ResAdapters: Residual Architecture in Language Model Adapters

Chia hong HSU
Student ID:20562937
chsuae@connect.ust.hk

Chia wei WU
Student ID: 20561282
cwuan@connect.ust.hk

Abstract

Training huge language models via fine-tuning is an effective transfer-learning task in NLP. However, there still exist some issues in such a fine-tuning process. Firstly, fine-tuning is expensive. For example, fine-tuning a BERT costs approximately 110M parameters, which is a time-consuming process for regular computers. Secondly, as different downstream tasks would require separate fine-tuned language models, it is storage-wise expensive to save a complete set of parameters just for a single task. As an alternative, researchers like N. Houlsby et. al. were the pioneers of the original "adapter" concept [1]. The main idea was to freeze the parameters of pretrained models and train only the additional parameters of these adapters. Based on this idea, in this paper we introduce ResAdapters, an unprecedented approach by applying the residual architecture to adapters. We show that residual networks are cheap since it is equivalent to a pipelining architecture without adding additional parameters. Moreover, we show that adding residual networks significantly improves the effectiveness of adapters.

1 Introduction

Transfer learning from pre-trained BERT can achieve excellent performance on many NLP tasks, for instance, on question answering and text classification (J. Devlin et al. [7] 2018). In our paper, we introduce a ResAdapter placed outside the pre-trained model, which extracts twelve hidden states of transformers of BERT as root components of the adapters. These adapters will next compose a residual network and achieve good performance in extractive question answering tasks.

The proposed model does not adjust the parameters of BERT, therefore retains the original injected knowledge. We apply an extractive question answering dataset Stanford SQuAD2 on ResAdapter as an indicator to judge the performance of the model, and compare it to other different models that use the same data set.

The key design is the utilization of adapters that possess effective performance on transfer learning with the integration of residual networks. Users only need to train 7M parameters to achieve similar performance on question answering tasks compared to whole BERT fine-tuning.

2 Related Work

2.1 Adapter Architectures

Integrated Adapter Models Proposed by N. Houlsby et. al., also the first appearance of the adapter concept, Integrated Adapter Models were designed to be plug-ins inside transformer layers of language models [1]. Inside an adapter is a feedforward project-down layer that reduces the training parameters. It then passes through a project-up feedforward layer before going through a non-linear layer. This architecture resolves the previous problem of training a large number of parameters when fine-tuning language models.

Pipelined Adapter Models (Parallel) Published by R. Wang et al.[2], the purpose is to alleviate the problem of knowledge forgetting caused by directly inserting adapters in the transformer of language models. Their adapter architecture directly pipelines the hidden states from N transformer layers. Pipelined hidden states go through 2 projection layers, project down layer and project up layer. The outputs of K adapters will be connected and concatenated with the hidden state from the last transformer. It finally passes through a fully connected layer and outputs the final result. This model has achieved remarkable performances in tasks such as question classification, extractive question answering. The pipeline approach can effectively combine the information possessed in different transformers of language models and new training knowledge.

2.2 Probing Insights

The probing model proposed by T. McGuire et al. extract the hidden state of twelve transformers[8]. We use T_i to represent each hidden state, each

hidden state corresponds to the input for probes P_i. Each probe is composed of two identical feed forward networks S and E that predicts start index and end index respectively.

$$\hat{y_s} = S(T_i) = \sigma(T_i * w + b)$$
$$\hat{y_e} = E(T_i) = \sigma(T_i * w + b)$$

The prediction score is then calculated by $\hat{y_s}[j] + \hat{y_e}[k]$, with j representing the start index and k representing the end index of the predicted answer. T. McGuire et al. evaluated and the results from 12 probes, and discussed their semantic meanings from the evaluation result [8]. More will be discussed in Section 3-2.

3 Methodology

3.1 Pipelined Adapters

We start by extending the Pipelined Adapter Models because it is more convenient to manipulate the pipelined results outside the hidden transformer layers. Pipelined Adapters are trained in parallel to the original BERT. Each cell takes inputs from the original transformer and from the previous Adapter. Besides that, based on T. McGuire et. al. probing analysis [8], it is more intuitive to implement split-transform-merge residual networks according to the similar learning semantics behind each layer. More on split-transform-merge will be introduced in the following discussion about residual networks.

Inside each adapter, the architecture is as follows: a project down layer, a batch normalization layer, an attention layer, a dropout, a project up layer, and finally, layer normalization. The approximated total number of parameters trained in our adaptors is 7M, which is only 7% of the number of parameters inside the BERT network.

3.2 Residual Architecture

ResNet The basic design of residual networks is straight forward by simply adding a path from blocks of layers before. Let $\boldsymbol{H}(\vec{h})$ be the objective function that needs to be approximated by our model. Each layer inside our model can be regarded as a transformation $\boldsymbol{T_k}(\vec{h})$. According to experimental results, K. He et. al. claimed that learning from residuals is easier compared to learning from blocks of transformation. In other words, stacking transformations, $\prod_{k=0} \boldsymbol{T_k}(\vec{h})$, is more difficult to approximate some $\boldsymbol{H}(\vec{h})$ compared to the residual approach,, $\boldsymbol{F}(\vec{h}) := \boldsymbol{H}(\vec{h}) - \vec{h}$. The reason is that identity aggregation of $\vec{h}$ reduces the chance of diverging away from our approximated objective. The most widely adopted ResNet blocks are often composed by 2 - 4 stage blocks.Therefore in our research, we experiment with stage-box of the size of 3 for all of our residual architecture designs**.**

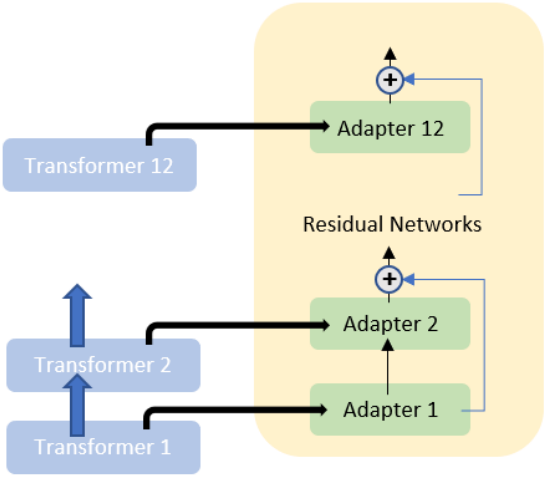

Figure 1: Pipelined Adapters are trained in parallel to the original BERT. As BERT's parameters are fixed, such architecture only backpropagates the loss and train the parameters inside these adapters. Each adapter cell takes 2 inputs, one from BERT and another from the previous adapter layer. Residual paths are added to obtain better approximation to our objective function.

ResNeXt Inspired by Google's Inception neural network, S. Xie et. al. showed that by splitting networks into higher cardinality [4], ResNeXt is a more generic form of normal ResNets. This is also known as the split-transform-merge strategy. In particular, the approximation can be represented as $\boldsymbol{H}(\vec{h}) = \vec{h} + \sum_k \boldsymbol{T_k}(\vec{h})$. The original residual network is just a special case when the total cardinality is restricted to 1. In our ResAdapter model, we simulated similar architecture by evenly distributing the 12 layers of transformers into four sectors. Each sector pipelines the hidden states from three consecutive layers of the BERT transformers. In the end, we aggregate the four sector results with a residual path from the first transformer layer. The reason for such splitting is that

we attempt to learn different semantic meanings from different layers of the BERT transformers. Discovered by T. McGuire et. al. via probing [8], the first three layers share lower-level semantics. Similarly, the last three layers capture higher semantics meaning that is dependent to the training task.

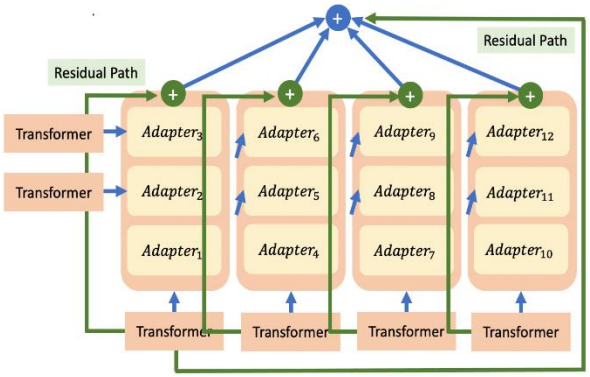

Figure 2: This is the architecture of our ResNeXt Adapter. The cardinality is set to 4, as we attempt to capture different semantic meanings from the lower, middle and higher layers. Two types of residual paths are implemented.

3.3 Approach

Extractive Question Answering Our experiments are based on extractive question answering task. It is a logistic regression task aimed to look for the start and end position from the given context. The predicted answer will be the span of the start and end index. With the end index smaller than the start index, we predict that the question is unanswerable.

Natural Language to Tokens Each of the data is composed of tokenized question and context separated by the [SEP] token. Besides, all data starts with a [CLS] token, and are padded to equal length with multiple [PAD] tokens. The following is the format of our raw data.

$$\vec{h}_0 = [CLS], [Question], [SEP], [Context], [PAD], [SEP]]$$

Firstly, raw data will be passed through BERT's embedding layer. Each token is represented by a 768 dimension vector. After data passes through each transformer layer inside BERT, it is then pipelined to the adapter cells inside our ResAdapter model. Normal adapter cells take 2 inputs, one from the transformer layer, the other from the previous adapter layer. If there is a residual path, adapters will take additional input from the residual block. The following is a detailed illustration of a cell architecture:

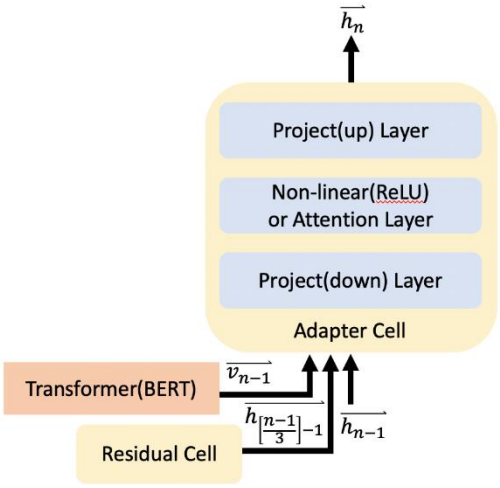

Figure 3: An Adapter Cell consists of three layers: project down layer, non-linear layer, and project up layer. We also implemented regularization layers that are not shown in the picture. Batch normalization is put after project down layer, and drop out is put after non-linear layer.

After passing through 12 adapter layers, we pass the last result into a linear layer that trains a score for each token in our hidden vector.

$$\overrightarrow{h_s} = Linear(\overrightarrow{h_{12}}),\ \overrightarrow{h_{12}} \in \mathbb{R}^{384\times768},\ \overrightarrow{h_s} \in \mathbb{R}^{384}$$

Applying nonlinear softmax, the highest score among the 384 dimension space represents the most likely starting(or ending) position for our span prediction. The loss function is then defined to be the sum of cross-entropy loss for start and end prediction.

$$Loss = \sum CrossEntropy(\overrightarrow{h_s}, \hat{h})$$

4 Experiments

4.1 Data Setup

Dataset Our model performance is evaluated in a single benchmark SQuAD2.0. As a commonly adopted MRC benchmark dataset, SQuAD2.0 consists of 100k questions in SQuAD1.1 with 50k unanswerable questions that were written by Stanford researchers based on adversarial theory. The

data set collects various topics, and designs some question answering pairs for those topics.

Our implementation is based on the BERT pre-trained model provided on Hugging Face. We set the learning rate to 3e-5 with a dropout rate of 0.3, and select the batch size of 8. The number of epochs is set to above 5 for our experiments. Contexts are tokenized using Pytorch transformer package BertTokenizer. The maximum sentence length is padded to 384. For answer verification, we follow the same setting according to the corresponding literature (J. Devlin [7] et al. 2019), which simply adopts the default answer threshold equals 1.

4.2 Evaluation Metrix Performance on SQuAD2 dataset is based on two metrics, exact match (EM) and F1 measure (F1). EM score of each answer prediction is either 1 or 0, with 1 represent the complete match between the predicted answer and ground truth answer. In other cases, the score is zero. The average EM score can be expressed by the below formula.

$$EM = \sum_{i}^{n_{samples}} \frac{1\{\hat{a_i} = a_i\}}{n_{samples}}$$

F1 represents the harmonic mean of recall and precision value.

$$F1 = 2(\frac{precision + recall}{precision \times recall})$$

4.3 Models Analysis

Table 1 shows two sets of comparison experiments, "non-linear" and "with attention", which are trained on 50K training dataset. According to the non-linear experiments, we first show that adding residual paths in adapters improves the performance significantly. Then, to further boost up the overall performance, we increase the number of training parameters by adding attention layers, and compare the results among those with and without residual paths. Our Attention ResNeXt Adaptor achieved the highest "has answers(HasAns)" f1 and exact match(EM) scores.

After showing the effectiveness of residual networks, we wanted to run an ultimate performance test on our ResNeXt adapters using all 130K training samples. We compare the results with the traditional fine-tuned BERT model in Table 2. Our ResNeXt model consists of only 7M parameters, and the BERT model has about 110M parameters.

Model	ALL		NoAns		HasAns	
	EM	**F1**	**EM**	**F1**	**EM**	**F1**
Non Linear						
Adapter	30.6	31.6	54.5	54.5	4.6	6.6
Adapter ResNet	**43.8**	**46.1**	*68.3*	*68.3*	**17.1**	**21.8**
With Attention						
Attention	33.2	40.0	*56.8*	*56.8*	7.5	21.8
Attention ResNet	45.0	49.0	53.5	53.5	35.7	43.9
Attention ResNeXt	**42.6**	**51.4**	42.8	42.8	**42.3**	**60.0**

Table 1: Performance on SQuAD2.0 dev set(trained on 50K samples). Residual significantly improved the performance of adaptors. The result shows that it is independent to the number of parameters being trained inside our adapters. Those models inside the Non Linear category consists of approximately 3M parameters, and those in "With Attention" consist of about 7M parameters. Note the generally high NoAns EM and f1 scores among all models. This indicates that adapters tend to learn and increase the score of HasAns, and ignore the side effect caused by NoAns until the scores of HasAns have converged.

Epochs/ Model	ALL		NoAns		HasAns	
	EM	**F1**	**EM**	**F1**	**EM**	**F1**
Epoch1	51.4	52.5	*85.1*	*85.1*	14.7	16.9
Epoch2	54.8	56.6	84.0	84.0	23.1	26.7
Epoch3	54.6	57.8	70.8	70.8	37.0	43.7
Epoch4	**56.3**	59.4	72.8	72.8	38.5	44.9
Epoch5	53.2	57.8	59.8	59.8	45.9	55.7
Epoch6	54.7	59.4	62.3	62.3	46.5	56.3
Epoch7	54.6	59.3	58.0	58.0	**50.9**	**60.8**
Epoch8	55.7	**59.8**	67.9	67.9	42.5	51.0
BERT Fine-tuned	**68.4**	**73.0**	**70.8**	**70.8**	**65.9**	**75.5**

Table 2: Attention ResNeXt Performance on SQuAD2.0 dev set(trained on 130K samples). Our ResNeXt consists of only 7M parameters. In contrast, BERT has 110M parameters.

5 Conclusion

Under a similar amount of parameters being trained, we find evidence to support that adding

residual networks inside adapters significantly improves the performance. The ResNeXt Adaptor even boosted the f1 score higher by separately learning different levels of semantics from different layers. Further work can be developed on adding additional attention layers inside the adaptor cells (as previous works from R. Wang et. al. implemented 2 layers of attention [2]), or other architectures that increase the size of parameters. As our adapter design is extended from Pipelined Adapters Models, it is also worth researching the effectiveness of residual networks for Integrated Adapter Models.

References

[1] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. D. Laroussilhe, A. Gesmundo, M. Attariyan, S. Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. ICML 2019.

[2] R. Wang, D. Tang, N. Duan, Z. Wei, X. Huang, J. Ji, G. Cao, D. Jiang, M. Zhou. 2020. K-Adapter: Infusing Knowledge into Pre-Trained Models with Adapters. arXiv:2002.01808.

[3] A. C. Stickland, I. Murray. 2019. BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning. ICML 2019.

[4] S. Xie, R. Girshick, P. Dollar, Z. Tu, K. He. 2017. Aggregated Residual Transformations for Deep Neural Networks. arXiv:1611.05431.

[5] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, H. Lin, Z. Zhang, Y. Sun, T. He, J. Mueller, R. Manmatha, M. Li, A. Smola. 2020. ResNeSt: Split-Attention Networks

[6] K. Lee, M. W. Chang, K. Toutanova. 2019. Latent Retrieval for Weakly Supervised Open Domain Question Answering. arXiv:1906.00300.

[7] J. Devlin, M. W. Chang, K. Lee, K. Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805

[8] T. McGuire, P. Bhardwaj. 2019. Transferability of Contextual Representations for Question Answering.