Efficient Training of BERT by Progressively Stacking

Linyuan Gong [1] Di He [1] Zhuohan Li [1] Tao Qin [2] Liwei Wang [1 3] Tie-Yan Liu [2]

Abstract

Unsupervised pre-training is commonly used in natural language processing: a deep neural network trained with proper unsupervised prediction tasks are shown to be effective in many downstream tasks. Because it is easy to create a large monolingual dataset by collecting data from the Web, we can train high-capacity models. Therefore, training efficiency becomes a critical issue even when using high-performance hardware. In this paper, we explore an efficient training method for the state-of-the-art bidirectional Transformer (BERT) model. By visualizing the self-attention distributions of different layers at different positions in a well-trained BERT model, we find that in most layers, the self-attention distribution will concentrate locally around its position and the start-of-sentence token. Motivated by this, we propose the *stacking* algorithm to transfer knowledge from a shallow model to a deep model; then we apply stacking *progressively* to accelerate BERT training. Experiments showed that the models trained by our training strategy achieve similar performance to models trained from scratch, but our algorithm is much faster.

1. Introduction

In recent years, deep neural networks have pushed the limits of many applications, including speech recognition (Hinton et al., 2012), image classification (He et al., 2016), and machine translation (Vaswani et al., 2017). The keys to the success are the advanced neural network architectures and massive databases of labeled instances (Deng et al., 2009). However, human annotations may be very costly to collect, especially in domains that require particular expertise.

In natural language processing, using unsupervised pre-trained models is one of the most effective ways to help train tasks in which labeled information is not rich enough. For example, word embedding learned from Wikipedia corpus (Mikolov et al., 2013; Pennington et al., 2014) can substantially improve the performance of sentence classification and textual similarity systems (Socher et al., 2011; Tai et al., 2015; Kalchbrenner et al., 2014). Recently, pre-trained contextual representation approaches (Devlin et al., 2018; Radford et al., 2018; Peters et al., 2018) have been developed and shown to be more effective than conventional word embedding. Different from word embedding that only extracts local semantic information of individual words, pre-trained contextual representations further learn sentence-level information by sentence-level encoders.

BERT (Devlin et al., 2018) is the current state-of-the-art pre-trained contextual representations based on a huge multi-layer Transformer encoder architecture (BERT-Base has 110M parameters and BERT-Large has 330M parameters) and trained by masked language modeling and next-sentence prediction tasks. Because these tasks require no human supervision, the size of the available training data easily scales up to billions of tokens. Therefore, the training efficiency of such a model becomes the most critical issue, and the requirement of extremely high-performance hardware becomes a barrier to its practical application.

In this paper, we aim to improve the training efficiency of the BERT model from in an algorithmic sense. Our motivation is from the observation of self-attention layers, which is the core component of the BERT model. We visualize a shallow BERT model and a deep BERT model and then study their differences and relationships. By carefully investigating the attention distributions in different layers at different positions, we find some interesting phenomena: First, the attention distributions of the shallow model are quite similar across different position and layers. At any position, the attention distribution is a mixture of two distributions. One distribution is local attention that focuses on neighbors. The other distribution focuses on the start-of-sentence token. Second, we find the attention distribution in the shallow model is similar to that of a deep model. This suggests that such knowledge can be shared from the shallow model to a

The work was done while the first and third author were visiting Microsoft Research Asia. [1]Key Laboratory of Machine Perception, MOE, School of EECS, Peking University [2]Microsoft Research [3]Center for Data Science, Peking University, Beijing Institute of Big Data Research. Correspondence to: Tao Qin <taoqin@microsoft.com>.

Proceedings of the 36^{th} *International Conference on Machine Learning*, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

deep model: Once we have a shallow model, we can stack the shallow model into a deep model by sharing weight between the top self-attention layers and the bottom self-attention layers, and then fine-tune all the parameters. As we can train the model from a shallow one to a deep one, training time can be largely reduced as training a shallow model usually requires less time.

We conduct extensive experiments on our proposed method to see (1) whether it can improve the training efficiency and convergence rate at the pre-training step, and (2) whether the trained model can achieve similar performance compared to the baseline models. According to our results, we find first during pre-training, our proposed method is about 25% faster than several baselines to achieve the same validation accuracy. Second, our final model is competitive and even better than the baseline model on several downstream tasks.

2. Related Work

2.1. Unsupervised Pre-training in Natural Language Processing

Pre-trained word vectors (Mikolov et al., 2013; Pennington et al., 2014) have been considered a standard component of most state-of-the-art NLP architectures, especially for those tasks where the amount of labeled data is not large enough (Socher et al., 2011; Tai et al., 2015; Kalchbrenner et al., 2014). However, these learned word vectors only capture the semantics of a single word independent of its surrounding context. The rich syntactic and semantic structures of sentences are not effectively exploited.

Pre-trained contextual representations overcomes the shortcomings of traditional word vectors by considering its surrounding context. Peters et al. (2018) first train language models using stacked LSTMs, and then use the hidden states in the stacked LSTMs as the contextual representation. Since LSTM processes word sequentially, the hidden state of LSTM at one position contains the information of the words in previous positions, and thus the representation contains not only the word semantics but also the sentence contexts. Radford et al. (2018) uses advanced self-attention units instead of LSTM units in language models. Devlin et al. (2018) further develops a masked language modeling task and achieves state-of-the-art performance on multiple natural language understanding tasks. As (masked) language modeling requires no human labeling effort, billions of sentences on the web can be used to train a very deep network. Therefore, a major challenge in learning such a model is training efficiency.

2.2. Network Training by Knowledge Transfer

Our iterative training method is also closely related to efficiently training deep neural networks using knowledge transfer. Chen et al. (2015) tackles the problem about how to train a deep neural network efficiently when we have a shallow neural network. In particular, *function-preserving initialization* is proposed which first initializes a deep neural network that represents the same function as the shallow one, and then continue to train the deep network by standard optimization methods. However, when dealing with sophisticated structures such as Transformer, function-preserving initialization is usually not effective. For example, the basic component in the Transformer is a composition of a self-attention layer and a feed-forward layer. According to our empirical study, simply setting the feed-forward layer to be near zero and randomly initializing the self-attention layer is a function-preserving initialization, but it is ineffective as most parameters in the self-attention layer stay untrained. In our work, we propose a different and more efficient method to transfer knowledge from shallow models to deep models.

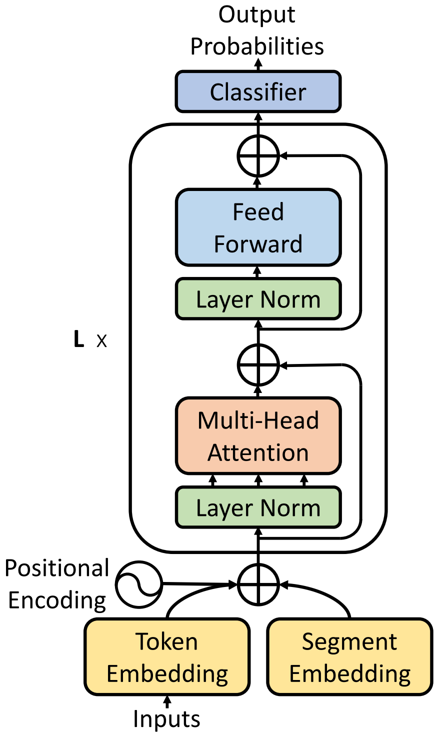

Figure 1. The model architecture of BERT.

3. Method

The BERT (**B**idirectional **E**ncoder **R**epresentation from **T**ransformers) model is developed on a multi-layer bidirectional Transformer (Vaswani et al., 2017) encoder. The architecture is shown in Figure 1. The encoder consists of L encoder layers, each of which consists of a *multi-head self-attention* sub-layer and a feed forward sub-layer: both of them have residual connections (He et al., 2015). The feed forward layer (FFN) is *point-wise*, i.e., it applies independently to each position of the input.

The key component of the Transformer encoder is the *multi-head self-attention* layer. An *attention* function can be formulated as querying a dictionary with key-value pairs (Vaswani et al., 2017), e.g.,

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V, \quad \text{where } Q \in \mathbb{R}^{n_q, d_k}, K \in \mathbb{R}^{n_e, d_k}, V \in \mathbb{R}^{n_e, d_v}. \tag{1}$$

d_k is the dimension of each *key* and each *query*, n_q is the number of queries, and n_e is the number of key-value entries. $A = \text{softmax}(QK^T/\sqrt{d_k}) \in \mathbb{R}^{n_q, n_e}$ defines the attention distribution. The output of each query is a weighted average of the rows of V with A as the coefficient. The attention distribution A helps us understand the attention function: A_{ij} reflects the importance of the i-th key-value entry with respect to the j-th query in generating the output (Bahdanau et al., 2014).

Multi-head self-attention layer is a attention function with multiple parameterized *heads*:

$$\begin{aligned} \text{MultiHeadAtt}(X) &= \text{Concat}(h_1, h_2, \ldots, h_H)W^O, \\ h_i &= \text{Attention}(XW_i^Q, XW_i^K, XW_i^V), \\ \text{where } W_i^Q, W_i^K, W_i^V &\in \mathbb{R}^{d, d/H}, \; W^O \in \mathbb{R}^{d,d}. \end{aligned} \tag{2}$$

Here H is the number of heads; $X \in \mathbb{R}^{n,d}$, a sequence of n d-dimension hidden states, is the input to this layer, and W^Q, W^K, W^V, W^O are trainable parameters. W_i^Q and W_i^K together define the similarity metric for hidden states in X to compute the attention distribution of each head, so there are H attention distributions in each layer.

To investigate the mechanism behind BERT, we visualize the attention distribution of each layer at each position. Implementation details of our experiments can be found in Section 4.1. We trained a model using the BERT-Base setting which is a 12-layer model with stacked self-attention layers. We randomly picked one sentence from the validation set of the corpus, and visualize the attention distributions of heads from different layers in Figure 2. More visualizations are included in the supplementary materials. From the figure, we found that:

1. The attention distribution is quite singular. The attention distributions of most heads are mixtures of two distributions: one distribution focuses on local positions, and one distribution focuses on the first token (which is the *class* (CLS) token). This suggests that neighboring positions and global information are both important for understanding a single token.
2. The attention distributions of many heads from top layers (e.g., layer 8, 10, 12) look very similar to the attention distributions of heads from bottom layers (e.g., layer 2, 4, 6).

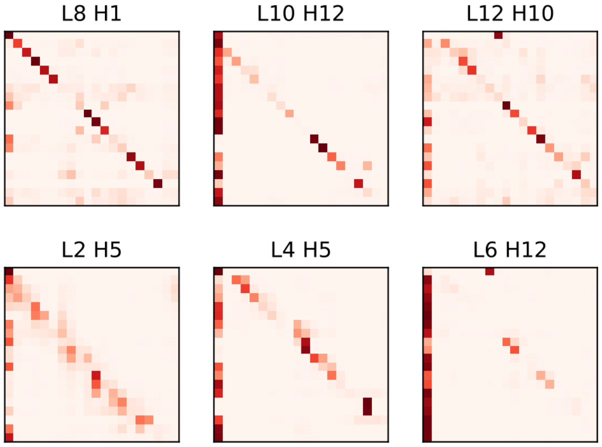

Figure 2. Visualization of attention distributions of BERT-Base. For a randomly chosen sample sentence, we visualize the attention distributions of 6 heads from different layers. For example, "L8 H1" denotes the first head of the eighth layer. In each heatmap, the color depth of the j-th element in the i-th row reflects the attention weight from position i to position j: the darker the color, the more attention position i pays to position j.

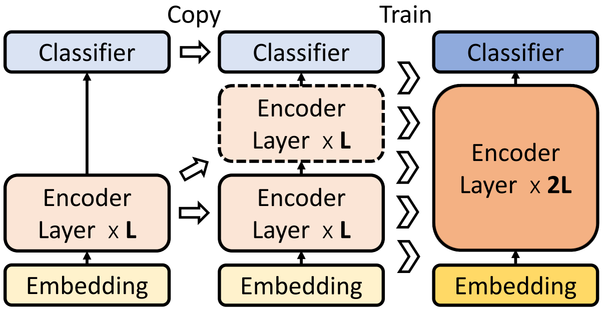

Figure 3. The diagram of the *stacking* algorithm.

The attention distributions of bottom layers are similar to the attention distributions of top layers. This fact suggests that to a certain extent, their functionalities are similar. In addition, the success of *Universal Transformers* (Dehghani et al., 2018) in sequence-to-sequence modeling tasks also suggests that it is possible for the Transformer at different depths to share the same parameters. Therefore, we think that it is also possible if we *copy* the parameters of a trained L-layer BERT, *stack* it into a $2L$-layer model.

Stacking. As a result, we designed the *stacking* algorithm. As is shown in Figure 3, if we have a L-layer trained BERT, we can construct a $2L$-layer BERT by copying its parameters: for $i \leq L$, the i-th layer and the $(i + L)$-th layer of the constructed BERT have the same parameter of the i-th layer of the trained BERT. By warm-starting with knowledge transferred from the L-layer trained BERT, we expect our model to learn faster than by training from scratch.

By stacking, the parameters of a trained L-layer BERT should be a good warm-start for self-attention layers of the top L layers. For example, as is mentioned above, an attention head of each position usually focuses on the *CLS* token and its neighbors. If we initialize top L layers with these parameters, these attention heads will still focus on the *CLS* position to extract global information as well as their neighbors to extract local information.

Progressive stacking. Because shallow models can usually be trained faster than deep models (for the same number of steps), the training time will be greatly reduced if we train deep models by stacking from a shallow one. Again, we can train this shallow model faster by stacking from a shallower model. By recursion, we design an iterative training algorithm based on our stacking technique to train a deep BERT faster. We call this algorithm *progressive stacking* (Algorithm 1).

Algorithm 1 Progressive stacking

```
$M'_0 \leftarrow$ InitBERT$(L/2^k)$
$M_0 \leftarrow$ Train$(M'_0)$ {Train from scratch.}
for $i \leftarrow 1$ to $k$ do
    $M'_i \leftarrow$ Stack$(M_i)$ {Doubles the number of layers.}
    $M_i \leftarrow$ Train$(M'_i)$ {$M_i$ has $L/2^{k-i}$ layers.}
end for
return $M_k$
```

4. Experiments

4.1. Experimental Design

All of our experiments are mainly based on our own reimplementation of BERT model (Devlin et al., 2018) using *fairseq* (Gehring et al., 2017) in PyTorch toolkit [1]. We set most of the hyperparameters to be the same as the original BERT (Devlin et al., 2018). The detailed hyperparameter setting is attached in the supplementary material.[2]

Datasets. We follow Devlin et al. (2018) to use English Wikipedia corpus[3] and BookCorpus[4] for pre-training. By concatenating the two datasets, we obtain our corpus with roughly 3400M words in total, which is comparable with the data corpus used in Devlin et al. (2018). We first segment documents into sentences with Spacy[5]; Then, we normalize, lower-case, and tokenize texts using Moses decoder (Koehn et al., 2007); Next, we apply *byte pair encoding* (BPE) (Sennrich et al., 2015). We randomly split documents into one training set and one validation set. The training-validation ratio for pre-training is 199:1.

pWe fine-tune each pre-trained model on 9 downstream tasks in GLUE (**G**eneral **L**anguage **U**nderstanding **E**valuation), a system for "evaluating and analyzing the performance of models across a diverse set of existing NLU tasks" (Wang et al., 2018). We briefly describe the tasks in Table 1. We follow Devlin et al. (2018) to skip WNLI in our experiments, because few submissions on the leaderboard[6] do better than predicting the majority class for this task.

Implementation details. Using our pre-processed data described previously, we train a 12-layer BERT-base model as the baseline model. The BERT-base model is trained for 400,000 updates from scratch, and the batch size for each update is set to be 122,880 tokens.

To show how our progressively stacking algorithm can speed up training, we first train a 3-layer BERT for 50,000 steps, stack it twice into a 6-layer BERT and then train this 6-layer BERT for 70,000 steps. In the final step, we stack the 6-layer BERT into a 12-layer BERT, and train the 12-layer BERT for 280,000 steps. Therefore, eventually, we use the same architecture and number of training steps as the BERT-base model.

For both models, we use Adam (Kingma & Ba, 2014) as the optimizer, and for our progressively stacking method, we reset the optimizer states (the first/second moment estimation of Adam) but keep the same learning rate when switching from the shallow model to the deep model.

We also compare our method with other iterative stacking methods popularly used for convolutional networks in computer vision tasks:

- **Identity.** As is shown in Figure 1, each encoder layer of our BERT implementation consists of two residual blocks, and the last steps of both residual blocks are linear transformations (the second linear transformation of a feed forward layer and W^O of a multi-head self-attention layer). Therefore, if we initialize these linear transformations to be zero for the top L layers, these layers add nothing to the hidden states, so the top L layers are strictly equivalent to an identity mapping. We initialize the bottom L layers with parameters of

[1] Since the authors of BERT do not publish their code for data pre-processing or multi-GPU training, we choose other platforms for ease of our implementation.

[2] Codes for the experiments are available at https://github.com/gonglinyuan/StackingBERT

[3] https://dumps.wikimedia.org/enwiki

[4] As the dataset BookCorpus (Zhu et al., 2015) is no longer freely distributed. We follow the suggestions from Devlin et al. (2018) to crawl smashwords.com and collect BookCorpus on our own.

[5] https://spacy.io

[6] https://gluebenchmark.com/leaderboard

Table 1. GLUE task descriptions and statistics. The second column denotes the number of training examples. The fourth column denotes the number of classes; STS-B is an exception because it is an regression task.

Corpus	Size	Task	$\|C\|$	Metric(s)	Domain
Single-Sentence Classification					
CoLA	8.5k	acceptibility	2	Matthews correlation	misc.
SST-2	67k	sentiment	2	accuracy	movie reviews
Sentence Similarity/Paraphrase					
MRPC	3.7k	paraphrase	2	accuracy/F1	news
STS-B	5.7k	similarity	-	Pearson/Spearman corr.	misc.
QQP	364k	similarity	2	accuracy/F1	social QA questions
Natural Language Inference (NLI)					
MNLI	393k	NLI	3	matched/mismatched acc.	misc.
QNLI	108k	QA/NLI	2	accuracy	Wikipedia
RTE	2.5	NLI	2	accuracy	misc.
WNLI	634	coreference/NLI	2	accuracy	fiction books

a trained L layer BERT, initialize the last transformations of each residual block on the top L layers as zeros, and initialize all other layers randomly. This construction is strictly *function-preserving*, as is required by *Net2Net* (Chen et al., 2015).

- **Identity with noise.** Although it is optional to add noise when using *Net2Net*, adding a small amount of noise usually aids in breaking the symmetry more rapidly (Chen et al., 2015). In our "identity with noise" setting, everything else is identical to the "identity" setting, but we add zero-mean Gaussian noises with $\sigma = 0.03$ to the layers that are previously initialized to be zero.

For these methods, we use the same set of hyperparameters and random seeds as stacking. To fairly compare the speed of different algorithms, we train all models in the same computation environment with 4 NVIDIA Tesla P40 GPUs.

When fine-tuning models on downstream tasks, we use the same hyperparameter search space as BERT for each downstream task. We perform a hyperparameter search on the validation set of each task with our baseline model and apply the resulting hyperparameter to other models. We use a new set of random seeds that is different from the seeds for hyperparameter search to prevent over-fitting.

4.2. Experiment Results

In this subsection, we provide experiment results of the baseline model and the models trained by our proposed methods.

By visualizing attention distributions of stacked checkpoints in Figure 4, we want to justify our assumptions made in Section 3 that stacking preserves some functionalities of original attention layers. From the first and the second row of the figure, we find that the attention distributions after stacking at top layers are similar to those before stacking and confirm this assumption. Moreover, the third row of this figure shows that after we train the stacked model, some heads (e.g., the rightmost head) regain their original functionalities that are lost during stacking, and some heads "evolve" new attention patterns (e.g., the leftmost head). As a result, most heads learn meaningful information in a relatively short period of time. In contrast, as is shown in the last 2 rows of this figure, the heads in models trained by *identity* and *identity+noise* fail to learn such information.

We then compare *progressive stacking* with our baseline on the efficiency in the pre-training step. For each method at different time steps, we record the validation loss of the model and plot them in Figure 5. From the figure, we make the following observations:

- Compared with the 12-layer BERT-base baseline model (red line), progressive stacking also reaches similar pre-training validation loss at the end of training. However, the training time of our proposed method is about 25% shorter (338 hours vs. 441 hours). This is mainly because for the same number of steps, training a small model needs less computation.

- Compared with other progressively training algorithms (*identity* and *identity+noise*), progressive stacking can achieve much lower loss which leads to better accuracy. The algorithms based on function-preserving initialization do not provide a good initialization for attention layers on the top, so they converge slowly and fail to reduce training time.

From the results above, we see that our proposed method substantially reduces pre-training time. However, one may

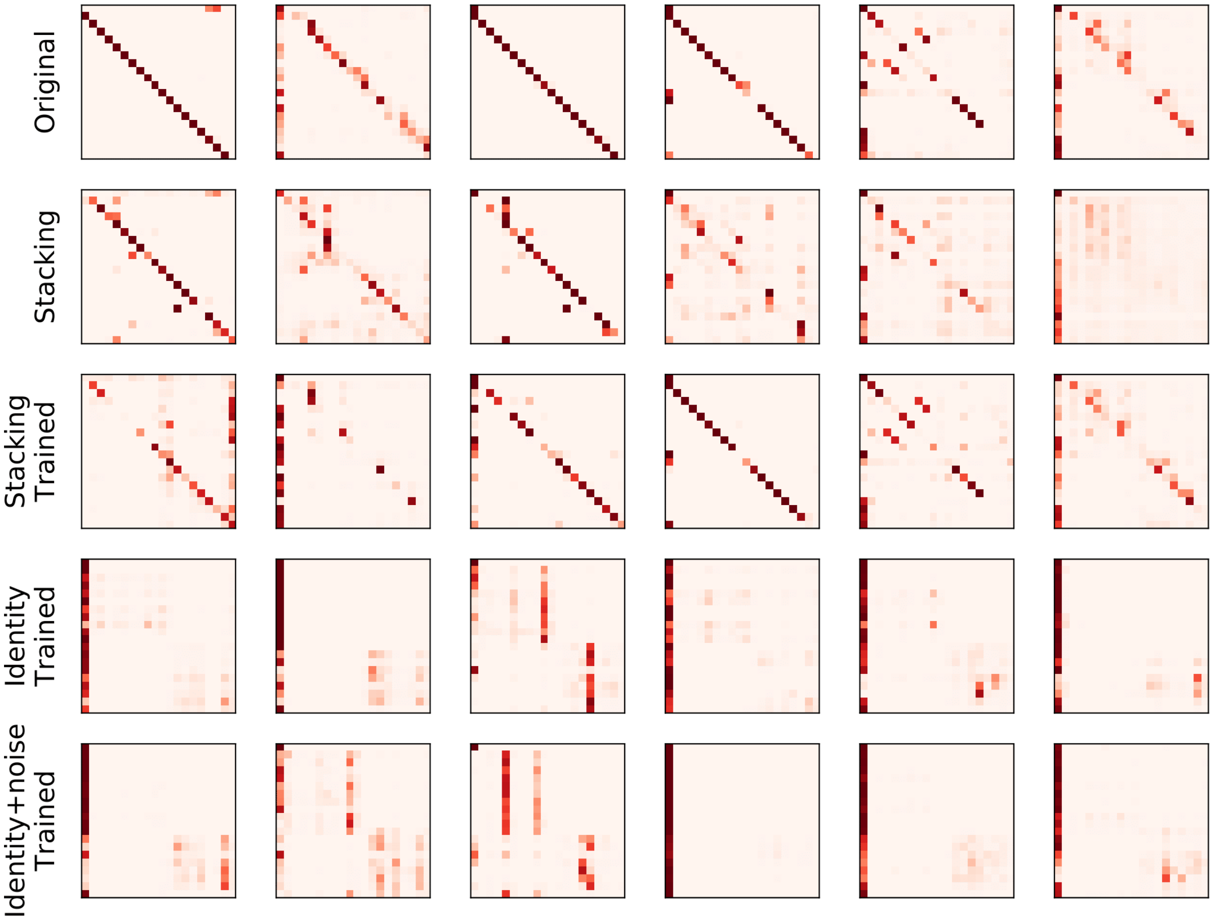

Figure 4. The first row shows the attention distributions for a randomly chosen sample sentence on random 6 heads of a trained 6-layer model. The second row shows the attention distributions of the corresponding heads with the same parameters in the top 6 layers when we stack the 6-layer model to a 12-layer model. The third row shows the attention distributions of these stacked heads after being trained for another 70,000 steps. We also show the attention distribution of models trained by *identity* and *identity+noise* under the same setting (the last 2 rows).

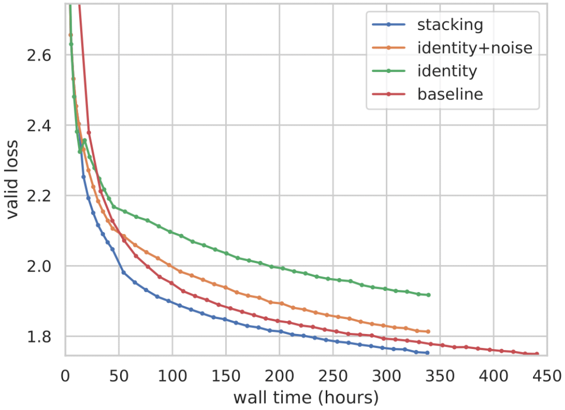

Figure 5. The validation loss curve of our baseline and three knowledge transfer methods. The x-axis is the wall time of training (excluded time for validation). Our stacking method only takes 76.77% training time to reach the same validation loss as our baseline.

still worry about whether such a model is as effective as the 12-layer BERT baseline model on downstream tasks. In the next experiment, we compare the performances of the model trained with our proposed stacking method and that of the 12-layer BERT baseline on these downstream tasks.

We dump models from different checkpoints during pre-training and fine-tune these models on downstream tasks. In this experiment, we use the validation sets of the downstream tasks for evaluation. We find that the validation scores are highly sensitive to random seeds, especially on small datasets, and thus we repeat fine-tuning on each downstream task for 3 times for larger datasets and 6 times for smaller datasets, respectively. We assume that the scores follow normal distribution and compute the 95% confidence interval of validation scores. Due to space limitations, we plot the validation curve for six out of the nine downstream tasks in Figure 6. From the figure, it is easy to see that our stacking algorithm not only trains BERT faster in terms

Table 2. The test results on the GLUE benchmark (except WNLI). The number below each task denotes the number of training examples. The metrics for these tasks can be found in the GLUE paper (Wang et al., 2018). For tasks with multiple metrics, the metrics are arithmetically averaged to compute the GLUE score. Our baseline is comparable with the original BERT-Base, and our stacking method is slightly better than our baseline. We also compare BERT-Base models with state-of-the-art pre-BERT models on the GLUE leaderboard: OpenAI GPT (Radford et al., 2018) on STILTS and ELMo (Peters et al., 2018) (GLUE Baselines).

	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m/mm	QNLI	RTE	GLUE
	8.5k	67k	3.7k	5.7k	364k	393k	108k	2.5k	
ELMo-BiLSTM-Attn	33.6	90.4	84.4/78.0	74.2/72.3	63.1/84.3	74.1/74.5	79.8	58.9	70.0
OpenAI GPT	47.2	93.1	87.7/83.7	85.3/84.8	70.1/88.1	80.7/80.6	87.2	**69.1**	76.9
BERT-Base (original)	52.1	93.5	**88.9/84.8**	**87.1/85.8**	**71.2/89.2**	**84.6**/83.4	**90.5**	66.4	78.3
BERT-Base (baseline)	52.8	92.8	87.3/83.0	81.2/80.0	70.2/88.4	84.4/83.7	90.4	64.9	77.4
BERT-Base (stacking)	**56.2**	**93.9**	88.2/83.9	84.2/82.5	70.4/88.7	84.4/**84.2**	90.1	67.0	**78.4**

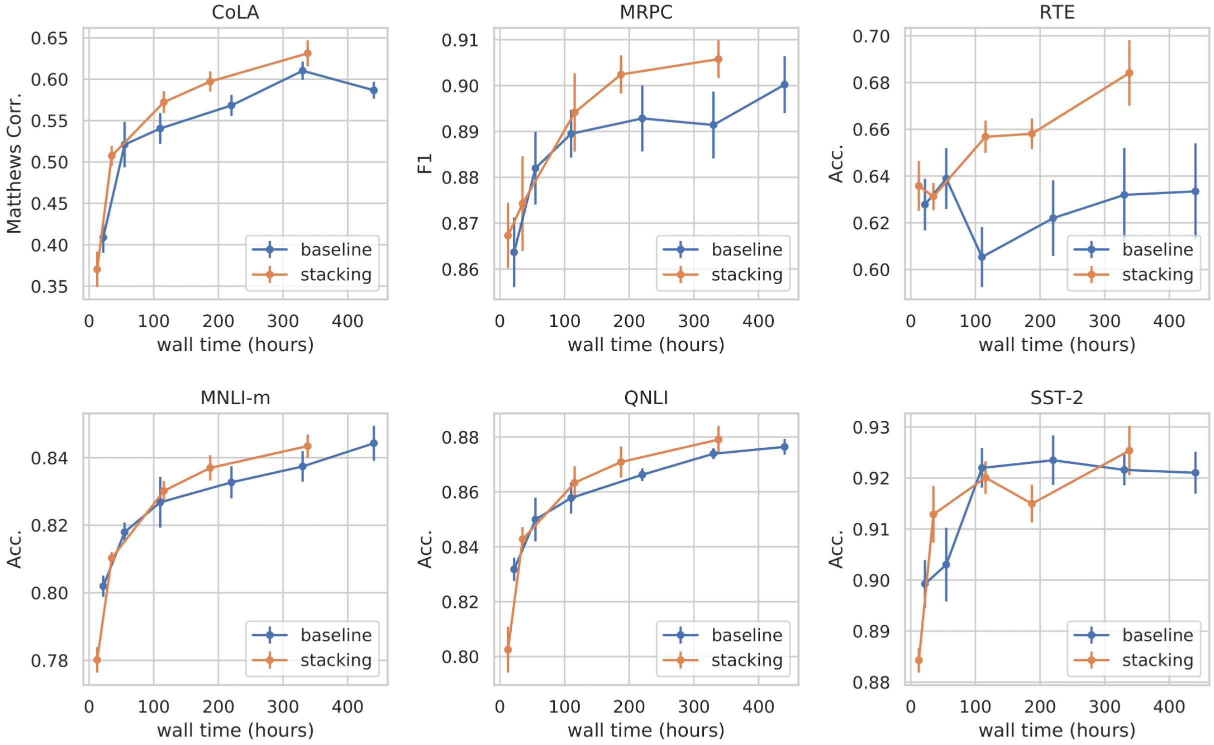

Figure 6. The validation scores on 6 downstream tasks of fine-tunes checkpoints. The x-axis is the wall time of BERT pre-training, after which we take these checkpoints. For CoLA, MRPC, RTE, and SST-2, we repeat fine-tuning 6 times; for MNLI and QNLI, we repeat fine-tuning 3 times. Every error bar reflects the confidence interval of each validation score. For most tasks, our stacking method takes shorter pre-training time to get a high score after fine-tuning.

of the pre-training validation loss but also preserves the speedup with respect to performance on downstream tasks.

Lastly, we submit the predictions of the model trained using our stacking algorithm and 12-layer BERT baseline on the online test evaluation system of GLUE. The result is listed in Table 2, from which we can see that:

- In most tasks, our 12-layer BERT baseline model achieves comparable performance to the original BERT-Base (Devlin et al., 2018). This fact confirms the validity of our proposed algorithm.
- Although our model is trained for a shorter time, it keeps competitive performance comparing to the baselines on the GLUE test set. Furthermore, it is interesting to see that our model achieves much higher performance on the CoLA task (56.2 v.s. 52.1). This suggests that the model trained by our stacking algorithm generalizes slightly better on downstream tasks than our baseline does. It is surprising because they are trained for the same number of steps, but a large

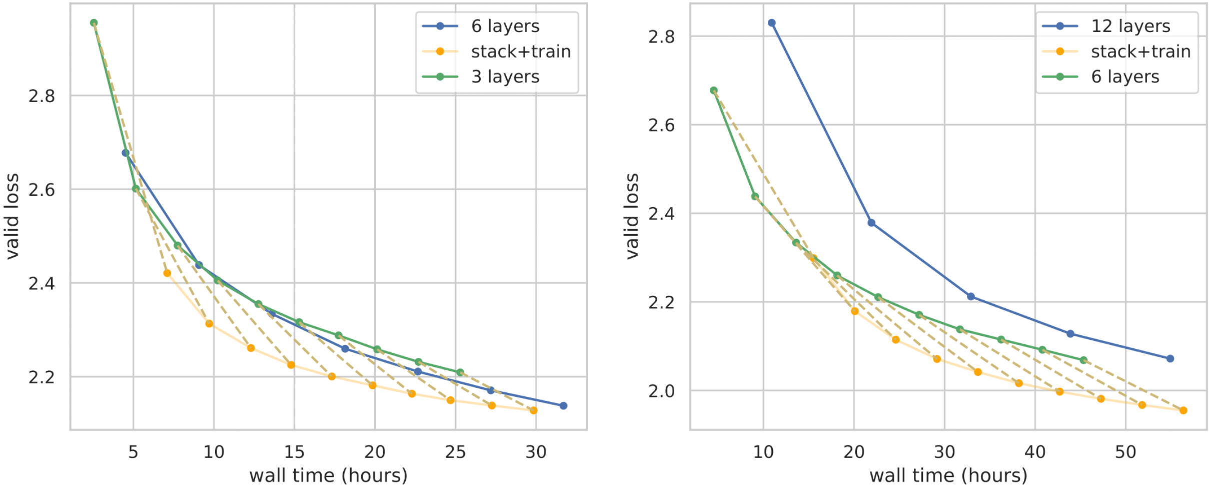

Figure 7. The validation loss curve of ablation experiments. The figure on the left shows the result for 3-to-6 stacking, and the figure on the right shows the result for 6-to-12 stacking. Each checkpoint of the shallow model is taken at a green dot; after stacking, it is trained for another 10,000 steps and is evaluated at the orange dot connected by a dashed edge. By comparing the orange line with the blue line, we can see how much the *switching time* can affect the speedup of our stacking algorithm.

percent of training steps are performed on a shallower network in our model.

In summary, according to multiple experiments, our *progressive stacking* algorithm is faster and achieves competitive performance comparing to the baseline models on validation and test sets of various tasks.

4.3. Ablation Study

In the experiments on our progressive stacking algorithm, we select the *switching time* (i.e., the time to switch to the deep model) at 50,000 steps and 120,000 steps. It is necessary to analyze how sensitive our model is to this hyperparameter.

Therefore, we conduct the following experiment for ablation study: We first train a 3-layer BERT for 100,000 steps and save one checkpoint after every 10,000 steps. Then we stack every checkpoint and train the resulting 6-layer BERT models for another 10,000 steps (we observe that after the stacked model is trained for a small number of steps, the training curve will be similar to a similar model trained from scratch, so 10,000 steps is enough to see the trend). Finally, we compare the results with the 6-layer BERT model trained from scratch. We also repeated the same experiment for 6-to-12 stacking. The results of both experiments are shown in Figure 7, from which we can see that:

- For a BERT model, there exists a threshold θ, such that if we pick the *switching time* $t < \theta$, our progressive stacking algorithm will train this model faster than training from scratch.

- If we pick $t > \theta$, we will not get a significant speedup. This is reasonable because if we train long enough (for example, until both models converge), the loss of the deeper model will be smaller than the loss of the shallower model. In this case, we can by no means speed up training by continuing to train this shallower model.

- This threshold is usually larger for deeper models. Hence, deeper models can benefit more from our progressive stacking algorithm.

5. Conclusion and Future Work

In this paper, we study the efficient training algorithms of the unsupervised pre-training model BERT for natural language processing tasks. By visualizing the self-attention distributions of different layers in a well-trained BERT model, we find that the self-attention distribution usually concentrate locally around its position and the start-of-sentence token. Motivated by this, we propose progressively training the BERT model from a shallow one to a deep one by our stacking technique. According to our experiment results, our training strategy achieves competitive performance to training a deep model from scratch at a faster rate.

We will explore new directions in the future. First, we will study other applications that use self-attention layers such as machine translation and text summarization to test the effectiveness of our training algorithm. Moreover, we are also interested in more flexible approaches to reuse trained parameters that can speed up the training of large models.

Acknowledgements

This work is supported by National Basic Research Program of China (973 Program) (grant no. 2015CB352502), NSFC (61573026) and BJNSF (L172037) and a grant from Microsoft Research Asia. We would like to thank the anonymous reviewers for their valuable comments on our paper.

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL `http://arxiv.org/abs/1409.0473`.

Chen, T., Goodfellow, I., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. Convolutional Sequence to Sequence Learning. In *Proc. of ICML*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. Moses: Open source toolkit for statistical machine translation. In *ACL*, 2007.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Pennington, J., Socher, R., and Manning, C. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_paper. pdf*, 2018.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2015. URL `http://arxiv.org/abs/1508.07909`.

Socher, R., Huang, E. H., Pennin, J., Manning, C. D., and Ng, A. Y. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems*, pp. 801–809, 2011.

Tai, K. S., Socher, R., and Manning, C. D. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018. URL `http://arxiv.org/abs/1804.07461`.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *arXiv preprint arXiv:1506.06724*, 2015.