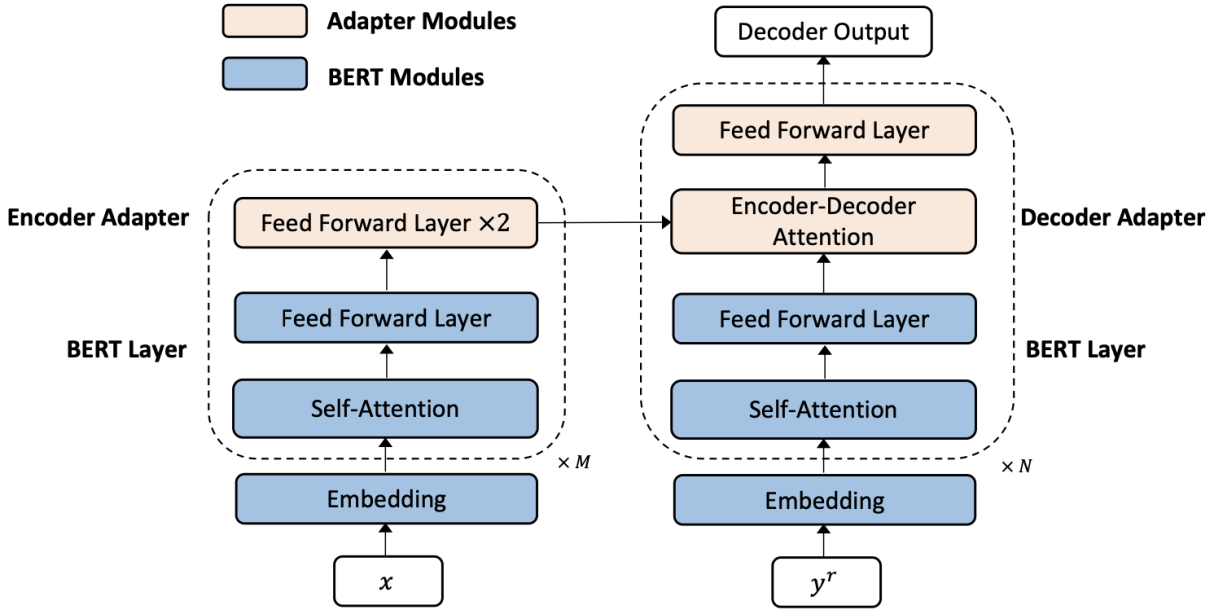
Adapters are nothing but light-weight network added over bigger network in order to adapt bigger model to some task without changing its weights. The major benefit of having adapters is that parameters of the pre-trained language model and task-specific adapters are decoupled, thus bypassing the catastrophic forgetting problems. Model deployment becomes easy, since task-specific weights are of small size as compared to bigger network.

1.2 Mask Predict

Mask-Predict algorithm results in predicting entire sequence in parallel. At each iteration, the algorithm selects a subset of tokens to mask, and then predicts them (in parallel) following the bidirectional masked language modeling criteria. This whole process is performed for some specific number of iterations (typically 10).

1.3 Model Architecture

Over each layer of BERT, small feed-forward networks (i.e. adapters) are added in both encoder and decoder side. Cross-attention layer is also introduced in order to combine encoder-decoder. Separate token for predicting length embedding is added in encoder side for predicting target sequence length. Language modeling head is added over decoder, for predicting tokens probabilities from decoder final hidden states.



Above figure shows the architecture presented in this paper. Simple feed-forward network are added over BERT in order to adapt it to translation task. Cross-attention layer is also introduced in decoder side for connecting encoder-decoder.

1.4 Training Objective

The objective function here, consists of two parts that are added together - word prediction loss and the length loss. The word prediction loss is the normal cross-entropy loss over predicted sequence while for length loss, a special length token is added to the encoder input and its encoder output is taken as representation, based on which target length is predicted and length loss is calculated.

Note: BERT weights in encoder and decoder size are kept frozen during training; only randomly initialized weights are trained.

2 Paper Analysis

2.1 Simultaneously training BERT and adapters

We wanted to understand the impact of training BERT and adapters simultaneously and hence conducted some experiments in which we trained both. But this model could not perform well and loss function was above as compared to training only adapters case. One possible reason could be that adapter weights are randomly initialized while BERT weights are pre-trained; so if we train both simultaneously, BERT weights are getting destroyed during initial rounds of training. And this will introduce imbalance in training.

2.2 Role of length token

Since we need the model to predict length of target sequence during inference, hence we appended special loss in overall loss. Now major problem is that if model predicts the wrong length, it will output wrong translation regardless of how good translator it is. This paper suggested some ways to tackle this problem by taking top 'K' lengths and keeping the sentence of maximum probability.

But still, we observed that model is outputting wrong length more often. Possible reason can be that we need to train model for more time. To handle this, we calculated the average length in both source and target dataset and made model translate based on average target length each time.

Other better solution can be calculating difference of target and source for each and every sample, mean it over complete data adding it to respective length of source to predict target length.

2.3 Effect of number of iterations

We observed that no of iterations is very important for getting better translation but beyond some particular no of iterations; translations were not improving.

This shows that initially model is quite unsure of what it is predicting. Possible reason could be because in 0th iteration, decoder is receiving only '[MASK]' tokens at all positions and it couldn't differentiate tokens. But as no of iteration is increasing, decoder is getting more and more information (in form of more unmasked tokens) which is improving its performance.

2.4 Why not autoregressive decoding

Since during pre-training stages, BERT utilizes bi-directional context information and ignores conditional dependency between tokens, the parallel sequence decoding algorithm Mask-Predict is used instead of autoregressive decoding during inference time. Now, this ensures that model is expected to behave in similar way during pre-training, fine-tuning and inference which is very essential for any model to work.

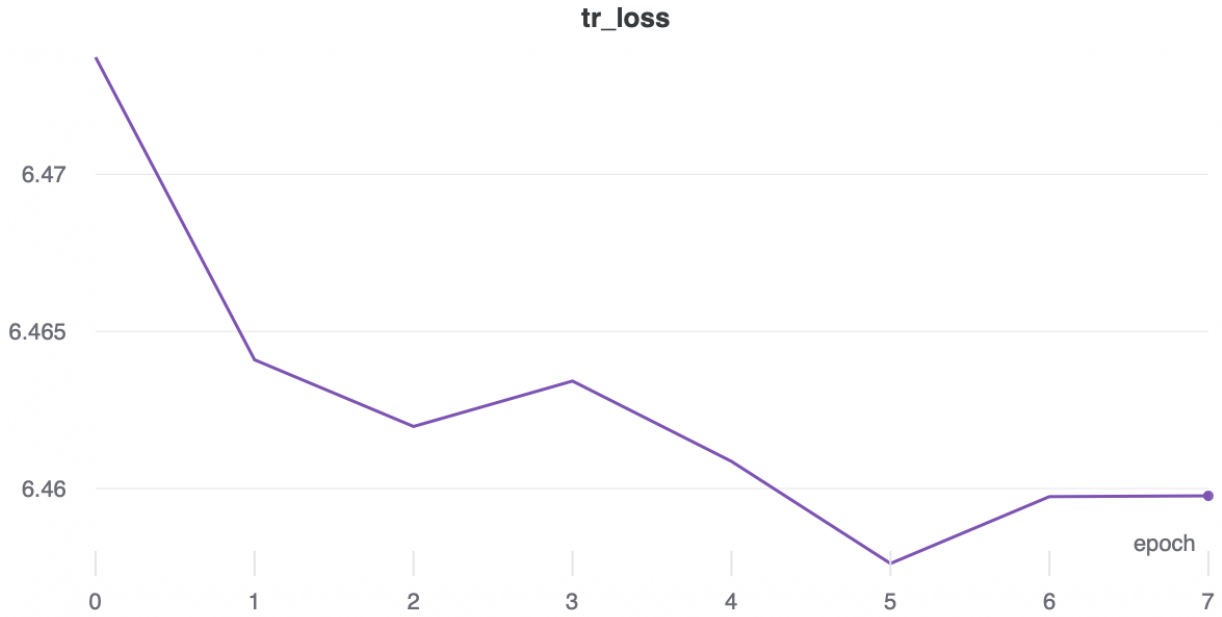
3 Results and Observations

3.1 Number of parameters

Table-1	
Model Name	**No of parameters**
bert-base-cased	109.4M
bert-base-german-cased	109M
abnet-iwslt14-de-en	323.7M

The table clearly shows that adapters are accounting for around 1/3 of total parameters. And major advantage of such a kind of architecture is that we need to train only 1/3 of parameters which will save 2/3 of memory. Hence bigger batches are possible which can speed-up the training process. Also while deploying these models, 2/3 of parameters will remain same for lots of tasks and we need to add only 1/3 of parameters. This can help us freeing more memory when deploying on mobile/other devices.

3.2 Training Graphs



Above figure shows training loss decreases as no of epochs are increasing. We can clearly see model loss is in decreasing trend which says that model is getting trained.

3.3 Inference

3.3.1 Predictions from trained model

Table-2		
German sentence	**Target sentence**	**Prediction**
Die Ergebnisse werden voraussichtlich gut sein	The results are expected to be good	woman ı names [unused114]
Die Ausgabe wird gut sein	The output will be good	woman ı names [unused114]

Above table shows that our predictions are quite random. We hypothesize that training more can make predictions accurate, which we couldn't because of unavailability of more compute resources. To assert whether our model is learning, we are summarizing results of untrained model in following table.

3.3.2 Predictions from untrained model

Table-3		
German sentence	**Target sentence**	**Prediction**
Die Ergebnisse werden voraussichtlich gut sein	The results are expected to be good	mixer mixersities assemble mixersities mixer mixer mixer mixer
Die Ausgabe wird gut sein	The output will be good	mixer mixersities assemble mixersities mixer mixer mixer mixer

Above table shows results in case of untrained model. Model is outputting same token again and again as expected. This is supporting section-2.3. This shows that our model is getting trained but more training is required for better results.

Note: Since model needs to predict target length also, we observed that our model was mostly giving 0 as target lengths, so we disregard the predicted target length and feed some non-zero as target length. We explained more on this in section-2.2.

4 Conclusion

Transformers can be seen everywhere in NLP and are able to achieve SOTA on several tasks. But generally, we need to train very deep models for achieving nice performance and training such a large model for each and every task can be hard. In these scenarios, these adapters makes it possible to train bigger model once and fine-tune small weights for adapting to particular task. Its hard to incorporate bi-directional models for seq2seq tasks because of unavailability of future tokens in real time. In those scenarios, mask-predict can enable us to adopt bidirectional models like BERT in decoder side.

5 Appendix

We coded the entire paper completely by ourselves except 'modeling/modeling-bert.py' and 'modeling/multihead-attention.py' which are taken from 'huggingface' and 'fairseq'. We trained model on colab-T4 gpu. Due to limited compute resources, we were unable to achieve good results comparable to paper. But still making our weights available in the following link for your reference. Link to weights: `https://huggingface.co/vasudevgupta/abnet-iwslt14-de-en`

Note: Above link is having only adapter weights. Since bert weights (bert-base-cased bert-base-german-cased) are not changed during the whole process, it can be downloaded from standard website.

We coded such that following code will automatically download all the weights from hugging-face hub and set you up for inference.

```
from modeling import TransformerMaskPredict
# following command will download weights from hub
model = TransformerMaskPredict.from_pretrained("vasudevgupta/abnet-iwslt14-de-en")
```