AdaptBert: Parameter Efficient Multitask Bert

Stanford CS224N Default Project

Shweta Agrawal
Stanford University
ashweta@stanford.edu

Jieting Qiu
Stanford University
jtqiu09@stanford.edu

Abstract

In this project, we explore a range of parameter efficient multitask learning methods to train Bert efficiently for multiple downstream NLP tasks. While fine-tuning the whole LLM is an effective transfer learning method and sets the bar for the accuracy on downstream tasks, full fine-tuning is parameter inefficient, requiring retraining a whole new model, updating, and potentially storing, a large number of parameters for every new task. We show that similar performance as full fine-tuning can be achieved using parameter efficient fine-tuning (PEFT) methods like LoRA, Adapter and Hypernetworks. We study each of these methods with a range of parameters, and also come up with a LoRA+Adapter combination, that is able to achieve similar performance overall and slightly better on some tasks compared to full multi-task fine-tuning while updating only 17% of the parameters per task.

1 Key Information to include

Olivia Lee was our assigned mentor. We do not have any external collaborators. And we are not sharing the project with any other course.

Contribution: Equal. We both independently finished part 1 implementation. Shweta implemented LoRA and Adapter, and the LoRA+Adapter combination. Jieting also did an implementation of Adapters, and implemented Hypernetworks. We both debugged and tuned various runs together and contributed to the report.

2 Introduction

Large language models (LLMs) pretrained on unlabelled text have been very effective in getting better accuracy in solving downstream tasks, through transfer learning. Pretrained models like Bert Devlin et al. (2018) are trained on large set of unlabeled data on generic tasks like detecting masked words, using a “masked language model" (MLM) pretraining objective. For fine-tuning, the Bert model is first initialized with the pretrained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. This full fine-tuning is very effective in transfer learning for downstream NLP tasks. However, in the presence of many downstream tasks, fine-tuning is parameter inefficient: an entire new model and retraining of 100% of parameters is required for every task. Parameter efficient fine-tuning (PEFT) methods enable efficient adaptation of LLMs to various downstream applications by only updating a small number of extra model parameters, hence saving computational and storage costs.

In this project, we explore three different PEFT methods LoRA Hu et al. (2021), Adapters Houlsby et al. (2019), and Hypernetworks-based Adapter Mahabadi et al. (2021), along with creating a new Adapter+LoRA variant by applying task-specific Adapter and shared LoRA attention adaptation (Figure 5 in appendix). We report our findings on three downstream tasks of Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity. We show a clear relationship between accuracy and number of parameters to train. This relationship is however not the same for the three methods. For the given dataset and tasks, Adapters being the simplest also seem to scale

the best. Ultimately, we are able to achieve same, or even slightly better accuracy on some tasks, with a multitask model training only a fraction of parameters, with LoRA+Adapter getting the best performance and parameter efficiency.

Semantic Analysis seems to be the most difficult task to get higher accuracy on. Hence, we additionally explore various hyper-parameter tuning for getting better performance on this task, including learning rate, dropout and weight decay parameters. We achieve slightly better accuracy with each of these methods, but none of them seem to produce dramatically better accuracy, indicating that this likely benefit from additional data or pretraining. We also investigate different loss functions for Semantic Textual Similarity, and find that the MSELoss works much better for compared to Cross entropy loss, by only needing to estimate one logit and Pearson coefficient used in the accuracy.

3 Related Work

Natural Language Processing (NLP) comprises of a wide variety of language understanding tasks. While the unlabelled natural language text is abundant on internet, labelled data for each of these specific tasks is limited. Pretraining a large model on a generic language understanding task predicting next word has been shown to be useful for effective for improving the accuracy and performance of Natural language processing tasks Radford et al. (2018). There are many ways with which these pretrained large language models can be used for downstream NLP tasks. Zero shot learning Radford et al. (2019) shows the ability of the model to do many NLP tasks with no examples; and no gradient updates. Few shot learning Brown et al. (2020) enables using a pretrained LLM for a new task by simply prepending examples of the task before your example. Sophisticated prompting (prompt-tuning or prompt-engineering) techniques have been explored to get LLMs to do new tasks, by just using better prompts, example Zero shot promot design for text classification in Hase et al. (2021).

However, many NLP tasks are too difficult to be solved by prompting alone. It is tedious to provide prompts every time, and the accuracy is limited that way. Hence, models are fine-tuned to specific downstream tasks. This idea has been used to achieve better accuracy for many downstream NLP tasks, for example with Bert Devlin et al. (2018). Full fine-tuning of all parameters is however expensive with large models. Storing and serving a separate full fine-tuned model for every task is also expensive and impractical. Parameter efficient multitask fine-tuning methods aim to train a single model for multiple downstream tasks, training only a fraction of the parameters to achieve same or better accuracy on downstream tasks through transfer learning of shared frozen parameters, and learning some task specific parameters.

Some of these techniques make use of sparsity in model parameters, and look to specifically prune some parameters before adaptation, by using some heuristics on which parameters might be less important Han et al. (2017), before adapting to specific tasks. A large class of techniques have been explored in some sort of low rank adaptation. LoRA Hu et al. (2021) does the adaptation using a low rank updates to the weights mostly in the attention layer, QLoRa improves it further by quantizing to 4-bit precision Dettmers et al. (2023), prefix tuning Li and Liang (2021) adds a task specific prefix of parameters, adapters Houlsby et al. (2019) introduce task specific adapter modules in each layer after attention and feed-forward modules, and Hypernetworks use a common adapter network, conditioned on task, layer and position Mahabadi et al. (2021).

Note: Many of the references are inspired by Diyi Yang's lecture on Efficient Adaptation.

4 Approach

We implement the basic Bert model attention and forward methods, as well as single and multitask classifiers as instructed in the handout. We use the following loss functions for the three downstream NLP tasks:

- Sentiment Analysis (SST): Apply a linear layer to product num_labels (5) logits, and use a cross entropy loss with labels.
- Paraphrase detection (Para): Apply a linear layer to produce one logit, and use binary cross entropy loss (BCELoss) against the labels.

- Semantic Textual Similarlity (STS): Apply a linear layer to produce one logit, and use mean square error (MSELoss) against the labels. We also tried the cross entropy loss but MSELoss gave significantly better performance.

Using these loss functions, we first train and evaluate the downstream tasks using option pretrain for which base Bert model parameters are frozen. This sets lower baseline for accuracy on these tasks. We then evaluate single-task fine-tuning for Semantic analysis using SST and CFIMDB datasets, that fine-tunes all parameters of the model into a separate model for each task. And, train a full fine-tune multi-task model for all three tasks that updates 100% of the parameters of the pretrained model to give one multitask model for all downstream tasks. These fully fine-tuned models set the bar for higher end of accuracy achievable by parameter efficient fine-tuning.

We then implement the following extensions for parameter efficient fine-tuning (PEFT):

4.1 LoRA

LoRA or Low Rank Adaptation (Hu et al., 2021), represents the updates to the original attention weight vectors with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in R^{d \times r}$, $A \in R^{r \times k}$, and the rank $r \ll min(d, k)$. During training, W_0 is frozen, while A and B contain trainable parameters. LoRA allows us to train some dense layers in the network indirectly by updating only smaller rank decomposition matrices during adaptation instead. We implement LoRA adaptation in attention layers from scratch for query, key and value vectors, adding a LoraBertSelfAttention module derived from BertSelfAttention, taking inspiration from the towardsdatascience blog.

4.2 Adapter

Adapter (Houlsby et al., 2019) represents task specific module that first down-projects input to a smaller adapter dimension, followed by a non-linear activation, followed by up-project back to the input dimension. We implement Adapter modules ourselves in the Bert code, following the ideas from paper (Houlsby et al., 2019). Each layer of the Transformer contains two primary sub-layers: an attention layer and a feed-forward layer. As described in the paper, we insert two serial adapters after each of self-attention and feed-forward layer, with output of these adapters fed into the corresponding add-norm operation. The total number of parameters added per layer, is $2md + m$ (where m is bottleneck dimension and d is the original dimension), since we do not use extra bias terms. During training, the parameters of the original Bert model remain frozen, and only the parameters of the task specific adapter modules are updated. We report the accuracy for different adapter dimension size which influence percentage of updatable parameters.

4.3 LoRA + Adapter

Our implementation of LoRA shares all the low rank attention weight matrices across tasks. This provide low parameter updates and transfer learning. However, the adapters are task specific and provide task isolation. We implement a combination of LoRA+Adapter to get both the benefits and find that it can achieve same or better accuracy compared to LoRA or Adapter alone using less number of parameter updates.

4.4 Hypernetwork

Task-specific Adapter (Houlsby et al., 2019) achieves decent transfer-learning performance by reducing negative interference across tasks by encapsulating task-specific information. However, adapter modules are trained disjointly for each task and thus do not share information across tasks. We follow the paper (Mahabadi et al., 2021) to implement shared hypernetworks in adapter modules that shares the information across all layers and tasks by computing conditional embeddings using task name, layer id and the position of the adapter (after self-attention block or after feed-forward block). We use a **single** embedding network to generate task embedding and **two** sets of hypernetworks to generate parameters to feed into down-projection, up-projections and layer-norm (as in Figure 6 in Appendix). We have to use two instead of one is because adapters after self-attention and feed-forward layers have different layer dimensions. In the following sections, we present shared Hypernetworks-based adapter modules.

One Shared Conditional Task Embedding Network: In order to allow share information across tasks, we first compute a task embedding conditioned on task, layer and adapter position by using a task project network $h_I(.)$, consisting of a feed-forward layer (FFL), a ReLU non-linearity and a feed-forward projection (FFP). We use task, layer id, and position id to build a concatenation of task embedding $Z_t \in \mathbb{R}^t$, layer embedding $l_i \in \mathbb{R}^t$, position embedding $p_j \in \mathbb{R}^t$ to feed into the task projector network. And then followed by a layer-norm (LN) to get the final task embedding:

$$\mathbf{I}_t = h_t(Z_t, l_i, p_j) = LN(FFP(ReLU(FFL(Z_t, l_i, p_j)))) \tag{1}$$

Conditional Hypernetworks: A hypernetwork is a network that generates the weights of another network (David Ha, 2017). We define hypernetwork $h^l_A(.)$ and $h^l_{LN}(.)$ that consist of simple linear layers to take in conditioned embeddings and generate parameters for adapters and layer-norms respectively.

$$(U_t, D_t) := h_A(\mathbf{I_t}) = (\mathbf{W}^{U_t}, \mathbf{W}^{D_t})\mathbf{I_t} \tag{2}$$

$$(\beta_t) := h_{LN}(\mathbf{I_t}) = (\mathbf{W}^{\beta})\mathbf{I_t} \tag{3}$$

where $\mathbf{W}^U$, $\mathbf{W}^D$ are parameters for the down-projection layer $\mathbf{D}_t$ and up-projection $\mathbf{U}_t$ respectively, and $\mathbf{W}^\beta$ is parameter of layer-norm.

Conditional Adapter Layers and Layer Normalization: We use the generated weights and biases from the conditional hypernetworks to apply on adapter layers and layer-norm in each Adapter block as in Figure 6. The adapter layer A_t consists of a down-projection $D_t \in \mathbb{R}^{h \times d}$, a GeLU non-linearity and up-projection $U_t \in \mathbb{R}^{d \times h}$. Additionally, we apply a task conditional layer normalization LN_t, with x is the input hidden state:

$$A_t(x) = LN_t(U_t(GeLU(D_t(\mathbf{x})))) + \mathbf{x} \tag{4}$$

4.5 Hyperparameter Tuning

Sentiment Analysis seems to be the most difficult task among the three NLP tasks, to achieve higher accuracy on, with given datasets. We also observe that while training accuracy for Sentiment Analysis on SST dataset keeps increasing, the eval accuracy does not improve as much, and in some cases start going down around middle epoch 3 Hence, we try various options to tune Sentiment Analysis training to reduce over-fitting, and try the raytune to find the optimal hyper-parameters.

5 Experiments

5.1 Data

We use the task specific labelled datasets linked in the handout for fine-tuning the three downstream tasks (separating train and dev for training and evaluation, and test set for testing).

- Sentiment Analysis
  - ☐ Stanford Sentiment Treebank (SST) dataset with 11,855 sentences from movie reviews labeled with 5 ratings from negative to positive.
  - ☐ CFIMDB dataset with 2,434 highly polar movie reviews, labelled negative or positive
- Paraphrase Detection: Quora Dataset with 400,000 question pairs with labels indicating whether particular instances are paraphrases of one another.
- Semantic Textual Similarity: SemEval STS Benchmark Dataset with 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning)

5.2 Evaluation method

For evaluating task performance, we use mean accuracy on class labels for evaluating the Sentiment Analysis and Paraphrase Detection, and Pearson correlation of the true similarity values against the predicted similarity values for Semantic Textual Similarity, as described in the Default project handout.

5.3 Experimental details

We use the pretrained Bert base model to initialize parameters, and fine-tune Bert model variants on three downstream NLP tasks, on a single GPU for 10 epochs each. We report accuracy on each of these tasks, and parameter efficiency in terms of number of parameters updated and number of parameters stored.

5.4 Results

We report the accuracy on eval (dev) dataset achieved from various model variations along with needed parameter updates per task and % of extra task specific parameters needed to be stored across three NLP tasks. Table 1 reports the accuracy on the two Sentiment Analysis tasks using the given single task classifier using the our implementation of Bert model, with or without fine-tuning, for SST and CFIMDB data sets. Table 2 reports the performance of base pretrain and fully fine-tuned single and multi-task models for 3 NLP tasks of Sentiment (SST), Paraphrase and Similarity Detection along with parameter efficiencies. Table 3 reports the performance of LoRA adaptation for various values of low rank (r) dimension, task-specific adapter modules for increasing values of adapter dimensions (a), as well as performance of combination of LoRA and Adapters. Table 3 also reports the performance of Hypernetworks-based adapter for various values of adapter hidden size dimensions (a) with fixed task embedding size (t=64). We show the fully fine-tuned multitask model performance for reference with these efficient adaptation methods. Additionally we study and report relationship of dev accuracy and training time to parameter updates, for various PEFT (parameter efficient fine-tuning) methods implemented, in Figure 1 and Figure 2 respectively.

In the end, we report our Test Leaderboard submission, where we submitted the Adapter based multitask model with dimension=256, and the LoRA+Adapter model, each with dimension 128. As of this writing we were placed 10^{th} in the Test Leaderboard with our LoRA+Adapter multitask efficient adaptation model.

Table 1: Single task Bert classifier dev accuracy on Seniment Analysis

Method	Type	SST	CFIMDB
Pretrain	Single-task	0.402	0.759
Finetune	Single-task	0.538	0.971

Table 2: Bert pretrained and fully fine-tuned models single and multi-task dev performance

Method	Type	Accuracy Average	Sentiment (SST)	Paraphrase	Similarity	Parameters updated per task	Additional Parameters stored
Pretrain	Single-task	0.416	0.304	0.665	0.183	0%	0%
Pretrain	Multi-task	0.384	0.402	0.665	0.182	0%	0%
Finetune	Single-task	0.753	**0.538**	**0.886**	0.869	100%	+200%
Finetune	Multi-task	**0.764**	0.506	0.879	**0.875**	100%	0%

6 Analysis

6.1 Parameter efficient Fine-tuning (PEFT)

- Pretrain only models have the lowest parameter overhead in terms of both % parameters updated per task ($\approx 0\%$) and total extra parameters stored across all three tasks (0%), as shown in Table 2; since all Bert pretrained parameters are frozen, and only the final classification layers are trained, and all parameters in pretrained model are shared across three tasks with no extra parameters added. However, these also have the lowest accuracy at about 0.4.
- Fully fine-tuned models, also shown in Table 2 are the most accurate but also most expensive in terms of % parameters updated per task (100%) and extra parameters stored, which are

Table 3: Parameter Efficient Fine-Tuning: Dev Accuracy

Method	Type	Accuracy Average	Sentiment (SST)	Paraphrase	Similarity	Parameters updated per task	Additional Parameters stored
LoRA[r=8]	Multi-task	0.676	0.426	0.747	0.854	0.40%	0.40%
LoRA[r=16]	Multi-task	0.681	0.404	0.782	0.857	0.81%	0.81%
LoRA[r=16]-20 epochs	Multi-task	0.710	0.447	0.830	0.852	0.81%	0.81%
LoRA[r=64]-20 epochs	Multi-task	0.703	0.400	0.851	0.858	3.23%	3.23%
Adapter[a=8]	Multi-task	0.684	0.416	0.855	0.782	0.72%	2.15%
Adapter[a=16]	Multi-task	0.712	0.456	0.860	0.821	1.39%	4.16%
Adapter[a=32]	Multi-task	0.730	0.495	0.864	0.831	2.74%	8.20%
Adapter[a=64]	Multi-task	0.741	0.501	0.873	0.850	5.42%	16.26%
Adapter[a=128]	Multi-task	0.746	0.509	0.876	0.854	10.8%	32.4%
Adapter[a=256]	Multi-task	0.752	**0.523**	0.877	0.857	21.56%	64.67%
LoRA+Adapter [r=8, a=64]	Multi-task	0.741	0.496	0.877	0.849	5.83%	16.67%
LoRA+Adapter [r=16, a=128]	Multi-task	0.749	0.509	0.876	0.862	11.61%	33.20%
LoRA+Adapter [r=128, a=128]	Multi-task	0.753	0.512	**0.879**	0.867	17.25%	38.85%
Hypernet[a=8]	Multi-task	0.683	0.421	0.822	0.805	1.49%	4.49%
Hypernet[a=16]	Multi-task	0.699	0.454	0.822	0.822	2.7%	8.1%
Hypernet[a=32]	Multi-task	0.709	0.471	0.835	0.821	5.1%	15.32%
Hypernet[a=64]	Multi-task	0.715	0.463	0.849	0.834	9.92%	29.76%
Hypernet[a=128]	Multi-task	0.712	0.457	0.841	0.837	19.54%	58.63%
Finetune	Multi-task	**0.764**	0.506	0.879	**0.875**	**100**%	0%

Table 4: Test Leaderboard submission accuracy

Test Set	Type	Test Accuracy	Sentiment (SST)	Paraphrase	Similarity	Parameters updated per task	Additional Parameters stored
Adapter[a=256]	Multi-task	0.776	0.519	0.877	0.859	21.56%	39.32%
LoRA+Adapter [r=128, a=128]	Multi-task	**0.778**	**0.525**	**0.879**	**0.862**	17.25%	38.85%

0% for a multitask fully fine-tuned model, but 3x (or +200%) for per-task separate fully fine-tuned models. The best overall accuracy here is 0.764.

- We observe a direct correlation of accuracy on the dev set to #parameters updated across fine-tuning methods as shown in Figure 1, across different methods of parameter efficient fine-tuning.
- We also observe a direct correlation between training time to #parameters updated during fine-tuning as shown in Figure 2. However, as shown in the figure, we are unable to get a training time below 5.5 minutes (roughly half of the full fine-tuning time for our GPU) even with training very less parameters like 0.8%. This could be because of the fixed memory read/write bottlenecks instead of compute bottlenecks.
- We are able to achieve accuracy of 0.749, very close to multi-task fully fine-tuned model, with LoRA+Adapter combination at 11.6% parameter updates per task, as shown in Table 3.

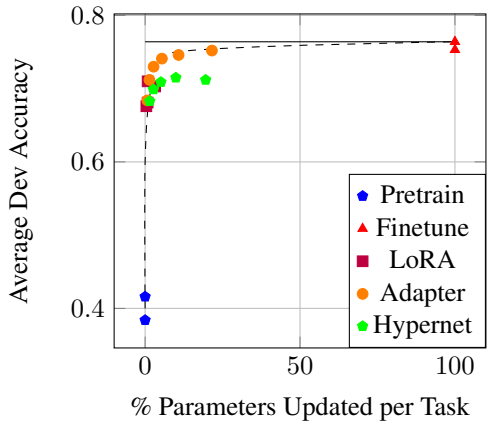

Figure 1: Dev Accuracy

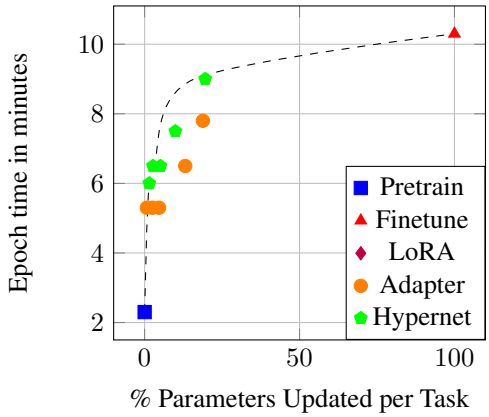

Figure 2: Training efficiency

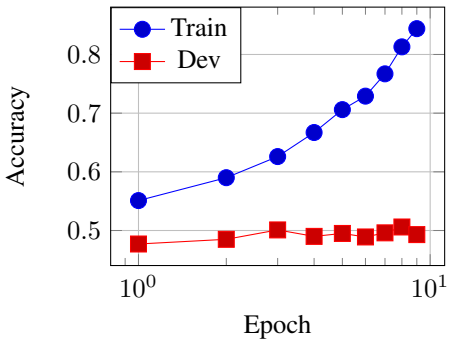

Figure 3: SST Accuracy over epochs

Figure 4: SST accuracy by Learning Rates

And even exceed performance of SST task at adapter dimension 256 with 21.6% parameter updates per task.

- Adpaters, as shown in Table 3 seem to scale more linearly to adding more parameters than LoRA and Hypernetworks. This could be because every parameter added for adapters is task specific, while LoRA and Hypernetworks share parameters and rely on transfer learning between these downstream tasks, which seems to generally be difficult here.
- Shared LoRA attention weights by itself does not seem to perform very well for Sentiment Analysis, as shown in Table 4, but is very parameter efficient, and is able to achieve 93% of overall accuracy compared to full fine-tuning only updating 0.8% of total parameters!
- LoRA+Adapter combination gives the best performance at better parameter efficiency than Adapter alone, likely due to combining benefits of task inference and isolation, at 0.753 overall accuracy and 17% parameter updates.

6.2 Hyperparameter tuning for SST

- Learning rate: We observe that the current learning rate of $1e^{-5}$ is pretty much the optimal learning rate and the model does not train well below or above this learning rate. $2e^{-}5$ is slightly better but not by a lot, as shown in Figure 4
- Classification layer Dropout and regularization: Since the model seems to be over-fitting on training data, we try raytune to tune the classification layer dropout and weight decay of AdamW optimizer by dev accuracy. Raytune arrives at a dropout prob of 0.223 and a weight decay of 0.445. We do a multi-task run with these values, but the accuracy across tasks does not go up significantly.

7 Conclusion & Future work

We are able to achieve similar overall, and even slightly better accuracy on some tasks, with multitask parameter efficient fine-tuning compared to fully fine-tuned model. Compared to 0.764 overall accuracy of our best fully fine-tuned multi-task model, we achieved overall accuracy of 0.753 (-0.011 or -1.4%) at 17% parameter updates with LoRA+Adapter combination. And 0.752 (diff of -0.012 or -1.5%) at 21.5% parameter updates with Adapter. A few other learnings and conclusions:

- Given we were trying to train a multi-task model with three tasks only, these tasks seemed to benefit less from transfer learning from each other but were more influenced by negative inference. Hence, techniques that reduced negative inference, for example, task-specific adapters performed better than techniques that relied on shared parameters and transfer learning like Hypernetworks.
- Our LoRA+Adapter combination of shared low rank adaptation of attention weights from LoRA, and task specific Adapters after attention and feed-forward layers worked well in getting same accuracy with less parameters, indicating that combining these parameter efficient techniques could result in benefits of task specific and cross-task transfer learnings.
- Sentiment Analysis was especially difficult task to improve upon. We explored hyperparameter tuning along multiple dimensions, however, this task seemed to be limited by labelled data a bit more than others. And likely additional pretraining with relevant datasets could have helped. We did try to add CFIMDB but that seemed to hurt SST performance even further in multitask environment. Likely because of negative inference from more polar reviews.

As future work, it would be interesting to see if the accuracy of SST can be improved by additional pretraining. And how these techniques would fare differently if there was lot more than 3 tasks. For example if these techniques were used for 10 or 50 or 100 tasks, would then something like hypernetworks would show much more advantage by enabling transfer learning across these downstream tasks and also parameter savings.

References

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Quoc V. Le David Ha, Andrew Dai. 2017. Hypernetworks.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, Bryan Catanzaro, and William J. Dally. 2017. Dsd: Dense-sparse-dense training for deep neural networks.

Peter Hase, Sebastian Schuster, Christopher Eichstaedt, and Iryna Gurevych. 2021. Zero-shot prompt design for text classification. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks.

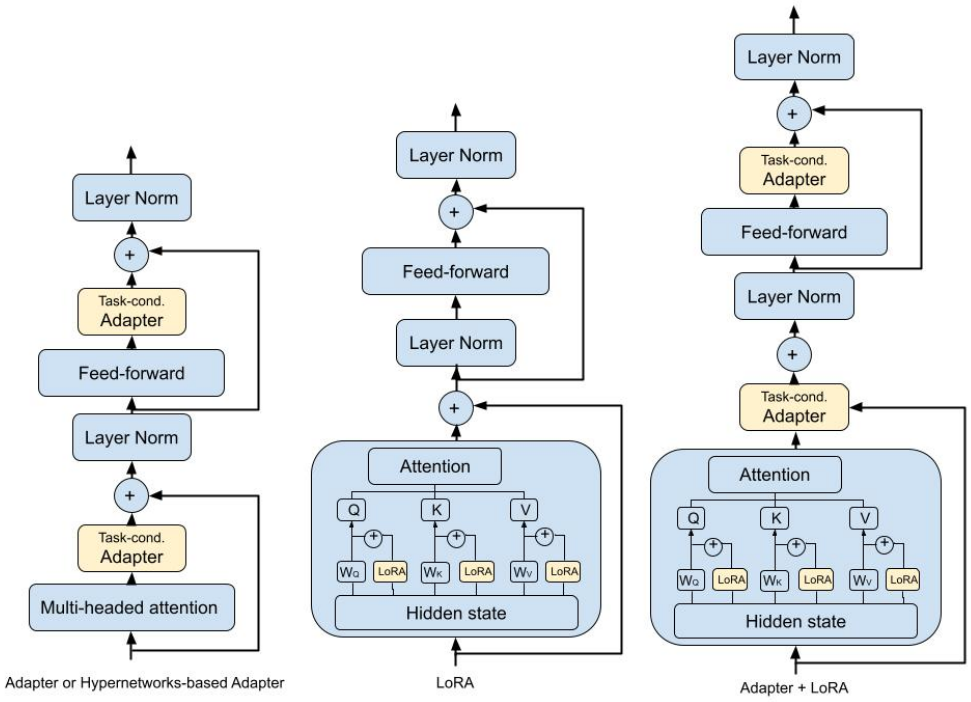

Figure 5: Variants: Adapter, LoRA, Adapter+LoRA

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pretraining. *URL: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language*$_u nderstanding_p aper.pdf$.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, and et al. 2019. Language models are unsupervised multitask learners. *arXiv preprint arXiv:1901.02860*.

A Appendix (optional)

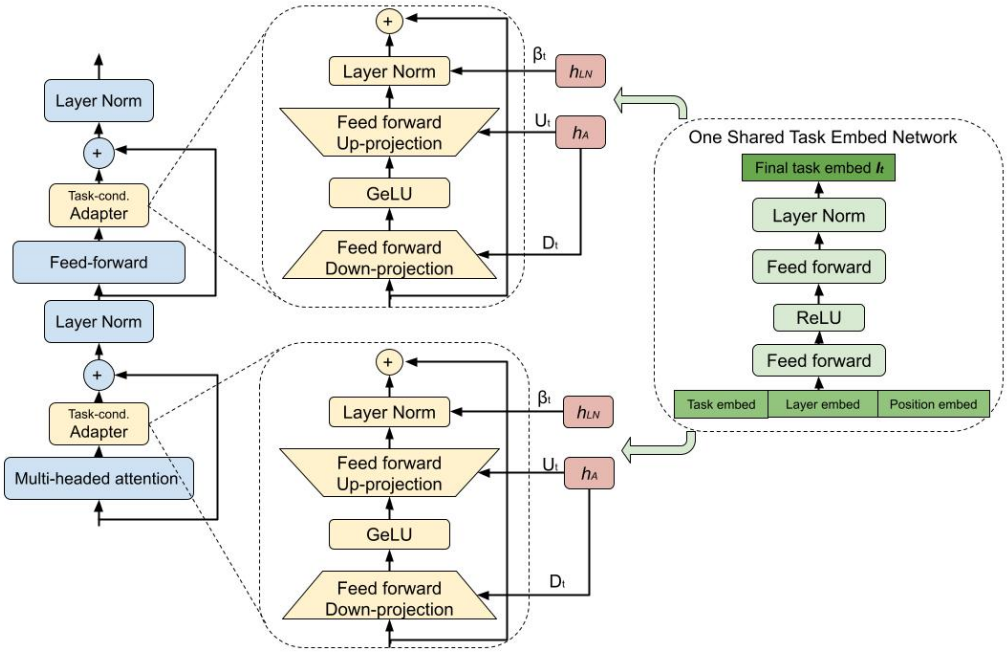

Figure 6: Hypernetworks-based Adapter