BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning

Asa Cooper Stickland [1]   **Iain Murray** [1]

Abstract

Multi-task learning allows the sharing of useful information between multiple related tasks. In natural language processing several recent approaches have successfully leveraged unsupervised pre-training on large amounts of data to perform well on various tasks, such as those in the GLUE benchmark (Wang et al., 2018a). These results are based on fine-tuning on each task separately. We explore the multi-task learning setting for the recent BERT (Devlin et al., 2018) model on the GLUE benchmark, and how to best add task-specific parameters to a pre-trained BERT network, with a high degree of parameter sharing between tasks. We introduce new adaptation modules, PALs or 'projected attention layers', which use a low-dimensional multi-head attention mechanism, based on the idea that it is important to include layers with inductive biases useful for the input domain. By using PALs in parallel with BERT layers, we match the performance of fine-tuned BERT on the GLUE benchmark with ≈7 times fewer parameters, and obtain state-of-the-art results on the Recognizing Textual Entailment dataset.

1. Introduction

Much machine learning research is directed at improving the performance of algorithms on single tasks, however methods that can tackle many problems within a single model are less well explored. Humans, of course, can easily perform large numbers of tasks, and often transfer knowledge between related tasks. This work concentrates on adapting deep neural networks pre-trained on large amounts of English text for multi-task learning on several natural language understanding (NLU) tasks.

Some multi-task learning approaches consider learning a general-purpose model that shares all parameters across tasks (e.g., the NLP decathlon introduced by McCann et al., 2018). This setting requires all tasks to have the same input and output space, and the input to indicate the task. Instead, we consider the setting where we share most parameters across all tasks, but have a small number of task-specific parameters which adapt the shared model.

Sharing parameters, and thus a common representation, between tasks can sometimes lead to better generalization. However, fine-tuning separate models for each task often works better in practice. Although we are of course interested in multi-task methods that can give results close to or better than state-of-the-art, there are separate motivations for maintaining shared parameters between tasks:

- On applications like mobile devices we may have constraints on battery life. We can incur energy costs by applying several different neural networks to the same input. If only the 'tops' of our models are task-specific, we can apply a shared transformation only once to the input, and use this transformed representation many times, as input to each task-specific function.

- Again on mobile devices, running several different neural networks for various tasks can incur a computational and energy overhead due to swapping parameters on a dedicated integrated circuit (Rebuffi et al., 2018).

- If we have an application with a large number of tasks we may have constraints on the number of parameters we can store for all our models. For example, with web-scale applications we may wish to avoid storing a separate large model for every user.

Given a large number of shared parameters in a 'base' model, and a small number of task-specific parameters, our key questions are: where should we be transforming the base model? What form should these transformations take? We assume the task is always known, so the model can always choose the correct adaptation parameters and output space.

We explore different ways of adding parameters to deep architectures that are pre-trained on large amounts of text with some auxiliary task (such as language modeling). The

[1]School of Informatics, University of Edinburgh. Correspondence to: Asa Cooper Stickland <a.cooper.stickland@ed.ac.uk>.

availability of huge corpora, especially in English, enables the use of pre-trained, general purpose architectures to be fine-tuned on diverse tasks, such as question-answering, named entity recognition, and many NLU tasks (Devlin et al., 2018). We experiment on a set of eight NLU tasks from the GLUE benchmark (Wang et al., 2018a). These tasks are chosen to test a single NLU model across multiple tasks, including question answering, sentiment analysis, and textual entailment. The number of training examples varies widely across the tasks, and we find it necessary to schedule training carefully so not to unduly favor the well-resourced tasks, or overfit to the low-resource tasks.

We use the recent BERT architecture (Devlin et al., 2018) as our base pre-trained model. BERT stands for Bidirectional Encoder Representations from Transformers. Pre-trained BERT representations can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, including the GLUE benchmark. However, the entire model is fine-tuned, meaning we need a separate model for each task. The transformer architecture that BERT is based on is a powerful and popular one, and finding the best way to adapt the parameters of this architecture for multi-task learning may be useful in other contexts where the transformer is a good choice of architecture, such as multilingual machine translation.

Previous work (Rebuffi et al., 2018) introduced 'residual adapter modules' which were used to parameterize the standard residual network architecture of He et al. (2016), for the purpose of multi-task learning for several computer vision tasks. The base ResNet architecture was pre-trained on ImageNet. The adapters contained a small fraction of the model parameters (less than 10% for each task) enabling a high-degree of parameter sharing between domains.

Our main contributions are: 1) We introduce the 'Projected Attention Layer' (PAL), which consists of a low-dimensional multi-head attention added in parallel to normal BERT layers. 2) We introduce a novel method of sampling which task to train on, where we sample tasks proportional to their training set size at first, and de-emphasize training set size as training proceeds. 3) We perform an empirical comparison of alternative adaptation modules for self-attention based architectures, and analyse the trade-offs for adaptation modules caused by potential constraints on number of task-specific parameters and number of shared operations. Making links to the vision literature, we identify general lessons for where to add parameters to adapt models. For the multi-task learning setting of the GLUE benchmark, we show that PALs enable comparable performance to fine-tuned BERT-base (BERT-base is the smaller of the two model configurations considered by (Devlin et al., 2018)) on many tasks with ≈7 times fewer parameters. We improve the performance of BERT-base on the recognising textual

entailment (RTE) task, achieving 76.6% accuracy, surpassing the performance of fine-tuned BERT-large (70.1%) and the recent 'MT-DNN' model (75.5%) (Liu et al., 2019). We also find the more parameter sharing we have, the better we do on the RTE task.

2. Background

2.1. Multi-task Learning

Multi-task learning aims to provide an inductive bias that means models have to learn features that are general enough to perform well on many tasks (Caruana, 1997). In NLP, examples of previous work include using a single model for chunking, tagging, named entity recognition, and semantic role labeling by applying a shared neural network to text, with different output layers (Collobert et al., 2011). Another approach outputs predictions at different layers using the idea of a linguistic hierarchy (Hashimoto et al., 2017; Sanh et al., 2018). Subramanian et al. (2018) train a sequence-to-sequence RNN model on tasks including machine translation and natural language inference, and learn sentence representations useful for downstream tasks. Outside NLP multi-task learning has been applied to diverse domains such as speech recognition (Deng et al., 2013) and reinforcement learning (Teh et al., 2017). Ruder (2017) provides a more general overview.

Many multi-task learning approaches can be categorized as either 'hard parameter sharing' or 'soft parameter sharing'. Hard parameter sharing generally means sharing the hidden layers between all tasks, and having several task-specific output layers. Soft parameter sharing refers to the approach where each task has its own model, but the distances between the parameters of the models are regularized to encourage the parameters to be similar. For example Duong et al. (2015) use the L2 distance, and Yang & Hospedales (2017) use the trace norm. In our case we are assuming soft-parameter sharing with the whole of BERT requires too many parameters. We are instead interested in finding the best ways of doing hard-parameter sharing, and not restricting ourselves to only having differing output layers.

2.2. Adaptation Parameters

Various strategies for adding adaptation parameters have been explored. Learning hidden unit contributions (LHUC, Swietojanski & Renals, 2014) is a method where we modify a neural network by multiplying each hidden unit by a learnable scalar. Since we only need d parameters per layer, where d is the dimension of the hidden vector, this approach requires a small number of parameters in comparison to other methods we consider.

The 'residual adapter modules' introduced by Rebuffi et al. (2018) for multi-task computer vision take the form of a

1×1 filter bank with a skip connection. This amounts to an additional $C \times C$ matrix per layer for each task, reshaped to C 1×1 convolutional filters, with C the number of channels. This $C \times C$ matrix can be compressed by replacing it with a low-rank approximation. Several of our methods were inspired by the idea of using a low-rank approximation to the key operation of a model (which is the convolutional layer when dealing with images).

2.3. GLUE Benchmark and Fine-tuning Approaches

A recent trend in transfer learning is to pre-train some model architecture on a language model (LM) objective before fine-tuning that same model for a supervised downstream task (Dai & Le, 2015; Howard & Ruder, 2018; Radford, 2018). BERT uses a similar approach, but was trained with an objective combining the prediction of words removed (or 'masked') from an input sentence, and a classification task of predicting whether two input sentences are adjacent in a corpus of text. Crucially, unlike a normal LM objective, BERT conditions on both left and right context when predicting the masked words, allowing a more flexible representation.

The current state-of-the-art pre-training approaches use large models consisting of a series of self-attention layers. These architectures are based on the Transformer model (Vaswani et al., 2017) first used for machine translation, achieving state-of-the-art results, and subsequently used in diverse contexts, e.g. language modeling (Dai et al., 2019), image generation (Zhang et al., 2018), and generalized to video classification, object detection/segmentation and human pose estimation (Wang et al., 2018b).

An alternative approach to multi-task learning on the GLUE benchmark was proposed concurrently to this work by Liu et al. (2019), who augment the BERT model with new output layers or procedures for most tasks, notably including (for some tasks) the stochastic answer network (Liu et al., 2018), a state-of-the-art neural natural language inference model. We see our approach as orthogonal to theirs, as we concentrate on augmenting BERT with a small number of parameters and do not consider changing our approach based on the specific GLUE task.

Another concurrent approach by Houlsby et al. (2019), introduces adapters very similar to our 'low-rank' layers (see section 3.3), and keep the BERT model fixed while training adapter modules. We concentrated on jointly fine-tuning the entire BERT model on all tasks. This has downsides: 1) Interference and 'forgetting' of stored knowledge is possible, 2) We require access to all tasks at training time. However the multi-task setup requires less adaptation parameters for good performance (we use $1.13\times$ parameters compared to their $1.3\times$ parameters[1] to match having seperate models for

---

[1] Although the results are not directly comparable since Houlsby

each GLUE task.), and is crucial for the transfer effects that gave us good performance on RTE.

3. Adapting Self Attention

The BERT model we are adapting is a multi-layer bidirectional Transformer encoder based on the original model of Vaswani et al. (2017). We only consider the smaller BERT-base model, which contains 110 million parameters. We somewhat arbitrarily limit ourselves to a $1.13\times$ increase in parameters, which is equivalent to 15 million total, or 1.9 million parameters per task.

In the following sections we first introduce various components of the full BERT model, and discuss how many parameters they require (section 3.1). We then show the exact form our parameter additions took, distinguishing between adding to the 'top' of the model, just before the output space (section 3.2), or within each layer of the BERT-base architecture (section 3.3).

3.1. Model Architecture and Multi-head Attention

BERT takes in a sequence (one or two English sentences in our case) and outputs a vector representation of that sequence. Each token in the sequence has its own hidden vector, and the first token of every sequence is always a special classification embedding (`[CLS]`). At each layer of BERT the hidden states of every sequence element are transformed, but only the final hidden state of `[CLS]` is used for classification/regression tasks. We now describe how the vector for one element of the sequence is transformed.

The multi-head attention layer (Vaswani et al., 2017) is the core of the transformer architecture that transforms hidden states for each element of the sequence based on the other elements (the fully-connected layers act on each element separately). The multi-head layer, which we write as $\mathrm{MH}(\cdot)$, consists of n different dot-product attention mechanisms. At a high level, with attention we compute a representation of a sequence based on a weighted sum of the hidden states of the sequence elements, and the weights are given by (in our case) dot product similarity of the hidden states.

Concretely, the ith attention mechanism takes the form:

$$\mathrm{Attention}_i(\mathbf{h}_j) = \sum_k \mathrm{softmax}\left(\frac{W_i^q \mathbf{h}_j \cdot W_i^k \mathbf{h}_k}{\sqrt{d/n}}\right) W_i^v \mathbf{h}_k \tag{1}$$

where $\mathbf{h}_j$ (we drop the j index in the following discussion) is a d dimensional hidden vector for a particular sequence element, and k runs over every sequence element. The W_i^q, W_i^k and W_i^v are matrices of size $d/n \times d$ (in BERT), and so in each 'head' we project down to subspaces of size

---

et al. (2019) use BERT-large and we use BERT-base.

d/n, meaning the heads can attend to different information. Finally the outputs of each attention head (which are also size d/n) are concatenated together (which we show as $[\cdot, ..., \cdot]$) and linearly transformed:

$$\text{MH}(\mathbf{h}) = W^o \left[\text{Attention}_1(\mathbf{h}), ..., \text{Attention}_n(\mathbf{h})\right] \quad (2)$$

with W^o a $d \times d$ matrix[2]. Throughout this section, we ignore terms linear in d (like bias terms) to avoid clutter, as they don't add significantly to the parameter count. The matrices in a multi-head layer have $3nd^2/n + d^2 = 4d^2$ parameters.

We further define another component of a BERT layer, the self-attention layer, which we write as $\text{SA}(\cdot)$:

$$\text{SA}(\mathbf{h}) = \text{FFN}(\text{LN}(\mathbf{h} + \text{MH}(\mathbf{h}))), \quad (3)$$

$\text{LN}(\cdot)$ is *layer normalisation* (Ba et al., 2016), requiring $2d$ parameters. FFN is a standard *feed-forward network*,

$$\text{FFN}(\mathbf{h}) = W_2 f(W_1 \mathbf{h} + b_1) + b_2, \quad (4)$$

with $f(\cdot)$ a non-linearity, GeLU (Hendrycks & Gimpel, 2016) in BERT. Matrix W_1 has size $d_{ff} \times d$ and W_2 has size $d \times d_{ff}$, so overall we require $2dd_{ff}$ parameters from the FFN component.

Putting this together, a BERT layer, which we write $\text{BL}(\cdot)$, is layer-norm applied to the output of a self-attention layer, with a residual connection.

$$\text{BL}(\mathbf{h}) = \text{LN}(\mathbf{h} + \text{SA}(\mathbf{h})) \quad (5)$$

We have $4d^2 + 2dd_{ff}$ total parameters from a BERT layer.

The entire BERT model is simply a stack of 12 BERT layers, followed by (in our case) a transformation to take us to the output space for a NLU task. We write the dimensions of the hidden states in BERT-base as $d_m = 768$. The final hidden state of the first token of every sequence is all that is used for the transformation to the output.

The exact form of the transformation applied to the final hidden state of the [CLS] token is a simple $d \times d$ linear transformation, known as a **'pooling layer'**, followed by another matrix multiply that projects to the output space. Since the output space is always three dimensional or less in our case, this projection does not require many parameters, but d^2 parameters per task from separate pooling layers adds significantly to our parameter budget. It is not obvious if we can share the pooling layer across all tasks, and we found that when sharing this layer we needed to use a non-standard training schedule; see section 4.1.

---

[2]Vaswani et al. (2017) provide a more detailed motivation and discussion.

3.2. Adding Parameters to the Top

The simplest way to add parameters to a model is to add them at the 'top' of the model, i.e. just before the classification layer.

We get our final hidden state for [CLS], $\mathbf{h}^f$, from the original vector embedding of [CLS], $\mathbf{h}^0$, by

$$\mathbf{h}^f = \text{TS}(\text{BERT}(\mathbf{h}^0)), \quad (6)$$

where $\text{TS}(\cdot)$ is a task-specific function that can potentially operate on a single vector, but depends on the entire sequence when it contains attention layers. $\text{BERT}(\cdot)$ always depends on the entire sequence, and is shared across tasks.

The benefits of this form are that at inference time we only apply $\text{BERT}(\mathbf{h}^0)$ once (assuming the setting where we perform multiple tasks on the same piece of text), which saves significantly on total operations because each $\text{TS}(\cdot)$ requires much fewer operations than the main BERT model.

The simplest form for the task-specific transformation of the hidden state $\text{TS}(\cdot)$ would be a linear transform. However this requires d_m^2 parameters, and d_m is fairly large even for BERT-base. The linear transform does not violate our 15 million parameter constraint, but we expect there are more efficient ways to add parameters.

Another obvious transformation, adding an extra BERT layer for each task, results in approximately a $1.67\times$ increase in number of parameters, or 73 million new parameters. d_{ff} is $4d_m$ for BERT, so for a BERT layer we get $4d_m^2 + 2d_m d_{ff} = 12d_m^2$ parameters. We include this architecture in our experiments for comparison, with the caveat that it requires many more parameters than our alternatives.

To avoid transformations requiring $O(d_m^2)$ parameters, we propose using task-specific functions of the form

$$\text{TS}(\mathbf{h}) = V^D g(V^E \mathbf{h}), \quad (7)$$

where V^E is a $d_s \times d_m$ 'encoder' matrix, V^D is a $d_m \times d_s$ 'decoder' matrix with $d_s < d_m$, and $g(\cdot)$ is an arbitrary function. Because we can make d_s as small as we like, $g(\cdot)$ can be composed of multiple layers of transformations, and not impose a large parameter budget. We found $d_s = 204$ worked well, and allowed us to stay within our $1.13\times$ parameter limit.

We experiment with these choices for each layer of $g(\cdot)$:

- Multi-head attention, optionally followed by a residual connection and layer-norm. We refer to this method as **Projected Attention**.

- A one or two layer feed-forward network followed by a residual connection and layer-norm, such that it has the same number of parameters as the previous form;
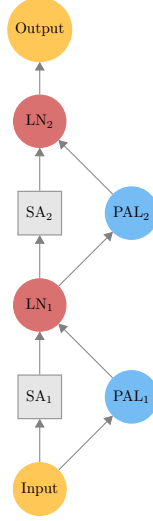
Figure 1. Schematic diagram of adding a task-specific function (here our 'Projected Attention Layers' or PALs) in parallel with self-attention (SA) layers in a BERT model (see section 3.3), with only two layers for simplicity. LN refers to layer-norm.

this means the intermediate layer is of size 408 (for a one layer network) or 252 (for a two layer network).

3.3. Adding Parameters within BERT

Instead of adding parameters to the top of the model, we may want to modify the $\text{BERT}(\cdot)$ function itself, in a way reminiscent of the 'residual adapter modules' of Rebuffi et al. (2018). Specifically, we wish to add task-specific parameters to each layer of the BERT model. See figure 1 for an illustration.

We can add a task-specific function 'in parallel' with each BERT layer as follows:

$$\mathbf{h}^{l+1} = \text{LN}(\mathbf{h}^l + \text{SA}(\mathbf{h}^l) + \text{TS}(\mathbf{h}^l)) \qquad (8)$$

where l indexes the layer. This means we recover the original BERT model if $\text{TS}(\cdot)$ outputs a zero vector. Alternatively we can add a 'serial' connection where we transform the output of a BERT layer:

$$\hat{\mathbf{h}}^{l+1} = \text{LN}(\mathbf{h}^l + \text{SA}(\mathbf{h}^l)) \qquad (9)$$

$$\mathbf{h}^{l+1} = \text{LN}(\hat{\mathbf{h}}^{l+1} + \text{TS}(\hat{\mathbf{h}}^{l+1})). \qquad (10)$$

Serial connections gave consistently much worse results than parallel connections and we report results for parallel connections in what follows.

We again consider task-specific functions of the form:

$$\text{TS}(\mathbf{h}) = V^D g(V^E \mathbf{h}), \qquad (11)$$

Table 1. How parameters are 'spent' for some of our methods, where T is the number of tasks, and there are 12 layers in the base network. The $2d_m d_s$ terms come from 'encoder' and 'decoder' matrices. PALs (section 3.3) use $3d_s^2$ parameters per multi-head layer (see section 3.1) rather than $4d_s^2$ because they do not use the final linear transform W^o. Projected attention (section 3.2) worked best with six rather than twelve layers.

METHOD	PARAMETERS
PALs	$T(2d_m d_s + 12 \times 3d_s^2)$
LOW RANK	$T(12 \times 2d_m d_s)$
PROJ. ATTN. ON TOP	$T(2d_m d_s + 6 \times 4d_s^2)$

with the difference that V^E (again a $d_s \times d_m$ matrix with $d_s < d_m$) and V^D (again a $d_m \times d_s$ matrix) are needed at each layer rather than only once each.

We experiment with $g(\cdot)$ taking the following forms:

- The identity function; This means our task-specific transform is just a low-rank linear transformation at each layer. To satisfy our parameter constraint we need $d_s = 100$. We refer to this method as **Low-rank Layers**.

- Multi-head attention. To satisfy our parameter constraint we need $d_s = 84$. We found that it was not necessary to use the W^o matrix (see section 3.1) when adapting within BERT, and did not use in any of our models.

- Multi-head attention, with shared V^E and V^D across layers (not tasks). This parameter sharing allows a larger $d_s = 204$. We refer to this method as **Projected Attention Layers (PALs)**.

- Shared V^E and V^D across layers, but with $g(\cdot)$ a feed-forward network with intermediate size 306 instead of attention (and again $d_s = 204$).

The motivation behind PALs is that we want to 'spend' our parameter budget on transformations with an inductive bias useful for sequences. The 'encoder' and 'decoder' matrices operate on each sequence element separately, unlike attention, which transforms the input based on the entire sequence. Finally, the attention mechanism of PALs can potentially be inspected to see which tokens in a sequence the task-specific parts of the model focus on, although we did not concentrate on this aspect in this work.

4. Multi-task Training and Experimental Setup

4.1. Sampling Tasks

A simple way to train a model on several tasks is to select a batch of training examples from each task, cycling through them in a fixed order. We refer to this as 'round-robin' sampling. However if each task has a different number of training examples then by the time we have seen every example from a particular task we could have looped through another, smaller task's dataset many times. This could lead to over-fitting on smaller tasks, and under-training on larger tasks. Potentially we could alleviate this issue by manually tuning regularisation hyper-parameters for each task.

Alternatively we can use methods where we see more examples from tasks with larger associated datasets. Concretely, we select a batch of examples from task i with probability p_i at each training step, and set p_i proportional to N_i, the number of training examples for task i:

$$p_i \propto N_i. \tag{12}$$

This is the approach of the multi-task BiLSTM of Wang et al. (2018a) on the GLUE benchmark, and was used by Sanh et al. (2018). It has the appealing property of selecting each example with the same probability as as combining all the tasks and picking examples uniformly (though we train on batches from each task not single examples).

Since the ratio of the largest to the smallest task sizes N_i we use is $\approx$158, we only rarely train on some tasks with the simple $\propto N_i$ method. Training on one task (or a particular subset of tasks) for many steps will lead to interference, where performance on the other tasks suffers. A more general approach to sampling tasks sets p_i as:

$$p_i \propto N_i^{\alpha}. \tag{13}$$

If we choose $\alpha < 1$ we reduce the disparity between the probabilities of choosing tasks. We consider $\alpha = 0.5$ in our experiments, and call this method **'square root sampling'**.

Finally, we noticed that it was beneficial to train on tasks more equally towards the end of training, where we are most concerned about interference, and so we constructed the **'annealed sampling'** method where α changes with each epoch[3] e:

$$\alpha = 1 - 0.8 \frac{e-1}{E-1}, \tag{14}$$

where E is the total number of epochs. So at epoch 1 we use $\alpha = 1 - 0.8 \frac{0}{E-1} = 1$, and in our final epoch we use

---

[3]Since we used multiple datasets we chose a somewhat arbitrary 'epoch' of 2400 training steps.

$\alpha = 1 - 0.8 \frac{E-1}{E-1} = 0.2$, and linearly move closer to $\alpha = 0.2$ as training goes on.

It was particularly important to use the square root or annealed sampling methods when sharing a pooling layer (see section 3.1), and it makes intuitive sense that when the layer just before the output is shared we need to guard against interference between tasks.

4.2. Setup

We based our experiments on the the PyTorch implementation of BERT [4]. No matter how we sampled tasks, we (unless stated otherwise) trained for 60,000 steps, with a minibatch size of 32, and a maximum sequence length of 128 tokens. We use Adam with learning rate of 2×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, L2 weight decay of 0.01, learning rate warmup over the first 10% of steps (usually 6,000), and linear decay of the learning rate after this, going down to zero at the end of training. We note warmup followed by linear decay is the 'slanted triangular learning rate' of Howard & Ruder (2018), who find it is suited for fine-tuning a language model on single tasks. We performed most of our experiments using either the 'proportional', 'square root' or 'annealed' sampling methods (see section 4.1). Round robin sampling gave consistently worse results.

We use twelve heads for the attention mechanism in PALs and other methods, except when using a smaller hidden size, where we decreased it proportionally. We did not find significant performance differences when changing the number of heads. We used the same BERT-base architecture as by Devlin et al. (2018), twelve attention heads, $d_{ff} = 3072$ and $d_m = 768$ (see section 3.1).

We found it was crucial to use the pre-trained weights [5] for BERT-base and not start from scratch. When training from scratch, with adaption parameters or not, we got significantly worse performance. For some tasks we did not get better results than random guessing after 90,000 steps. Although we note we used the same hyper-parameters as when training from the pre-trained weights, which might not be optimal for starting from scratch. We experimented briefly with freezing the BERT-base parameters and fine-tuning only the PALs and alternatives, but to approach matching the performance of fine-tuned BERT-base it was crucial to fine-tune all of the model weights.

---

[4]https://github.com/huggingface/pytorch-pretrained-BERT
[5]We use the pre-trained BERT-base model released by Devlin et al. (2018).

4.3. Details of GLUE Tasks

We test our methods for multi-task adaptation on eight of the nine tasks in the GLUE benchmark (Wang et al., 2018a)[6].

Single-sentence tasks: Acceptability classification with CoLA (Warstadt et al., 2018); binary sentiment classification with SST (Socher et al., 2013).

Sentence pair tasks: Semantic similarity with the MSR Paraphrase Corpus (MRPC: Dolan & Brockett, 2005), STS-Benchmark (STS: Cer et al., 2017) and Quora Question Pairs (QQP) dataset, and textual entailment with Multi-Genre NLI Corpus (MNLI: Williams et al., 2018), a subset of the RTE challenge corpora (Dagan et al., 2006), and data from SQuAD (QNLI: Rajpurkar et al., 2016).

Like Devlin et al. (2018) we exclude the Winograd NLI task. When systems are trained on this task they have always performed worse than the 65.1 baseline accuracy of predicting the majority class. For our submissions we also simply predicted the majority class.

5. Experiments and Discussion

Table 2 lists our results on GLUE for our best-performing PAL model (chosen by average development set performance), and some alternatives. Our main comparison is against fine-tuned BERT-base, which in the absence of transfer effects represents an upper bound on our performance, since it involves tuning all BERT-base parameters to perform well on each task individually, therefore requiring approximately $8\times$ as many parameters as our methods. By construction, apart from our adaptation parameters we use the exact same architecture as BERT-base. We note that with the exception of our results for RTE, better performance can be obtained by fine-tuning the BERT-large model that has approximately $3\times$ the parameters of BERT-base.

The use of multi-task training significantly improves results on the RTE task, achieving state-of-the-art performance. Similar improvements have been observed with multi-task LSTM-based systems (Wang et al., 2018a) and by pretraining on MNLI before fine-tuning on RTE (Phang et al., 2018). Since RTE has the smallest number of training examples, and is similar to MNLI, it makes intuitive sense that it benefits from multi-task training. Sharing more parameters increased performance on RTE, and our fully-shared model has slightly better performance on RTE than PALs, however PALs are the only model that matches BERT-base on the larger tasks as well as performing well on RTE.

For the large sentence-pair tasks, MNLI, QQP and QNLI, performance is almost exactly the same as BERT-base with

PALs. For the two single sentence tasks: the syntax-oriented CoLA task and the SST sentiment task we see the largest drops in performance with PALs. This is in agreement with the results of Phang et al. (2018) who did not observe any transfer from various intermediate tasks, and, for CoLA, mirrors the results of Bowman et al. (2019) that language modeling alone is the best pre-training task for CoLA.

5.1. PALs and Alternatives

Table 3 lists our results on the GLUE benchmark development set for various ways of adding task-specific parameters and sampling strategies.

Our best results came with PALs adapting every layer within BERT. The performance of PALs increased with a larger hidden state. Having separate 'encoder' and 'decoder' matrices (see section 3.3) across layers, or having separate pooling layers for each task, with the appropriate reduction in hidden state size to make up for the extra parameters, resulted in worse performance for PALs. However sharing 'encoder' and 'decoder' matrices between *tasks* or both layers and tasks hurt results. A larger hidden state size seems important for Transformer models[7], e.g. the performance of BERT-large vs. BERT-base (Devlin et al., 2018) or the ablation study by Vaswani et al. (2017).

We tested two adaption layers that did not use attention: Low-rank layers, and our method with shared 'encoder' and 'decoder' matrices but with a small feedforward network inbetween them instead of attention. The latter model did not achieve good performance, but low-rank layers and PALs have roughly equivalent mean performance (within standard error). By inspecting the best-performing single models of each method we see a contrast: the strong results for low-rank layers are from better performance on CoLA[8], while PALs consistently perform better for the three largest tasks, MNLI, QQP and QNLI, and equivalently for other tasks.

We see this as evidence for PALs having greater representational capacity; the only model that achieved comparable performance on the large tasks was adding an entire BERT-layer to the top, but this model had worse performance on the RTE task and uses many more parameters. The fact that 'spending' parameters on linear transforms in the encoder, decoder or pooling matrices gives worse performance, and the worse performance of feedforward layers compared to multi-head attention, points towards the inductive bias provided by attention being important for good performance.

However at sufficiently parameter constrained regimes (for

---

[6]Wang et al. (2018a) provide a more detailed discussion of these tasks.

[7]And similarly for LSTMs, e.g. language modeling results on the billion word benchmark (Józefowicz et al., 2016).

[8]CoLA tends to see larger changes in score between models than other tasks since it is scored by Matthews correlation coefficient not accuracy.

Table 2. GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. We show F1/accuracy scores for QQP and MRPC, and accuracy on the matched/mismatched test sets for MNLI. The 'Av.' column is slightly different than the official GLUE score, since we exclude WNLI. 'Bert-base' results are from Devlin et al. (2018). 'Shared' refers to the model where all parameters are shared except the final projection to output space. The models we tested are a result of the 'annealed sampling' method for multi-task training as it produced the best results on the dev set.

METHOD	PARAMS	MNLI-(M/MM) 392K	QQP 363K	QNLI 108K	SST-2 67K	CoLA 8.5K	STS-B 5.7K	MRPC 3.5K	RTE 2.5K	Av.
BERT-BASE	8×	<u>84.6</u>/83.4	<u>89.2</u>/71.2	<u>90.1</u>	<u>93.5</u>	<u>52.1</u>	<u>85.8</u>	<u>84.8</u>/<u>88.9</u>	66.4	79.6
SHARED	1.00×	84.0/83.4	88.9/70.8	89.3	93.4	51.2	83.6	81.3/86.7	<u>76.6</u>	79.9
TOP PROJ. ATTN.	1.10×	84.0/83.2	88.8/71.2	89.7	93.2	47.1	85.3	83.1/87.5	75.5	79.6
PALs (204)	1.13×	84.3/<u>83.5</u>	<u>89.2</u>/<u>71.5</u>	90.0	92.6	51.2	<u>85.8</u>	84.6/88.7	76.0	**80.4**

Table 3. GLUE performance, in terms of average score across each task's development set; this score is accuracy except for CoLA, where it is Matthews correlation, and STS-B, where it is Pearson correlation. We show the mean and standard error over three random seeds, unless standard error is < 0.005. For the details of the sampling strategies see section 4.1. For the 'within BERT' methods we show the smaller hidden state size in brackets, and write 'top' to mean adding in parallel to the six BERT layers just before the output, and 'bottom' to mean adding in parallel to the six BERT layers just after the input.

METHOD	NO. PARAMS	NEW LAYERS	PROP. SAMP.	SQRT. SAMP.	ANNEAL SAMP.
SHARED	1.00×	0	79.17±0.03	80.56±0.04	80.7±0.3
ADDING ON TOP OF BERT					
BERT LAYER	1.66×	1	80.6±0.2	81.6±0.3	81.5±0.2
PROJ. ATTN.	1.10×	6	80.3±0.1	<u>81.4±0.1</u>	81.5±0.1
PROJ. FFN (1 LAYER)	1.10×	6		81.07	80.8±0.1
ADDING WITHIN BERT					
PALs (204)	1.13×	12	<u>80.6±0.2</u>	81.0±0.2	**81.7±0.2**
LOW RANK (100)	1.13×	12			**81.9±0.2**
PALs (276, TOP)	1.13×	6			81.61±0.06
PALs (276, BOTTOM)	1.13×	6			81.4±0.1

example 1.5 million parameters, which implies $d_s = 10$ for low-rank transforms, and $d_s = 60$ for PALs), PALs and low-rank layers performed similarly to the fully-shared model. Using the LHUC method (see section 2.2), which requires even fewer parameters, also gave no improvement over the fully-shared baseline.

When adding parameters to the top of BERT-base, again it was important to use attention rather than feedforward transforms. Six additional layers worked best, outperforming using twelve or three layers. We also found it was crucial to use layer-norm and residual connections after each application of attention. Surprisingly, for these models using a separate pooling layer did not noticeably change results, and we report results with a shared pooling layer, which requires fewer parameters. These models saw worse performance on the RTE task, perhaps because transfer from other tasks is important, and splitting the model into multiple 'heads' for each task dampens the benefits of shared knowledge.

5.2. Where should we add Adaptation Modules?

We draw some of the same conclusions as Rebuffi et al. (2018) for 'residual adapter modules'. As that previous work was multi-task computer vision with residual networks (section 2.2), we hope that these principles will apply broadly.

Adding task-specific functions *within* networks works better than adding them to the *top* (for a given number of parameters). As found by Rebuffi et al. (2018), the best performing models had adaptations at *every layer* of the base network, and adding adapter modules to the *final half* of the base model worked better than adding to the half *just after the input*. Unfortunately, adapting every layer of the base model represents the worst case for sharing operations between tasks. (We note again that this sharing is possible only when we want to perform many tasks on the same piece of text). But adapting the final half achieved slightly better performance than adding to the top of BERT-base. When adapting the final half we can still share the first six layers worth of operations, offering a useful compromise.

For within-network adaptations, *parallel* connections worked better than *serial* ones, also as found by Rebuffi et al. (2018). Our results with serial connections were much worse than simply not including any adapters. While the parallel configuration acts as a perturbation on the base network, the serial configuration more directly changes the hidden states being fed into the next layer. In these ways, the parallel configuration is less prone to the loss of the 'knowledge' stored in the base network.

6. Further Discussion

We found the details of how to schedule training examples from each task were important. With a lot of parameter sharing, sampling tasks proportional to dataset size impaired performance compared to our 'annealing' method, where we slowly decrease the influence of dataset size on sampling probability. Annealing increased the variance of performance across random seeds as well as mean performance, meaning that we may need to pay the cost of several training runs to obtain the best single models from this method. We did not consider many variations of training method, and used no methods to reduce interference from training on separate tasks (to take one example, the 'Gradient Episodic Memory' of Lopez-Paz & Ranzato (2017)). How these methods interact with choice of adaptation parameters is a direction for further research.

We introduced 'Projected Attention Layers' as a transformation that can adapt the BERT sentence representation model for multi-task learning. PALs give a higher capacity for a given number of parameters compared to all the alternatives we considered. If we adapt all the layers of BERT-base, we cannot share any operations across tasks. Ultimately the choice of which method to use depends on the constraints in place; if parameters are less constrained but you want to share as many operations as possible, adding an entire task-specific BERT layer on top of the model makes sense. If shared operations are not an issue adding PALs to every layer will perform well with few parameters. And adapting only the final half of the base model offers a compromise between performance and sharing operations.

Acknowledgements

References

Ba, J., Kiros, R., and Hinton, G. E. Layer normalization. *CoRR*, abs/1607.06450, 2016.

Bowman, S. R., Pavlick, E., Grave, E., Durme, B. V., Wang, A., Hula, J., Xia, P., Pappagari, R., McCoy, R. T., Patel, R., Kim, N., Tenney, I., Huang, Y., Yu, K., Jin, S., and Chen, B. Looking for ELMo's friends: Sentence-level pretraining beyond language modeling, 2019.

Caruana, R. Multitask learning. *Mach. Learn.*, 28(1): 41–75, July 1997. ISSN 0885-6125. doi: 10.1023/A: 1007379606734.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14. Association for Computational Linguistics, 2017. doi: 10.18653/v1/S17-2001.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12: 2493–2537, November 2011. ISSN 1532-4435.

Dagan, I., Glickman, O., and Magnini, B. The pascal recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, MLCW'05, pp. 177–190, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-33427-0, 978-3-540-33427-9. doi: 10.1007/11736790_9.

Dai, A. M. and Le, Q. V. Semi-supervised sequence learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 3079–3087. Curran Associates, Inc., 2015.

Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-XL: Language modeling with longer-term dependency, 2019.

Deng, L., Hinton, G., and Kingsbury, B. New types of deep neural network learning for speech recognition and related applications: an overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8599–8603, May 2013. doi: 10.1109/ICASSP.2013.6639344.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

Dolan, W. B. and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Duong, L., Cohn, T., Bird, S., and Cook, P. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 845–850. Association for Computational Linguistics, 2015. doi: 10.3115/v1/P15-2139.

Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. A joint many-task model: Growing a neural network for multiple nlp tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1923–1933. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1206.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *ECCV*, 2016.

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

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-Efficient Transfer Learning for NLP. *CoRR*, abs/1902.00751, 2019.

Howard, J. and Ruder, S. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 328–339. Association for Computational Linguistics, 2018.

Józefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016.

Liu, X., Shen, Y., Duh, K., and Gao, J. Stochastic answer networks for machine reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1694–1704. Association for Computational Linguistics, 2018.

Liu, X., He, P., Chen, W., and Gao, J. Multi-task deep neural networks for natural language understanding. *CoRR*, abs/1901.11504, 2019.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and

Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6467–6476. Curran Associates, Inc., 2017.

McCann, B., Keskar, N. S., Xiong, C., and Socher, R. The natural language decathlon: Multitask learning as question answering. *CoRR*, abs/1806.08730, 2018.

Phang, J., Févry, T., and Bowman, S. R. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *CoRR*, abs/1811.01088, 2018.

Radford, A. Improving language understanding by generative pre-training. 2018.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1264.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. Efficient parametrization of multi-domain deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2 2018.

Ruder, S. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

Sanh, V., Wolf, T., and Ruder, S. A hierarchical multi-task approach for learning embeddings from semantic tasks. *CoRR*, abs/1811.06031, 2018.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642. Association for Computational Linguistics, 2013.

Subramanian, S., Trischler, A., Bengio, Y., and Pal, C. J. Learning general purpose distributed sentence representations via large scale multi-task learning. In *International Conference on Learning Representations*, 2018.

Swietojanski, P. and Renals, S. Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pp. 171–176, Dec 2014. doi: 10.1109/SLT.2014.7078569.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4496–4506. Curran Associates, Inc., 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355. Association for Computational Linguistics, 2018a.

Wang, X., Girshick, R., Gupta, A., and He, K. Non-local neural networks. *CVPR*, 2018b.

Warstadt, A., Singh, A., and Bowman, S. R. Neural network acceptability judgments. *CoRR*, abs/1805.12471, 2018.

Williams, A., Nangia, N., and Bowman, S. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-1101.

Yang, Y. and Hospedales, T. M. Trace norm regularised deep multi-task learning. In *ICLR Workshop*, 2017.

Zhang, H., Goodfellow, I. J., Metaxas, D. N., and Odena, A. Self-attention generative adversarial networks. *CoRR*, abs/1805.08318, 2018.