Total Time Consumption in Generating Answers: 85.97 seconds

Total Time Consumption in Evaluating Answers: 119.66 seconds

LLM Evaluation Summary

Evaluation 1:

Evaluation of Answer Quality:

Relevance: Excellent

The answer directly addresses the question by explaining what a GPT pre-trained model is, including its architecture, training process, and applications.

Faithfulness: Excellent

The answer is strictly supported by the given context, accurately describing GPT models, their transformer architecture, and the concept of generative pretraining.

Completeness: Excellent

The answer covers all major aspects of the question, including the definition, training method, and applications of GPT pre-trained models.

Clarity: Excellent

The answer is well-structured, clear, and easy to understand, with logical flow and appropriate terminology.

Conciseness: Excellent

The answer avoids unnecessary repetition and verbosity while providing comprehensive information.

Weighted Score: 100.0

Question 1: What is a GPT pre-trained model?

Answer 1: A GPT pre-trained model is a type of large language model (LLM) built using the transformer deep learning architecture. It is trained using a method called **generative pretraining**, which is a form of unsupervised learning. In this process, the model is exposed to vast amounts of unlabeled text data and learns to detect patterns, relationships, and structures within the data. This pretraining enables the model to understand and generate natural language effectively.

The pre-trained model can then be fine-tuned on specific tasks, such as text generation, translation, summarization, or question answering, making it highly versatile for applications in natural language processing (NLP). Examples of such models include GPT-3 and GPT-4, which are widely used in tools like chatbots, virtual assistants, and content generation platforms.

Response Latency: 5.88 seconds

Evaluation Latency: 4.35 seconds

[Prompt used for Q1]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

[Web] A generative pre-trained transformer (GPT) is <strong>a type of large language model (LLM) and a prominent framework for generative artificial intelligence</strong>. It is an artificial neural network that is used in natural language processing. It is based on the transformer deep learning architecture, pre-trained ...

[Web] Generative pretraining is <strong>a form of unsupervised learning, where the model is fed unlabeled data and forced to make sense of it on its own</strong>. By learning to detect patterns in unlabeled datasets, machine learning models gain the ability to draw similar conclusions when exposed to new inputs, ...

[Web] How was GPT-3 trained? What are examples of some applications that use GPT? How can AWS help you run large language models like GPT-3? Generative Pre-trained Transformers, commonly known as GPT, are <strong>a family of neural network models that uses the transformer architecture</strong> and is a key advancement ...

Current Question:

What is a GPT pre-trained model?

Answer:

Evaluation 2:

Relevance: Excellent

The answer directly addresses the question by listing and explaining the advantages of pre-trained models compared to others.

Faithfulness: Excellent

The answer is strictly supported by the given context, referencing the benefits of large-scale pre-trained models like BERT and GPT.

Completeness: Excellent

The answer comprehensively covers multiple key advantages, including reduced training time, generalization ability, transfer learning, state-of-the-art performance, ease of integration, cost-effectiveness, community support, and versatility.

Clarity: Excellent

The answer is well-structured, logically organized into numbered points, and easy to understand.

Conciseness: Excellent

The answer is concise and avoids unnecessary repetition or verbosity, presenting the information clearly and efficiently.

Weighted Score: 100.0

Question 2: What advantages does it have compared with other pre-trained models?

Answer 2: Pre-trained models, such as large-scale pre-trained language models (PTMs) like BERT and GPT, offer distinct advantages compared with other pre-trained models. Here are some key advantages:

1. **Reduced Training Time and Computational Cost**: Pre-trained models come with weights and parameters already optimized from prior training on large datasets. This eliminates the need to train a model from scratch, saving significant time and

computational resources.

*2. **Generalization Ability**: Pre-trained models, especially large-scale ones, often generalize better across tasks because they are trained on diverse and extensive datasets. This makes them suitable for a wide range of applications, even without substantial fine-tuning.*

*3. **Transfer Learning**: These models are highly effective for transfer learning. You can fine-tune them on specific tasks with relatively small datasets, leveraging their pre-existing knowledge to achieve high performance.*

*4. **State-of-the-Art Performance**: Models like GPT and BERT have set benchmarks in many natural language processing (NLP) tasks, such as text classification, sentiment analysis, question answering, and more.*

*5. **Ease of Integration**: Pre-trained models are readily available through platforms like Hugging Face, OpenAI, and others. This makes it easier for developers to integrate advanced AI capabilities into their products without needing deep expertise in model architecture and training.*

*6. **Cost-Effectiveness**: By using pre-trained models, companies can avoid the costs associated with curating large datasets, training models, and maintaining infrastructure, making them a more economical option.*

*7. **Community Support and Updates**: Popular pre-trained models benefit from active developer communities and continuous updates, ensuring access to improvements, bug fixes, and new features.*

*8. **Versatility Across Domains**: Many pre-trained models can be adapted to tasks beyond their original training domain. For example, GPT models can be fine-tuned for creative writing, customer service, or technical documentation.*

These advantages make pre-trained models a powerful choice for incorporating AI into products and workflows compared to other pre-trained options.

Response Latency: 6.44 seconds

Evaluation Latency: 6.94 seconds

[Prompt used for Q2]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

[Web] Adding language AI to your product? Let&#x27;s explore whether it makes more sense to build it in-house or use a <strong>pre</strong>-<strong>trained</strong> LLM.

[Web] Large-scale <strong>pre</strong>-<strong>trained</strong> <strong>models</strong> (PTMs) such as BERT and GPT have recently achieved great success and become a milestone in the field of artificial inte?

[Web] Answer (1 of 2): <strong>Trained</strong> <strong>and</strong> <strong>pre</strong>-<strong>trained</strong> is usually the same thing. It?s just a <strong>model</strong> that was already <strong>trained</strong> and has calculated weights with it. Sometimes people also share weights of particular layers, usually first ones, as they generalize better. Main <strong>advantage</strong> - you don?t need to ...

Current Question:

What advantages does it have compared with other pre-trained models?

Answer:

Evaluation 3:

### Evaluation:

Relevance: Excellent

The answer directly addresses the question by enumerating and explaining the key advantages of LoRA, closely aligning with the context provided.

Faithfulness: Good

The answer is largely faithful to the context, but it introduces details (e.g., specific examples like GPT-3 175B training efficiency and checkpoint sizes) that are not explicitly mentioned in the provided context. While these details are accurate and relevant, they go beyond the given information.

Completeness: Excellent

The answer is comprehensive and covers all the major advantages mentioned in the context, such as parameter efficiency, reduced hardware requirements, no inference latency, task modularity, orthogonality, and efficiency in training and storage.

Clarity: Excellent

The answer is well-structured and easy to understand, with clearly labeled points and concise explanations.

Conciseness: Good

While the answer is informative, it includes additional examples and elaborations that, while helpful, might make it slightly more verbose than necessary for a concise response.

Weighted Score: 94.0

Question 3: what's the key advantages of LoRA?

Answer 3: The key advantages of **Low-Rank Adaptation (LoRA)** are as follows:

1. **Parameter Efficiency**: LoRA reduces the number of trainable parameters by optimizing low-rank matrices during adaptation while keeping the pre-trained model weights frozen. This significantly lowers the memory and storage requirements for training.

2. **Reduced Hardware Barrier**: Since LoRA freezes most of the pre-trained model parameters and trains only smaller low-rank matrices, it requires less computational resources. For example, LoRA reduces VRAM consumption during training on large models like GPT-3 175B, enabling training with fewer GPUs.

3. **No Inference Latency**: LoRA can merge the trainable low-rank matrices with the frozen weights during deployment, ensuring no additional inference latency compared to fully fine-tuned models.

4. **Task Modularity**: LoRA allows pre-trained models to share their weights across multiple tasks. By switching the smaller LoRA modules (matrices A and B), tasks can be changed efficiently without requiring full fine-tuning or reloading large models, reducing task-switching overhead.

5. **Storage Efficiency**: LoRA drastically decreases checkpoint sizes. For instance, fine-tuning GPT-3 175B requires 350GB per checkpoint, while LoRA reduces this to just 35MB per task-specific LoRA module, making it ideal for multi-task setups.

6. **Training Speed**: LoRA speeds up training by focusing on fewer parameters (the low-rank matrices) instead of the entire model. In GPT-3 175B, LoRA achieves up to a 25% speedup in training throughput compared to full fine-tuning.

7. **Orthogonality to Other Methods**: LoRA is compatible with other adaptation techniques, such as prefix-tuning, making it versatile for combining methods to improve performance on downstream tasks.

8. **Scalability**: LoRA performs well on models of varying sizes, including extremely large-scale ones like GPT-3 175B, while maintaining strong performance across tasks.

*9. **Sample Efficiency**: LoRA exhibits high sample efficiency in low-data regimes, outperforming full fine-tuning and prefix-based methods when training on small datasets.*

*10. **Preservation of Pre-trained Knowledge**: By freezing the majority of model parameters, LoRA ensures that pre-trained knowledge is preserved while adapting the model to new tasks, reducing the risk of catastrophic forgetting.*

These advantages make LoRA a highly efficient and practical approach for adapting large pre-trained models to diverse downstream tasks while minimizing computational and storage costs.

Response Latency: 10.87 seconds

Evaluation Latency: 5.05 seconds

[Prompt used for Q3]

You are a helpful assistant. Use the following history and context to answer.

History:

Q: What is a GPT pre-trained model?

*A: A GPT pre-trained model is a type of large language model (LLM) built using the transformer deep learning architecture. It is trained using a method called **generative pretraining**, which is a form of unsupervised learning. In this process, the model is exposed to vast amounts of unlabeled text data and learns to detect patterns, relationships, and structures within the data. This pretraining enables the model to understand and generate natural language effectively.*

The pre-trained model can then be fine-tuned on specific tasks, such as text generation, translation, summarization, or question answering, making it highly versatile for applications in natural language processing (NLP). Examples of such models include GPT-3 and GPT-4, which are widely used in tools like chatbots, virtual assistants, and content generation platforms.

Q: What advantages does it have compared with other pre-trained models?

A: Pre-trained models, such as large-scale pre-trained language models (PTMs) like BERT and GPT, offer distinct advantages compared with other pre-trained models. Here are some key advantages:

*1. **Reduced Training Time and Computational Cost**: Pre-trained models come with weights and parameters already optimized* *from prior training on large datasets. This eliminates the need to train a model from scratch, saving significant time and computational resources.*

*2. **Generalization Ability**: Pre-trained models, especially large-scale ones, often generalize better across tasks because they are trained on diverse and extensive datasets. This makes them suitable for a wide range of applications, even without substantial fine-tuning.*

*3. **Transfer Learning**: These models are highly effective for transfer learning. You can fine-tune them on specific tasks with relatively small datasets, leveraging their pre-existing knowledge to achieve high performance.*

*4. **State-of-the-Art Performance**: Models like GPT and BERT have set benchmarks in many natural language processing (NLP) tasks, such as text classification, sentiment analysis, question answering, and more.*

*5. **Ease of Integration**: Pre-trained models are readily available through platforms like Hugging Face, OpenAI, and others. This makes it easier for developers to integrate advanced AI capabilities into their products without needing deep expertise in model architecture and training.*

*6. **Cost-Effectiveness**: By using pre-trained models, companies can avoid the costs associated with curating large datasets, training models, and maintaining infrastructure, making them a more economical option.*

*7. **Community Support and Updates**: Popular pre-trained models benefit from active developer communities and continuous updates, ensuring access to improvements, bug fixes, and new features.*

*8. **Versatility Across Domains**: Many pre-trained models can be adapted to tasks beyond their original training domain. For example, GPT models can be fine-tuned for creative writing, customer service, or technical documentation.*

These advantages make pre-trained models a powerful choice for incorporating AI into products and workflows compared to other pre-trained options.

Context:

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.06

LR Schedule

Linear

RoBERTa base

LoRA

Batch Size

16

16

16

32

32

16

32

16

# Epochs

30

60

30

80

25

25

80

40

Learning Rate

5E-04

5E-04

4E-04

4E-04

4E-04

5E-04

5E-04

4E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

512

RoBERTa large

LoRA

Batch Size

4

4

4

4

4

4

8

8

# Epochs

10

10

20

20

10

20

20

30

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

128

512

128

512

512

512

512

RoBERTa large

LoRA?

Batch Size

4

# Epochs

10

10

20

20

10

20

20

10

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

RoBERTa large

AdptP (3M)?

Batch Size

32

# Epochs

10

20

20

20

10

20

20

20

Learning Rate

3E-05

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptP (0.8M)?

Batch Size

32

# Epochs

5

20

20

20

10

20

20

20

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

16

Max Seq. Len.

128

RoBERTa large

AdptH (6M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptH (0.8M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

8

Max Seq. Len.

128

Table 9: The hyperparameters we used for RoBERTa on the GLUE benchmark.

D.3

GPT-2

We train all of our GPT-2 models using AdamW (Loshchilov & Hutter, 2017) with a linear learning rate schedule for 5 epochs. We use the batch size, learning rate, and beam search beam size described in Li & Liang (2021). Accordingly, we also tune the above hyperparameters for LoRA. We report the mean over 3 random seeds; the result for each run is taken from the best epoch. The hyperparameters used for LoRA in GPT-2 are listed in Table 11. For those used for other baselines, see Li & Liang (2021).

D.4

GPT-3

For all GPT-3 experiments, we train using AdamW (Loshchilov & Hutter, 2017) for 2 epochs with

a batch size of 128 samples and a weight decay factor of 0.1. We use a sequence length of 384 for

often introduce inference latency (Houlsby et al., 2019; Rebuf?et al., 2017) by extending model

depth or reduce the model?s usable sequence length (Li & Liang, 2021; Lester et al., 2021; Ham-

bardzumyan et al., 2020; Liu et al., 2021) (Section 3). More importantly, these method often fail to

match the ?ne-tuning baselines, posing a trade-off between ef?ciency and model quality.

We take inspiration from Li et al. (2018a); Aghajanyan et al. (2020) which show that the learned

over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the

change in weights during model adaptation also has a low ?intrinsic rank?, leading to our proposed

Low-Rank Adaptation (LoRA) approach. LoRA allows us to train some dense layers in a neural

network indirectly by optimizing rank decomposition matrices of the dense layers? change during

adaptation instead, while keeping the pre-trained weights frozen, as shown in Figure 1. Using GPT-3

175B as an example, we show that a very low rank (i.e., r in Figure 1 can be one or two) suf?ces even

when the full rank (i.e., d) is as high as 12,288, making LoRA both storage- and compute-ef?cient.

LoRA possesses several key advantages.

? A pre-trained model can be shared and used to build many small LoRA modules for dif-

ferent tasks. We can freeze the shared model and ef?ciently switch tasks by replacing the

matrices A and B in Figure 1, reducing the storage requirement and task-switching over-

head signi?cantly.

? LoRA makes training more ef?cient and lowers the hardware barrier to entry by up to 3

times when using adaptive optimizers since we do not need to calculate the gradients or

maintain the optimizer states for most parameters. Instead, we only optimize the injected,

much smaller low-rank matrices.

? Our simple linear design allows us to merge the trainable matrices with the frozen weights

when deployed, introducing no inference latency compared to a fully ?ne-tuned model, by

construction.

? LoRA is orthogonal to many prior methods and can be combined with many of them, such

as pre?x-tuning. We provide an example in Appendix E.

Terminologies and Conventions

We make frequent references to the Transformer architecture *and use the conventional terminologies for its dimensions.*

We call the input and output di-

mension size of a Transformer layer dmodel.

We use Wq, Wk, Wv, and Wo to refer to the

query/key/value/output projection matrices in the self-attention module. W or W0 refers to a pre-

trained weight matrix and ?W its accumulated gradient update during adaptation. We use r to

denote the rank of a LoRA module. We follow the conventions set out by (Vaswani et al., 2017;

Brown et al., 2020) and use Adam (Loshchilov & Hutter, 2019; Kingma & Ba, 2017) for model

optimization and use a Transformer MLP feedforward dimension dffn = 4 × dmodel.

2

PROBLEM STATEMENT

While our proposal is agnostic to training objective, we focus on language modeling as our motivat-

ing use case. Below is a brief description of the language modeling problem and, in particular, the

maximization of conditional probabilities given a task-speci?c prompt.

Suppose we are given a pre-trained autoregressive language model P?(y|x) parametrized by ?.

For instance, P?(y|x) can be a generic multi-task learner such as GPT (Radford et al., b; Brown

et al., 2020) based on the Transformer architecture (Vaswani et al., 2017). Consider adapting this

pre-trained model to downstream conditional text generation tasks, such as summarization, machine

reading comprehension (MRC), and natural language to SQL (NL2SQL). Each downstream task is

represented by a training dataset of context-target pairs: Z = {(xi, yi)}i=1,..,N, where both xi and

yi are sequences of tokens. For example, in NL2SQL, xi is a natural language query and yi its

corresponding SQL command; for summarization, xi is the content of an article and yi its summary.

2

guarantees that we do not introduce any additional latency during inference compared to a ?ne-tuned

model by construction.

4.2

APPLYING LORA TO TRANSFORMER

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the

number of trainable parameters. In the Transformer architecture, there are four weight matrices in

the self-attention module (Wq, Wk, Wv, Wo) and two in the MLP module. We treat Wq (or Wk, Wv)

as a single matrix of dimension dmodel × dmodel, even though the output dimension is usually sliced

into attention heads. We limit our study to only adapting the attention weights for downstream

tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity *and parameter-ef?ciency.We further study the effect on adapting different types of attention weight matrices in a Transformer in Section 7.1. We leave the empirical investigation of adapting the MLP layers, LayerNorm layers, and biases to a future work.*

Practical Bene?ts and Limitations.

The most signi?cant bene?t comes from the reduction in

memory and storage usage. For a large Transformer trained with Adam, we reduce that VRAM usage by up to 2/3 if r ?dmodel as we do not need to store the optimizer states for the frozen parameters. On GPT-3 175B, we reduce the VRAM consumption during training from 1.2TB to 350GB. With r = 4 and only the query and value projection matrices being adapted, the checkpoint size is reduced by roughly 10,000× (from 350GB to 35MB)4. This allows us to train with signi?-cantly fewer GPUs and avoid I/O bottlenecks. Another bene?t is that we can switch between tasks while deployed at a much lower cost by only swapping the LoRA weights as opposed to all the parameters. This allows for the creation of many customized models that can be swapped in and out on the ?y on machines that store the pre-trained weights in VRAM. We also observe a 25% speedup during training on GPT-3 175B compared to full ?ne-tuning5 as we do not need to calculate the gradient for the vast majority of the parameters.

LoRA also has its limitations. For example, it is not straightforward to batch inputs to different tasks with different A and B in a single forward pass, if one chooses to absorb A and B into W to eliminate additional inference latency. Though it is possible to not merge the weights and dynamically choose the LoRA modules to use for samples in a batch for scenarios where latency is not critical.

5

EMPIRICAL EXPERIMENTS

We evaluate the downstream task performance of LoRA on RoBERTa (Liu et al., 2019), De-BERTa (He et al., 2021), and GPT-2 (Radford et al., b), before scaling up to GPT-3 175B (Brown et al., 2020). Our experiments cover a wide range of tasks, from natural language understanding (NLU) to generation (NLG). Speci?cally, we evaluate on the GLUE (Wang et al., 2019) benchmark for RoBERTa and DeBERTa. We follow the setup of Li & Liang (2021) on GPT-2 for a direct com-parison and add WikiSQL (Zhong et al., 2017) (NL to SQL queries) and SAMSum (Gliwa et al., 2019) (conversation summarization) for large-scale experiments on GPT-3. See Appendix C for more details on the datasets we use. We use NVIDIA Tesla V100 for all experiments.

5.1

BASELINES

To compare with other baselines broadly, we replicate the setups used by prior work and reuse their *reported numbers whenever possible. This, however, means that some baselines might only appear*

in certain experiments.

Fine-Tuning (FT) is a common approach for adaptation. During ?ne-tuning, the model is initialized

to the pre-trained weights and biases, and all model parameters undergo gradient updates.A simple

variant is to update only some layers while freezing others. We include one such baseline reported

in prior work (Li & Liang, 2021) on GPT-2, which adapts just the last two layers (FTTop2).

4We still need the 350GB model during deployment; however, storing 100 adapted models only requires

*350GB + 35MB * 100 ?354GB as opposed to 100 * 350GB ?35TB.*

5For GPT-3 175B, the training throughput for full ?ne-tuning is 32.5 tokens/s per V100 GPU; with the same

number of weight shards for model parallelism, the throughput is 43.1 tokens/s per V100 GPU for LoRA.

5

to maximize downstream performance? 2) Is the ?optimal? adaptation matrix ?W really rank-

de?cient? If so, what is a good rank to use in practice? 3) What is the connection between ?W and

W? Does ?W highly correlate with W? How large is ?W comparing to W?

We believe that our answers to question (2) and (3) shed light on the fundamental principles of using

pre-trained language models for downstream tasks, which is a critical topic in NLP.

7.1

WHICH WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Given a limited parameter budget, which types of weights should we adapt with LoRA to obtain

the best performance on downstream tasks? As mentioned in Section 4.2, we only consider weight

matrices in the self-attention module. We set a parameter budget of 18M (roughly 35MB if stored

in FP16) on GPT-3 175B, which corresponds to r = 8 if we adapt one type of attention weights or

r = 4 if we adapt two types, for all 96 layers. The result is presented in Table 5.

# of Trainable Parameters = 18M

Weight Type

Wq

Wk

Wv

Wo

Wq, Wk

Wq, Wv

Wq, Wk, Wv, Wo

Rank r

8

8

8

8

4

4

2

WikiSQL (±0.5%)

70.4

70.0

73.0

73.2

71.4

73.7

73.7

MultiNLI (±0.1%)

91.0

90.8

91.0

91.3

91.3

91.3

91.7

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both Wq and Wv gives the best performance overall. We ?nd the standard deviation across random seeds to be consistent for a given dataset, which we report in the ?rst column.

Note that putting all the parameters in ?Wq or ?Wk results in signi?cantly lower performance, while adapting both Wq and Wv yields the best result. This suggests that even a rank of four captures enough information in ?W such that it is preferable to adapt more weight matrices than adapting a single type of weights with a larger rank.

7.2

WHAT IS THE OPTIMAL RANK r FOR LORA?

We turn our attention to the effect of rank r on model performance.

We adapt {Wq, Wv},

{Wq, Wk, Wv, Wc}, and just Wq for a comparison.

Weight Type

r = 1

r = 2

r = 4

r = 8

r = 64

WikiSQL(±0.5%)

Wq

68.8

69.6

70.5

70.4

70.0

Wq, Wv

73.4

73.3

73.7

73.8

73.5

Wq, Wk, Wv, Wo

74.1

73.7

74.0

74.0

73.9

MultiNLI (±0.1%)

Wq

90.7

90.9

91.1

90.7

90.7

Wq, Wv

91.3

91.4

91.3

91.6

91.4

Wq, Wk, Wv, Wo

91.2

91.7

91.7

91.5

91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a

rank as small as one suf?ces for adapting both Wq and Wv on these datasets while training Wq alone

needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

Table 6 shows that, surprisingly, LoRA already performs competitively with a very small r (more

so for {Wq, Wv} than just Wq). This suggests the update matrix ?W could have a very small

?intrinsic rank?.6 To further support this ?nding, we check the overlap of the subspaces learned by

different choices of r and by different random seeds. We argue that increasing r does not cover a

more meaningful subspace, which suggests that a low-rank adaptation matrix is suf?cient.

6However, we do not expect a small r to work for every task or dataset. Consider the following thought

experiment: if the downstream task were in a different language than the one used for pre-training, retraining

the entire model (similar to LoRA with r = dmodel) could certainly outperform LoRA with a small r.

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.1

LR Schedule

Linear

DeBERTa XXL

LoRA

Batch Size

8

8

32

4

6

8

4

4

# Epochs

5

16

30

10

8

11

11

10

Learning Rate

1E-04

6E-05

2E-04

1E-04

1E-04

1E-04

2E-04

2E-04

Weight Decay

0

0.01

0.01

0

0.01

0.01

0.01

0.1

CLS Dropout

0.15

0

0

0.1

0.1

0.2

0.2

0.2

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

256

128

128

64

512

320

320

128

Table 10: The hyperparameters for DeBERTa XXL on tasks included in the GLUE benchmark.

Dataset

E2E

WebNLG

DART

Training

Optimizer

AdamW

Weight Decay

0.01

0.01

0.0

Dropout Prob

0.1

0.1

0.0

Batch Size

8

# Epoch

5

Warmup Steps

500

Learning Rate Schedule

Linear

Label Smooth

0.1

0.1

0.0

Learning Rate

0.0002

Adaptation

rq = rv = 4

LoRA ?

32

Inference

Beam Size

10

Length Penalty

0.9

0.8

0.8

no repeat ngram size

4

Table 11: The hyperparameters for GPT-2 LoRA on E2E, WebNLG and DART.

WikiSQL (Zhong et al., 2017), 768 for MNLI (Williams et al., 2018), and 2048 for SAMSum (Gliwa

et al., 2019). We tune learning rate for all method-dataset combinations. See Section D.4 for more

details on the hyperparameters used. For pre?x-embedding tuning, we ?nd the optimal lp and li

to be 256 and 8, respectively, totalling 3.2M trainable parameters. We use lp = 8 and li = 8 for

pre?x-layer tuning with 20.2M trainable parameters to obtain the overall best performance. We

present two parameter budgets for LoRA: 4.7M (rq = rv = 1 or rv = 2) and 37.7M (rq = rv = 8

or rq = rk = rv = ro = 2). We report the best validation performance from each run. The training

hyperparameters used in our GPT-3 experiments are listed in Table 12.

E

COMBINING LORA WITH PREFIX TUNING

LoRA can be naturally combined with existing pre?x-based approaches. In this section, we evaluate

two combinations of LoRA and variants of pre?x-tuning on WikiSQL and MNLI.

LoRA+Pre?xEmbed (LoRA+PE) combines LoRA with pre?x-embedding tuning, where we insert

lp + li special tokens whose embeddings are treated as trainable parameters. For more on pre?x-

embedding tuning, see Section 5.1.

LoRA+Pre?xLayer (LoRA+PL) combines LoRA with pre?x-layer tuning. We also insert lp + li

special tokens; however, instead of letting the hidden representations of these tokens evolve natu-

tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are *there more principled ways to do it? 4) Finally, the rank-de?ciency of ?W suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.*

REFERENCES

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. arXiv:2012.13255 [cs], December 2020. URL http://arxiv.org/abs/2012.13255.

Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Ef?ciently, Going Beyond Kernels? In NeurIPS, 2019. Full version available at http://arxiv.org/abs/1905.10337.

Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. arXiv preprint arXiv:2001.04413, 2020a.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature puri?cation: How adversarial training performs robust deep learning. arXiv preprint arXiv:2005.10190, 2020b.

*Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In ICML, 2019. Full version available at http://arxiv.org/abs/1811.
03962.*

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs], July 2020. URL http://arxiv.org/abs/2005.14165.

Jian-Feng Cai, Emmanuel J Cand`es, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. SIAM Journal on optimization, 20(4):1956?1982, 2010.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), 2017. doi: 10.18653/v1/s17-2001. URL http://dx.doi.org/10.18653/v1/S17-2001.

Ronan Collobert and Jason Weston. A uni?ed architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, ICML ?08, pp. 160?167, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL *https://doi.org/10.1145/1390156.1390177.*

Misha Denil, Babak Shakibi, Laurent Dinh, Marc?Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019a.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs], May 2019b. URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005), 2005. URL https://aclanthology.org/I05-5002.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg challenge: Generating text from rdf data. In Proceedings of the 10th International Conference on Natural Language Generation, pp. 124?133, 2017.

Model&Method	*# Trainable*	*WikiSQL*	*MNLI-m*	*SAMSum*
	Parameters	*Acc. (%)*	*Acc. (%)*	*R1/R2/RL*
GPT-3 (FT)	*175,255.8M*	*73.8*	*89.5*	*52.0/28.0/44.5*
GPT-3 (BitFit)	*14.2M*	*71.3*	*91.0*	

				51.3/27.4/43.5
GPT-3 (PreEmbed)	*3.2M*	*63.1*	*88.6*	*48.3/24.2/40.5*
GPT-3 (PreLayer)	*20.2M*	*70.1*	*89.5*	*50.8/27.3/43.5*
GPT-3 (AdapterH)	*7.1M*	*71.9*	*89.8*	*53.0/28.9/44.8*
GPT-3 (AdapterH)	*40.1M*	*73.2*	*91.5*	*53.2/29.0/45.1*
GPT-3 (LoRA)	*4.7M*	*73.4*	*91.7*	*53.8/29.8/45.9*
GPT-3 (LoRA)	*37.7M*	*74.0*	*91.6*	*53.4/29.2/45.1*

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on

SAMSum. LoRA performs better than prior approaches, including full ?ne-tuning. The results on WikiSQL have a ?uctuation around ±0.5%, MNLI-m around ±0.1%, and SAMSum around ±0.2/±0.2/±0.1 for the three metrics.

5.5

SCALING UP TO GPT-3 175B

As a ?nal stress test for LoRA, we scale up to GPT-3 with 175 billion parameters. Due to the high training cost, we only report the typical standard deviation for a given task over random seeds, as opposed to providing one for every entry. See Section D.4 for details on the hyperparameters used.

As shown in Table 4, LoRA matches or exceeds the ?ne-tuning baseline on all three datasets. Note that not all methods bene?t monotonically from having more trainable parameters, as shown in Figure 2. We observe a signi?cant performance drop when we use more than 256 special tokens for pre?x-embedding tuning or more than 32 special tokens for pre?x-layer tuning. This corroborates similar observations in Li & Liang (2021). While a thorough investigation into this phenomenon is out-of-scope for this work, we suspect that having more special tokens causes the input distribution to shift further away from the pre-training data distribution. Separately, we investigate the performance of different adaptation approaches in the low-data regime in Section F.3.

6

7

8

9

10

11

log10 # Trainable Parameters

0.55

0.60

0.65

0.70

0.75

Validation Accuracy

WikiSQL

Method

Fine-Tune

PrefixEmbed

PrefixLayer

Adapter(H)

LoRA

6

7

8

9

10

11

log10 # Trainable Parameters

0.84

0.86

0.88

0.90

0.92

MultiNLI-matched

Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation

methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance.

See Section F.2 for more details on the plotted data points.

6

RELATED WORKS

Transformer Language Models.

Transformer (Vaswani et al., 2017) is a sequence-to-sequence

architecture that makes heavy use of self-attention. Radford et al. (a) applied it to autoregressive lan-

guage modeling by using a stack of Transformer decoders. Since then, Transformer-based language

models have dominated NLP, achieving the state-of-the-art in many tasks. A new paradigm emerged

with BERT (Devlin et al., 2019b) and GPT-2 (Radford et al., b) ? both are large Transformer lan-

8

Hyperparameters

Fine-Tune

PreEmbed

PreLayer

BitFit

AdapterH

LoRA

Optimizer

AdamW

Batch Size

128

# Epoch

2

Warmup Tokens

250,000

LR Schedule

Linear

Learning Rate

5.00E-06

5.00E-04

1.00E-04

1.6E-03

1.00E-04

2.00E-04

Table 12: The training hyperparameters used for different GPT-3 adaption methods. We use the same hyperparameters for all datasets after tuning learning rate.

rally, we replace them after every Transformer block with an input agnostic vector. Thus, both the embeddings and subsequent Transformer block activations are treated as trainable parameters. For more on pre?x-layer tuning, see Section 5.1.

In Table 15, we show the evaluation results of LoRA+PE and LoRA+PL on WikiSQL and MultiNLI. First of all, LoRA+PE signi?cantly outperforms both LoRA and pre?x-embedding tuning on WikiSQL, which indicates that LoRA is somewhat orthogonal to pre?x-embedding tuning. On MultiNLI, the combination of LoRA+PE doesn?t perform better than LoRA, possibly because LoRA on its own already achieves performance comparable to the human baseline. Secondly, we notice that LoRA+PL performs slightly worse than LoRA even with more trainable parameters. We attribute this to the fact that pre?x-layer tuning is very sensitive to the choice of learning rate and thus makes the optimization of LoRA weights more dif?cult in LoRA+PL.

F

ADDITIONAL EMPIRICAL EXPERIMENTS

F.1

ADDITIONAL EXPERIMENTS ON GPT-2

We also repeat our experiment on DART (Nan et al., 2020) and WebNLG (Gardent et al., 2017)

following the setup of Li & Liang (2021). The result is shown in Table 13. Similar to our result

on E2E NLG Challenge, reported in Section 5, LoRA performs better than or at least on-par with

pre?x-based approaches given the same number of trainable parameters.

Method

# Trainable

DART

Parameters

BLEU?

MET?

TER?

GPT-2 Medium

Fine-Tune

354M

46.2

0.39

0.46

AdapterL

0.37M

42.4

0.36

0.48

AdapterL

11M

45.2

0.38

0.46

FTTop2

24M

41.0

0.34

0.56

PrefLayer

0.35M

46.4

0.38

0.46

LoRA

0.35M

47.1±.2

0.39

0.46

GPT-2 Large

Fine-Tune

774M

47.0

0.39

0.46

AdapterL

0.88M

45.7±.1

0.38

0.46

AdapterL

23M

47.1±.1

0.39

0.45

PrefLayer

0.77M

46.7

0.38

0.45

LoRA

0.77M

47.5±.1

0.39

0.45

Table 13: GPT-2 with different adaptation methods on DART. The variances of MET and TER are

less than 0.01 for all adaption approaches.

Method

WebNLG

BLEU?

MET?

TER?

U

S

A

U

S

A

U

S

A

GPT-2 Medium

Fine-Tune (354M)

27.7

64.2

46.5

.30

.45

.38

.76

.33

.53

AdapterL (0.37M)

45.1

54.5

50.2

.36

.39

.38

.46

.40

.43

AdapterL (11M)

48.3

60.4

54.9

.38

.43

.41

.45

.35

.39

FTTop2 (24M)

18.9

53.6

36.0

.23

.38

.31

.99

.49

.72

Pre?x (0.35M)

45.6

62.9

55.1

.38

.44

.41

.49

.35

.40

LoRA (0.35M)

46.7±.4

62.1±.2

55.3±.2

.38

.44

.41

.46

.33

.39

GPT-2 Large

Fine-Tune (774M)

43.1

65.3

55.5

.38

.46

.42

.53

.33

.42

AdapterL (0.88M)

49.8±.0

61.1±.0

56.0±.0

.38

.43

.41

.44

.35

.39

AdapterL (23M)

49.2±.1

64.7±.2

57.7±.1

.39

.46

.43

.46

.33

.39

Pre?x (0.77M)

47.7

63.4

56.3

.39

.45

.42

.48

.34

.40

LoRA (0.77M)

48.4±.3

64.0±.3

57.0±.1

.39

.45

.42

.45

.32

.38

Table 14: GPT-2 with different adaptation methods on WebNLG. The variances of MET and TER are less than 0.01 for all the experiments we ran. ?U? indicates unseen categories, ?S? indicates seen categories, and ?A? indicates all categories in the test set of WebNLG.

F.2

ADDITIONAL EXPERIMENTS ON GPT-3

We present additional runs on GPT-3 with different adaptation methods in Table 15. The focus is on identifying the trade-off between performance and the number of trainable parameters.

F.3

LOW-DATA REGIME

To evaluate the performance of different adaptation approaches in the low-data regime. we randomly sample 100, 1k and 10k training examples from the full training set of MNLI to form the low-data MNLI-n tasks. In Table 16, we show the performance of different adaptation approaches on MNLI-n. To our surprise, Pre?xEmbed and Pre?xLayer performs very poorly on MNLI-100 dataset, with Pre?xEmbed performing only slightly better than random chance (37.6% vs. 33.3%). Pre?xLayer performs better than Pre?xEmbed but is still signi?cantly worse than Fine-Tune or LoRA on MNLI-100. The gap between pre?x-based approaches and LoRA/Fine-tuning becomes smaller as we increase the number of training examples, which might suggest that pre?x-based approaches are not suitable for low-data tasks in GPT-3. LoRA achieves better performance than ?ne-tuning on both MNLI-100 and MNLI-Full, and comparable results on MNLI-1k and MNLI-10K considering the (±0.3) variance due to random seeds.

The training hyperparameters of different adaptation approaches on MNLI-n are reported in Table 17. We use a smaller learning rate for Pre?xLayer on the MNLI-100 set, as the training loss does not decrease with a larger learning rate.

G

MEASURING SIMILARITY BETWEEN SUBSPACES

In this paper we use the measure ?(A, B, i, j) = ?(U i

A, U j

B) = ?U i?

A UB?2

F

min{i,j}

to measure the subspace

similarity between two column orthonormal matrices U i

A ?Rd×i and U j

B ?Rd×j, obtained by

taking columns of the left singular matrices of A and B. We point out that this similarity is simply

a reverse of the standard Projection Metric that measures distance between subspaces Ham & Lee

(2008).

22

Method

Hyperparameters

# Trainable Parameters

WikiSQL

MNLI-m

Fine-Tune

-

175B

73.8

89.5

Pre?xEmbed

lp = 32, li = 8

0.4 M

55.9

84.9

lp = 64, li = 8

0.9 M

58.7

88.1

lp = 128, li = 8

1.7 M

60.6

88.0

lp = 256, li = 8

3.2 M

63.1

88.6

lp = 512, li = 8

6.4 M

55.9

85.8

Pre?xLayer

lp = 2, li = 2

5.1 M

68.5

89.2

lp = 8, li = 0

10.1 M

69.8

88.2

lp = 8, li = 8

20.2 M

70.1

89.5

lp = 32, li = 4

44.1 M

66.4

89.6

lp = 64, li = 0

76.1 M

64.9

87.9

AdapterH

r = 1

7.1 M

71.9

89.8

r = 4

21.2 M

73.2

91.0

r = 8

40.1 M

73.2

91.5

r = 16

77.9 M

73.2

91.5

r = 64

304.4 M

72.6

91.5

LoRA

rv = 2

4.7 M

73.4

91.7

rq = rv = 1

4.7 M

73.4

91.3

rq = rv = 2

9.4 M

73.3

91.4

rq = rk = rv = ro = 1

9.4 M

74.1

91.2

rq = rv = 4

18.8 M

73.7

91.3

rq = rk = rv = ro = 2

18.8 M

73.7

91.7

rq = rv = 8

37.7 M

73.8

91.6

rq = rk = rv = ro = 4

37.7 M

74.0

91.7

rq = rv = 64

301.9 M

73.6

91.4

rq = rk = rv = ro = 64

603.8 M

73.9

91.4

LoRA+PE

rq = rv = 8, lp = 8, li = 4

37.8 M

75.0

91.4

rq = rv = 32, lp = 8, li = 4

151.1 M

75.9

91.1

rq = rv = 64, lp = 8, li = 4

302.1 M

76.2

91.3

LoRA+PL

rq = rv = 8, lp = 8, li = 4

52.8 M

72.9

90.2

Table 15: Hyperparameter analysis of different adaptation approaches on WikiSQL and MNLI. Both pre?x-embedding tuning (Pre?xEmbed) and pre?x-layer tuning (Pre?xLayer) perform worse as we increase the number of trainable parameters, while LoRA?s performance stabilizes. Performance is measured in validation accuracy.

Method

MNLI(m)-100

MNLI(m)-1k

MNLI(m)-10k

MNLI(m)-392K

GPT-3 (Fine-Tune)

60.2

85.8

88.9

89.5

GPT-3 (Pre?xEmbed)

37.6

75.2

79.5

88.6

GPT-3 (Pre?xLayer)

48.3

82.5

85.9

89.6

GPT-3 (LoRA)

63.8

85.6

89.2

91.7

Table 16: Validation accuracy of different methods on subsets of MNLI using GPT-3 175B. MNLI-

n describes a subset with n training examples. We evaluate with the full validation set. LoRA

performs exhibits favorable sample-ef?ciency compared to other methods, including ?ne-tuning.

To be concrete, let the singular values of U i?

A U j

B to be ?1, ?2, · · · , ?p where p = min{i, j}. We

know that the Projection Metric Ham & Lee (2008) is de?ned as:

d(U i

A, U j

B) =

v

u

u

tp ?

p

X

i=1

?2

i ?[0, ?p]

23

Current Question:

what's the key advantages of LoRA?

Answer:

Evaluation 4:

Evaluation of the Answer:

Relevance: **Excellent**

The answer comprehensively addresses LoRA's practical benefits, advantages, and limitations as requested in the question. Each point directly relates to the context provided and covers the relevant aspects.

Faithfulness: **Excellent**

The answer remains strictly supported by the context. All claims regarding LoRA's benefits and limitations align accurately with the provided information, such as reductions in VRAM usage, task-switching efficiency, and batching constraints.

Completeness: **Excellent**

The answer thoroughly covers all major aspects of LoRA's benefits and limitations, including reduced memory usage, cost-effectiveness, scalability, batching constraints, and sensitivity to rank selection. No significant points are omitted.

Clarity: **Excellent**

The answer is well-structured with clear headings for benefits and limitations. Each point is concisely and logically explained, making it easy to understand.

Conciseness: **Good**

While the answer is detailed and informative, some points, particularly within the benefits section, could be slightly condensed to improve brevity without losing essential information. For example, the explanation of VRAM reduction and cost-effectiveness overlaps slightly.

Final Ratings:

- **Relevance:** Excellent

- **Faithfulness:** Excellent

*- **Completeness:** Excellent*

*- **Clarity:** Excellent*

*- **Conciseness:** Good*

Weighted Score: 98.0

Question 4: Lora's practical benefits, advantages and limitations.

*Answer 4: **LoRA's Practical Benefits and Advantages:***

*1. **Reduced Memory and Storage Usage:** LoRA significantly reduces memory and storage requirements during training and deployment. By freezing the pre-trained model weights and optimizing only the injected low-rank matrices, LoRA reduces VRAM consumption by up to 2/3 during training and reduces the checkpoint size by as much as 10,000× in large-scale models like GPT-3 175B.*

*2. **Cost-Effective Training:** LoRA lowers the hardware barrier for training large language models by reducing the number of trainable parameters, making it feasible for researchers and organizations with limited computational resources. For instance, LoRA reduces VRAM consumption from 1.2TB to 350GB for GPT-3 175B, enabling training with fewer GPUs.*

*3. **No Additional Inference Latency:** LoRA introduces no latency during inference compared to fully fine-tuned models. This is achieved by merging the low-rank matrices with the frozen weights during deployment, allowing LoRA to perform as efficiently as full fine-tuning.*

*4. **Task Switching Efficiency:** LoRA enables efficient task switching by swapping the small task-specific LoRA weights rather than retraining or reloading the entire model. This allows for quick adaptation to multiple tasks with minimal storage and computational overhead.*

5. **Parameter Efficiency:** LoRA achieves comparable or superior performance to full fine-tuning while drastically reducing the number of trainable parameters. For example, LoRA can reduce trainable parameters by 10,000× in GPT-3 175B while maintaining or improving model quality.

6. **Scalability Across Models:** LoRA can be applied to various Transformer-based architectures, including RoBERTa, DeBERTa, GPT-2, and GPT-3, making it highly versatile for both natural language understanding (NLU) and generation (NLG) tasks.

7. **Compatibility with Other Methods:** LoRA is orthogonal to other adaptation techniques, such as prefix-tuning, and can be combined with them for enhanced performance.

8. **Improved Training Throughput:** By freezing most parameters and only optimizing the injected low-rank matrices, LoRA speeds up training. For GPT-3 175B, LoRA achieves a 25% speedup in training throughput compared to full fine-tuning.

9. **Effective in Low-Data Regimes:** LoRA performs well in scenarios with limited training data, outperforming prefix-based methods in low-data tasks like MNLI-100, demonstrating its robustness in resource-constrained settings.

LoRA's Limitations:

1. **Batching Constraints for Task-Specific Modules:** In scenarios where task-specific matrices (A and B) are absorbed into the pre-trained weights to eliminate inference latency, batching inputs for different tasks in a single forward pass becomes challenging. This limitation may hinder efficiency in certain multi-task setups.

2. **Heuristic Weight Selection:** LoRA relies on heuristics to select which weight matrices to apply its low-rank adaptation. More principled methods for weight selection are an area for future improvement.

*3. **Sensitivity to Rank Selection:** While LoRA can perform well with very low ranks (e.g., r = 1 or 2), the effectiveness of the rank depends on the task and dataset. Determining the optimal rank for specific scenarios may require experimentation.*

*4. **Limited Support for Dynamic Inputs:** LoRA may not be ideal for tasks requiring dynamic input adaptation, as the injected matrices are fixed during training.*

*5. **Performance Variability in Combination Methods:** When combined with techniques like prefix-layer tuning, LoRA may experience optimization challenges, leading to slightly worse performance in certain cases (e.g., LoRA+PL on MultiNLI).*

*6. **Focus on Attention Weights:** LoRA primarily adapts attention weights in the Transformer architecture (e.g., Wq, Wk, Wv, Wo), leaving some layers (e.g., MLP modules, LayerNorm layers, biases) frozen. Adapting additional layers might improve performance for certain tasks, but remains unexplored.*

LoRA's benefits in reducing computational costs, enabling efficient task switching, and maintaining high performance make it a powerful tool for adapting large language models. Its limitations, while notable, are outweighed by its practical advantages in various NLP applications.

Response Latency: 14.05 seconds

Evaluation Latency: 6.55 seconds

[Prompt used for Q4]

You are a helpful assistant. Use the following history and context to answer.

History:

Q: What is a GPT pre-trained model?

A: A GPT pre-trained model is a type of large language model (LLM) built using the transformer deep learning architecture. It is *trained using a method called **generative pretraining**, which is a form of unsupervised learning. In this process, the model is exposed to vast amounts of unlabeled text data and learns to detect patterns, relationships, and structures within the data. This pretraining enables the model to understand and generate natural language effectively.*

The pre-trained model can then be fine-tuned on specific tasks, such as text generation, translation, summarization, or question answering, making it highly versatile for applications in natural language processing (NLP). Examples of such models include GPT-3 and GPT-4, which are widely used in tools like chatbots, virtual assistants, and content generation platforms.
Q: What advantages does it have compared with other pre-trained models?
A: Pre-trained models, such as large-scale pre-trained language models (PTMs) like BERT and GPT, offer distinct advantages compared with other pre-trained models. Here are some key advantages:

*1. **Reduced Training Time and Computational Cost**: Pre-trained models come with weights and parameters already optimized from prior training on large datasets. This eliminates the need to train a model from scratch, saving significant time and computational resources.*

*2. **Generalization Ability**: Pre-trained models, especially large-scale ones, often generalize better across tasks because they are trained on diverse and extensive datasets. This makes them suitable for a wide range of applications, even without substantial fine-tuning.*

*3. **Transfer Learning**: These models are highly effective for transfer learning. You can fine-tune them on specific tasks with relatively small datasets, leveraging their pre-existing knowledge to achieve high performance.*

*4. **State-of-the-Art Performance**: Models like GPT and BERT have set benchmarks in many natural language processing (NLP) tasks, such as text classification, sentiment analysis, question answering, and more.*

*5. **Ease of Integration**: Pre-trained models are readily available through platforms like Hugging Face, OpenAI, and others. This makes it easier for developers to integrate advanced AI capabilities into their products without needing deep expertise in model architecture and training.*

*6. **Cost-Effectiveness**: By using pre-trained models, companies can avoid the costs associated with curating large datasets, training models, and maintaining infrastructure, making them a more economical option.*

*7. **Community Support and Updates**: Popular pre-trained models benefit from active developer communities and continuous* *updates, ensuring access to improvements, bug fixes, and new features.*

*8. **Versatility Across Domains**: Many pre-trained models can be adapted to tasks beyond their original training domain. For example, GPT models can be fine-tuned for creative writing, customer service, or technical documentation.*

These advantages make pre-trained models a powerful choice for incorporating AI into products and workflows compared to other pre-trained options.

Context:

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.06

LR Schedule

Linear

RoBERTa base

LoRA

Batch Size

16

16

16

32

32

16

32

16

# Epochs

30

60

30

80

25

25

80

40

Learning Rate

5E-04

5E-04

4E-04

4E-04

4E-04

5E-04

5E-04

4E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

512

RoBERTa large

LoRA

Batch Size

4

4

4

4

4

4

8

8

# Epochs

10

10

20

20

10

20

20

30

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

128

512

128

512

512

512

512

RoBERTa large

LoRA?

Batch Size

4

# Epochs

10

10

20

20

10

20

20

10

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

RoBERTa large

AdptP (3M)?

Batch Size

32

# Epochs

10

20

20

20

10

20

20

20

Learning Rate

3E-05

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptP (0.8M)?

Batch Size

32

# Epochs

5

20

20

20

10

20

20

20

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

16

Max Seq. Len.

128

RoBERTa large

AdptH (6M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptH (0.8M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

8

Max Seq. Len.

128

Table 9: The hyperparameters we used for RoBERTa on the GLUE benchmark.

D.3

GPT-2

We train all of our GPT-2 models using AdamW (Loshchilov & Hutter, 2017) with a linear learning rate schedule for 5 epochs. We use the batch size, learning rate, and beam search beam size described in Li & Liang (2021). Accordingly, we also tune the above hyperparameters for LoRA. We report the mean over 3 random seeds; the result for each run is taken from the best epoch. The hyperparameters used for LoRA in GPT-2 are listed in Table 11. For those used for other baselines, see Li & Liang (2021).

D.4

GPT-3

For all GPT-3 experiments, we train using AdamW (Loshchilov & Hutter, 2017) for 2 epochs with a batch size of 128 samples and a weight decay factor of 0.1. We use a sequence length of 384 for

19

often introduce inference latency (Houlsby et al., 2019; Rebuf?et al., 2017) by extending model depth or reduce the model?s usable sequence length (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021) (Section 3). More importantly, these method often fail to match the ?ne-tuning baselines, posing a trade-off between ef?ciency and model quality.

We take inspiration from Li et al. (2018a); Aghajanyan et al. (2020) which show that the learned over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the change in weights during model adaptation also has a low ?intrinsic rank?, leading to our proposed Low-Rank Adaptation (LoRA) approach. LoRA allows us to train some dense layers in a neural network indirectly by optimizing rank decomposition matrices of the dense layers? change during adaptation instead, while keeping the pre-trained weights frozen, as shown in Figure 1. Using GPT-3 175B as an example, we show that a very low rank (i.e., r in Figure 1 can be one or two) suf?ces even when the full rank (i.e., d) is as high as 12,288, making LoRA both storage- and compute-ef?cient.

LoRA possesses several key advantages.

? A pre-trained model can be shared and used to build many small LoRA modules for different tasks. We can freeze the shared model and ef?ciently switch tasks by replacing the matrices A and B in Figure 1, reducing the storage requirement and task-switching over-

head signi?cantly.

? LoRA makes training more ef?cient and lowers the hardware barrier to entry by up to 3

times when using adaptive optimizers since we do not need to calculate the gradients or

maintain the optimizer states for most parameters. Instead, we only optimize the injected,

much smaller low-rank matrices.

? Our simple linear design allows us to merge the trainable matrices with the frozen weights

when deployed, introducing no inference latency compared to a fully ?ne-tuned model, by

construction.

? LoRA is orthogonal to many prior methods and can be combined with many of them, such

as pre?x-tuning. We provide an example in Appendix E.

Terminologies and Conventions

We make frequent references to the Transformer architecture

and use the conventional terminologies for its dimensions.

We call the input and output di-

mension size of a Transformer layer dmodel.

We use Wq, Wk, Wv, and Wo to refer to the

query/key/value/output projection matrices in the self-attention module. W or W0 refers to a pre-

trained weight matrix and ?W its accumulated gradient update during adaptation. We use r to

denote the rank of a LoRA module. We follow the conventions set out by (Vaswani et al., 2017;

Brown et al., 2020) and use Adam (Loshchilov & Hutter, 2019; Kingma & Ba, 2017) for model

optimization and use a Transformer MLP feedforward dimension dffn = 4 × dmodel.

2

PROBLEM STATEMENT

While our proposal is agnostic to training objective, we focus on language modeling as our motivat-

ing use case. Below is a brief description of the language modeling problem and, in particular, the

maximization of conditional probabilities given a task-speci?c prompt.

Suppose we are given a pre-trained autoregressive language model P?(y|x) parametrized by ?.

For instance, P?(y|x) can be a generic multi-task learner such as GPT (Radford et al., b; Brown

et al., 2020) based on the Transformer architecture (Vaswani et al., 2017). Consider adapting this

pre-trained model to downstream conditional text generation tasks, such as summarization, machine

reading comprehension (MRC), and natural language to SQL (NL2SQL). Each downstream task is

represented by a training dataset of context-target pairs: Z = {(xi, yi)}i=1,..,N, where both xi and

yi are sequences of tokens. For example, in NL2SQL, xi is a natural language query and yi its *corresponding SQL command; for summarization, xi is the content of an article and yi its summary.*

2

guarantees that we do not introduce any additional latency during inference compared to a ?ne-tuned

model by construction.

4.2

APPLYING LORA TO TRANSFORMER

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the number of trainable parameters. In the Transformer architecture, there are four weight matrices in the self-attention module (Wq, Wk, Wv, Wo) and two in the MLP module. We treat Wq (or Wk, Wv) as a single matrix of dimension dmodel × dmodel, even though the output dimension is usually sliced into attention heads. We limit our study to only adapting the attention weights for downstream tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity and parameter-ef?ciency.We further study the effect on adapting different types of attention weight matrices in a Transformer in Section 7.1. We leave the empirical investigation of adapting the MLP layers, LayerNorm layers, and biases to a future work.

Practical Bene?ts and Limitations.

The most signi?cant bene?t comes from the reduction in

memory and storage usage. For a large Transformer trained with Adam, we reduce that VRAM usage by up to 2/3 if r ?dmodel as we do not need to store the optimizer states for the frozen parameters. On GPT-3 175B, we reduce the VRAM consumption during training from 1.2TB to 350GB. With r = 4 and only the query and value projection matrices being adapted, the checkpoint size is reduced by roughly 10,000× (from 350GB to 35MB)4. This allows us to train with signi?-cantly fewer GPUs and avoid I/O bottlenecks. Another bene?t is that we can switch between tasks while deployed at a much lower cost by only swapping the LoRA weights as opposed to all the parameters. This allows for the creation of many customized models that can be swapped in and out on the ?y on machines that store the pre-trained weights in VRAM. We also observe a 25% speedup during training on GPT-3 175B compared to full ?ne-tuning5 as we do not need to calculate the gradient for the vast majority of the parameters.

LoRA also has its limitations. For example, it is not straightforward to batch inputs to different tasks with different A and B in a single forward pass, if one chooses to absorb A and B into W to eliminate additional inference latency. Though it is possible to not merge the weights and dynamically choose the LoRA modules to use for samples in a batch for scenarios where latency is not critical.

EMPIRICAL EXPERIMENTS

We evaluate the downstream task performance of LoRA on RoBERTa (Liu et al., 2019), De-

BERTa (He et al., 2021), and GPT-2 (Radford et al., b), before scaling up to GPT-3 175B (Brown

et al., 2020). Our experiments cover a wide range of tasks, from natural language understanding

(NLU) to generation (NLG). Speci?cally, we evaluate on the GLUE (Wang et al., 2019) benchmark

for RoBERTa and DeBERTa. We follow the setup of Li & Liang (2021) on GPT-2 for a direct com-

parison and add WikiSQL (Zhong et al., 2017) (NL to SQL queries) and SAMSum (Gliwa et al.,

2019) (conversation summarization) for large-scale experiments on GPT-3. See Appendix C for

more details on the datasets we use. We use NVIDIA Tesla V100 for all experiments.

5.1

BASELINES

To compare with other baselines broadly, we replicate the setups used by prior work and reuse their

reported numbers whenever possible. This, however, means that some baselines might only appear

in certain experiments.

Fine-Tuning (FT) is a common approach for adaptation. During ?ne-tuning, the model is initialized

to the pre-trained weights and biases, and all model parameters undergo gradient updates.A simple

variant is to update only some layers while freezing others. We include one such baseline reported

in prior work (Li & Liang, 2021) on GPT-2, which adapts just the last two layers (FTTop2).

4We still need the 350GB model during deployment; however, storing 100 adapted models only requires

*350GB + 35MB * 100 ?354GB as opposed to 100 * 350GB ?35TB.*

5For GPT-3 175B, the training throughput for full ?ne-tuning is 32.5 tokens/s per V100 GPU; with the same

number of weight shards for model parallelism, the throughput is 43.1 tokens/s per V100 GPU for LoRA.

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.1

LR Schedule

Linear

DeBERTa XXL

LoRA

Batch Size

8

8

32

4

6

8

4

4

# Epochs

5

16

30

10

8

11

11

10

Learning Rate

1E-04

6E-05

2E-04

1E-04

1E-04

1E-04

2E-04

2E-04

Weight Decay

0

0.01

0.01

0

0.01

0.01

0.01

0.1

CLS Dropout

0.15

0

0

0.1

0.1

0.2

0.2

0.2

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

256

128

128

64

512

320

320

128

Table 10: The hyperparameters for DeBERTa XXL on tasks included in the GLUE benchmark.

Dataset

E2E

WebNLG

DART

Training

Optimizer

AdamW

Weight Decay

0.01

0.01

0.0

Dropout Prob

0.1

0.1

0.0

Batch Size

8

# Epoch

5

Warmup Steps

500

Learning Rate Schedule

Linear

Label Smooth

0.1

0.1

0.0

Learning Rate

0.0002

Adaptation

rq = rv = 4

LoRA ?

32

Inference

Beam Size

10

Length Penalty

0.9

0.8

0.8

no repeat ngram size

4

Table 11: The hyperparameters for GPT-2 LoRA on E2E, WebNLG and DART.

WikiSQL (Zhong et al., 2017), 768 for MNLI (Williams et al., 2018), and 2048 for SAMSum (Gliwa

et al., 2019). We tune learning rate for all method-dataset combinations. See Section D.4 for more

details on the hyperparameters used. For pre?x-embedding tuning, we ?nd the optimal lp and li

to be 256 and 8, respectively, totalling 3.2M trainable parameters. We use lp = 8 and li = 8 for

pre?x-layer tuning with 20.2M trainable parameters to obtain the overall best performance. We

present two parameter budgets for LoRA: 4.7M (rq = rv = 1 or rv = 2) and 37.7M (rq = rv = 8

or rq = rk = rv = ro = 2). We report the best validation performance from each run. The training

hyperparameters used in our GPT-3 experiments are listed in Table 12.

E

COMBINING LORA WITH PREFIX TUNING

LoRA can be naturally combined with existing pre?x-based approaches. In this section, we evaluate

two combinations of LoRA and variants of pre?x-tuning on WikiSQL and MNLI.

LoRA+Pre?xEmbed (LoRA+PE) combines LoRA with pre?x-embedding tuning, where we insert

lp + li special tokens whose embeddings are treated as trainable parameters. For more on pre?x-

embedding tuning, see Section 5.1.

LoRA+Pre?xLayer (LoRA+PL) combines LoRA with pre?x-layer tuning. We also insert lp + li

special tokens; however, instead of letting the hidden representations of these tokens evolve natu-

20

tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are

there more principled ways to do it? 4) Finally, the rank-de?ciency of ?W suggests that W could

be rank-de?cient as well, which can also be a source of inspiration for future works.

REFERENCES

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. arXiv:2012.13255 [cs], December 2020. URL http://arxiv.org/abs/2012.13255.

Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Ef?ciently, Going Beyond Kernels? In NeurIPS, 2019. Full version available at http://arxiv.org/abs/1905.10337.

Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. arXiv preprint arXiv:2001.04413, 2020a.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature puri?cation: How adversarial training performs robust deep learning. arXiv preprint arXiv:2005.10190, 2020b.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In ICML, 2019. Full version available at http://arxiv.org/abs/1811.03962.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs], July 2020. URL http://arxiv.org/abs/2005.14165.

Jian-Feng Cai, Emmanuel J Cand`es, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. SIAM Journal on optimization, 20(4):1956?1982, 2010.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), 2017. doi: 10.18653/v1/s17-2001. URL http://dx.doi.org/10.18653/v1/S17-2001.

Ronan Collobert and Jason Weston. A uni?ed architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, ICML ?08, pp. 160?167, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc?Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019a.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs], May 2019b. URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005), 2005. URL https://aclanthology.org/I05-5002.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg challenge: Generating text from rdf data. In Proceedings of the 10th International Conference on Natural Language Generation, pp. 124?133, 2017.

13

Method

WebNLG

BLEU?

MET?

TER?

U

S

A

U

S

A

U

S

A

GPT-2 Medium

Fine-Tune (354M)

27.7

64.2

46.5

.30

.45

.38

.76

.33

.53

AdapterL (0.37M)

45.1

54.5

50.2

.36

.39

.38

.46

.40

.43

AdapterL (11M)

48.3

60.4

54.9

.38

.43

.41

.45

.35

.39

FTTop2 (24M)

18.9

53.6

36.0

.23

.38

.31

.99

.49

.72

Pre?x (0.35M)

45.6

62.9

55.1

.38

.44

.41

.49

.35

.40

LoRA (0.35M)

46.7±.4

62.1±.2

55.3±.2

.38

.44

.41

.46

.33

.39

GPT-2 Large

Fine-Tune (774M)

43.1

65.3

55.5

.38

.46

.42

.53

.33

.42

AdapterL (0.88M)

49.8±.0

61.1±.0

56.0±.0

.38

.43

.41

.44

.35

.39

AdapterL (23M)

49.2±.1

64.7±.2

57.7±.1

.39

.46

.43

.46

.33

.39

Pre?x (0.77M)

47.7

63.4

56.3

.39

.45

.42

.48

.34

.40

LoRA (0.77M)

48.4±.3

64.0±.3

57.0±.1

.39

.45

.42

.45

.32

.38

Table 14: GPT-2 with different adaptation methods on WebNLG. The variances of MET and TER are less than 0.01 for all the experiments we ran. ?U? indicates unseen categories, ?S? indicates seen categories, and ?A? indicates all categories in the test set of WebNLG.

F.2

ADDITIONAL EXPERIMENTS ON GPT-3

We present additional runs on GPT-3 with different adaptation methods in Table 15. The focus is on identifying the trade-off between performance and the number of trainable parameters.

F.3

LOW-DATA REGIME

To evaluate the performance of different adaptation approaches in the low-data regime. we randomly sample 100, 1k and 10k training examples from the full training set of MNLI to form the low-data MNLI-n tasks. In Table 16, we show the performance of different adaptation approaches on MNLI-n. To our surprise, Pre?xEmbed and Pre?xLayer performs very poorly on MNLI-100 dataset, with Pre?xEmbed performing only slightly better than random chance (37.6% vs. 33.3%). Pre?xLayer performs better than Pre?xEmbed but is still signi?cantly worse than Fine-Tune or LoRA on MNLI-100. The gap between pre?x-based approaches and LoRA/Fine-tuning becomes smaller as we in-crease the number of training examples, which might suggest that pre?x-based approaches are not suitable for low-data tasks in GPT-3. LoRA achieves better performance than ?ne-tuning on both MNLI-100 and MNLI-Full, and comparable results on MNLI-1k and MNLI-10K considering the (±0.3) variance due to random seeds.

The training hyperparameters of different adaptation approaches on MNLI-n are reported in Ta-ble 17. We use a smaller learning rate for Pre?xLayer on the MNLI-100 set, as the training loss does not decrease with a larger learning rate.

G

MEASURING SIMILARITY BETWEEN SUBSPACES

In this paper we use the measure ?(A, B, i, j) = ?(U i

A, U j

B) = ?U i?

A UB?2

F

min{i,j}

to measure the subspace

similarity between two column orthonormal matrices U i

A ?Rd×i and U j

B ?Rd×j, obtained by

taking columns of the left singular matrices of A and B. We point out that this similarity is simply

a reverse of the standard Projection Metric that measures distance between subspaces Ham & Lee

(2008).

22

to maximize downstream performance? 2) Is the ?optimal? adaptation matrix ?W really rank-

de?cient? If so, what is a good rank to use in practice? 3) What is the connection between ?W and

W? Does ?W highly correlate with W? How large is ?W comparing to W?

We believe that our answers to question (2) and (3) shed light on the fundamental principles of using

pre-trained language models for downstream tasks, which is a critical topic in NLP.

7.1

WHICH WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Given a limited parameter budget, which types of weights should we adapt with LoRA to obtain

the best performance on downstream tasks? As mentioned in Section 4.2, we only consider weight

matrices in the self-attention module. We set a parameter budget of 18M (roughly 35MB if stored

in FP16) on GPT-3 175B, which corresponds to r = 8 if we adapt one type of attention weights or

r = 4 if we adapt two types, for all 96 layers. The result is presented in Table 5.

# of Trainable Parameters = 18M

Weight Type

Wq

Wk

Wv

Wo

Wq, Wk

Wq, Wv

Wq, Wk, Wv, Wo

Rank r	*WikiSQL (±0.5%)*	*MultiNLI (±0.1%)*
8	*70.4*	*91.0*
8	*70.0*	*90.8*
8	*73.0*	*91.0*
8	*73.2*	*91.3*
4	*71.4*	*91.3*
4	*73.7*	*91.3*
2	*73.7*	*91.7*

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both Wq and Wv gives the best performance overall. We ?nd the standard deviation across random seeds to be consistent for a given dataset, which we report in the ?rst column.

Note that putting all the parameters in ?Wq or ?Wk results in signi?cantly lower performance, while adapting both Wq and Wv yields the best result. This suggests that even a rank of four captures enough information in ?W such that it is preferable to adapt more weight matrices than adapting a single type of weights with a larger rank.

7.2

WHAT IS THE OPTIMAL RANK r FOR LORA?

We turn our attention to the effect of rank r on model performance.

We adapt {Wq, Wv},

{Wq, Wk, Wv, Wc}, and just Wq for a comparison.

Weight Type

r = 1

r = 2

r = 4

r = 8

r = 64

WikiSQL(±0.5%)

Wq

68.8

69.6

70.5

70.4

70.0

Wq, Wv

73.4

73.3

73.7

73.8

73.5

Wq, Wk, Wv, Wo

74.1

73.7

74.0

74.0

73.9

MultiNLI (±0.1%)

Wq

90.7

90.9

91.1

90.7

90.7

Wq, Wv

91.3

91.4

91.3

91.6

91.4

Wq, Wk, Wv, Wo

91.2

91.7

91.7

91.5

91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suf?ces for adapting both Wq and Wv on these datasets while training Wq alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

Table 6 shows that, surprisingly, LoRA already performs competitively with a very small r (more so for {Wq, Wv} than just Wq). This suggests the update matrix ?W could have a very small ?intrinsic rank?.6 To further support this ?nding, we check the overlap of the subspaces learned by different choices of r and by different random seeds. We argue that increasing r does not cover a more meaningful subspace, which suggests that a low-rank adaptation matrix is suf?cient.

6However, we do not expect a small r to work for every task or dataset. Consider the following thought experiment: if the downstream task were in a different language than the one used for pre-training, retraining the entire model (similar to LoRA with r = dmodel) could certainly outperform LoRA with a small r.

LORA: LOW-RANK ADAPTATION OF LARGE LAN-

GUAGE MODELS

Edward Hu?

Yelong Shen?

Phillip Wallis

Zeyuan Allen-Zhu

Yuanzhi Li

Shean Wang

Lu Wang

Weizhu Chen

Microsoft Corporation

{edwardhu, yeshe, phwallis, zeyuana,

yuanzhil, swang, luw, wzchen}@microsoft.com

yuanzhil@andrew.cmu.edu

(Version 2)

ABSTRACT

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full ?ne-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example ? deploying indepen-dent instances of ?ne-tuned models, each with 175B parameters, is prohibitively expensive. We propose Low-Rank Adaptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable pa-rameters for downstream tasks. Compared to GPT-3 175B ?ne-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than ?ne-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite hav-ing fewer trainable parameters, a higher training throughput, and, unlike adapters, no additional inference latency. We also provide an empirical investigation into rank-de?ciency in language model adaptation, which sheds light on the ef?cacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at https://github.com/microsoft/LoRA.

1

INTRODUCTION

Pretrained

Weights

????×?

x

h

?= 0

?= ?(0, ?2)

?

?

Pretrained

Weights

????×?

x

f(x)

?

Figure 1: Our reparametriza-

tion. We only train A and B.

Many applications in natural language processing rely on adapt-

ing one large-scale, pre-trained language model to multiple down-

stream applications. Such adaptation is usually done via ?ne-tuning,

which updates all the parameters of the pre-trained model. The ma-

jor downside of ?ne-tuning is that the new model contains as many

parameters as in the original model. As larger models are trained

every few months, this changes from a mere ?inconvenience? for

GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a

critical deployment challenge for GPT-3 (Brown et al., 2020) with

175 billion trainable parameters.1

Many sought to mitigate this by adapting only some parameters or

learning external modules for new tasks. This way, we only need

to store and load a small number of task-speci?c parameters in ad-

dition to the pre-trained model for each task, greatly boosting the

operational ef?ciency when deployed. However, existing techniques

?Equal contribution.

0Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.

1While GPT-3 175B achieves non-trivial performance with few-shot learning, ?ne-tuning boosts its perfor-

mance signi?cantly as shown in Appendix A.

arXiv:2106.09685v2 [cs.CL] 16 Oct 2021

Hyperparameters

Fine-Tune

PreEmbed

PreLayer

BitFit

AdapterH

LoRA

Optimizer

AdamW

Batch Size

128

# Epoch

2

Warmup Tokens

250,000

LR Schedule

Linear

Learning Rate

5.00E-06

5.00E-04

1.00E-04

1.6E-03

1.00E-04

2.00E-04

Table 12: The training hyperparameters used for different GPT-3 adaption methods. We use the same hyperparameters for all datasets after tuning learning rate.

rally, we replace them after every Transformer block with an input agnostic vector. Thus, both the embeddings and subsequent Transformer block activations are treated as trainable parameters. For more on pre?x-layer tuning, see Section 5.1.

In Table 15, we show the evaluation results of LoRA+PE and LoRA+PL on WikiSQL and MultiNLI. First of all, LoRA+PE signi?cantly outperforms both LoRA and pre?x-embedding tuning on

WikiSQL, which indicates that LoRA is somewhat orthogonal to pre?x-embedding tuning. On MultiNLI, the combination of LoRA+PE doesn?t perform better than LoRA, possibly because LoRA on its own already achieves performance comparable to the human baseline. Secondly, we notice that LoRA+PL performs slightly worse than LoRA even with more trainable parameters. We attribute this to the fact that pre?x-layer tuning is very sensitive to the choice of learning rate and thus makes the optimization of LoRA weights more dif?cult in LoRA+PL.

F

ADDITIONAL EMPIRICAL EXPERIMENTS

F.1

ADDITIONAL EXPERIMENTS ON GPT-2

We also repeat our experiment on DART (Nan et al., 2020) and WebNLG (Gardent et al., 2017) following the setup of Li & Liang (2021). The result is shown in Table 13. Similar to our result on E2E NLG Challenge, reported in Section 5, LoRA performs better than or at least on-par with pre?x-based approaches given the same number of trainable parameters.

Method

# Trainable

DART

Parameters

BLEU?

MET?

TER?

GPT-2 Medium

Fine-Tune

354M

46.2

0.39

0.46

AdapterL

0.37M

42.4

0.36

0.48

AdapterL

11M

45.2

0.38

0.46

FTTop2

24M

41.0

0.34

0.56

PrefLayer

0.35M

46.4

0.38

0.46

LoRA

0.35M

47.1±.2

0.39

0.46

GPT-2 Large

Fine-Tune

774M

47.0

0.39

0.46

AdapterL

0.88M

45.7±.1

0.38

0.46

AdapterL

23M

47.1±.1

0.39

0.45

PrefLayer

0.77M

46.7

0.38

0.45

LoRA

0.77M

47.5±.1

0.39

0.45

Table 13: GPT-2 with different adaptation methods on DART. The variances of MET and TER are

less than 0.01 for all adaption approaches.

21

Batch Size

32

16

1

Sequence Length

512

256

128

|?|

0.5M

11M

11M

Fine-Tune/LoRA

1449.4±0.8

338.0±0.6

19.8±2.7

AdapterL

1482.0±1.0 (+2.2%)

354.8±0.5 (+5.0%)

23.9±2.1 (+20.7%)

AdapterH

1492.2±1.0 (+3.0%)

366.3±0.5 (+8.4%)

25.8±2.2 (+30.3%)

Table 1: Infernece latency of a single forward pass in GPT-2 medium measured in milliseconds, averaged over 100 trials. We use an NVIDIA Quadro RTX8000. ?|?|? denotes the number of trainable parameters in adapter layers. AdapterL and AdapterH are two variants of adapter tuning, which we describe in Section 5.1. The inference latency introduced by adapter layers can be signi?cant in an online, short-sequence-length scenario. See the full study in Appendix B.

4

OUR METHOD

We describe the simple design of LoRA and its practical bene?ts. The principles outlined here apply to any dense layers in deep learning models, though we only focus on certain weights in Transformer language models in our experiments as the motivating use case.

4.1

LOW-RANK-PARAMETRIZED UPDATE MATRICES

A neural network contains many dense layers which perform matrix multiplication. The weight matrices in these layers typically have full-rank. When adapting to a speci?c task, Aghajanyan et al. (2020) shows that the pre-trained language models have a low ?instrisic dimension? and can still learn ef?ciently despite a random projection to a smaller subspace. Inspired by this, we hypothesize the updates to the weights also have a low ?intrinsic rank? during adaptation. For a pre-trained weight matrix W0 ?Rd×k, we constrain its update by representing the latter with a low-rank decomposition W0 + ?W = W0 + BA, where B ?Rd×r, A ?Rr×k, and the rank r ?min(d, k).

During training, W0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W0 and ?W = BA are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For h = W0x, our modi?ed forward pass yields:

h = W0x + ?Wx = W0x + BAx

(3)

We illustrate our reparametrization in Figure 1. We use a random Gaussian initialization for A and zero for B, so ?W = BA is zero at the beginning of training. We then scale ?Wx by ?

r , where ?

is a constant in r. When optimizing with Adam, tuning ? is roughly the same as tuning the learning

rate if we scale the initialization appropriately. As a result, we simply set ? to the ?rst r we try

and do not tune it. This scaling helps to reduce the need to retune hyperparameters when we vary

r (Yang & Hu, 2021).

A Generalization of Full Fine-tuning.

A more general form of ?ne-tuning allows the training of

a subset of the pre-trained parameters. LoRA takes a step further and does not require the accumu-

lated gradient update to weight matrices to have full-rank during adaptation. This means that when

applying LoRA to all weight matrices and training all biases2, we roughly recover the expressive-

ness of full ?ne-tuning by setting the LoRA rank r to the rank of the pre-trained weight matrices. In

other words, as we increase the number of trainable parameters 3, training LoRA roughly converges

to training the original model, while adapter-based methods converges to an MLP and pre?x-based

methods to a model that cannot take long input sequences.

No Additional Inference Latency.

When deployed in production, we can explicitly compute and

store W = W0 + BA and perform inference as usual. Note that both W0 and BA are in Rd×k.

When we need to switch to another downstream task, we can recover W0 by subtracting BA and

then adding a different B?A?, a quick operation with very little memory overhead. Critically, this

2They represent a negligible number of parameters compared to weights.

3An inevitability when adapting to hard tasks.

4

Current Question:

Lora's practical benefits, advantages and limitations.

Answer:

Evaluation 5:

### Evaluation of the Answer:

Relevance: Excellent

The answer directly addresses the question by identifying the specific weight matrices in the Transformer (Wq and Wv) that should be adapted using LoRA to achieve optimal performance under the given parameter budget.

Faithfulness: Excellent

The answer is strictly supported by the provided context and empirical findings, such as validation accuracy results and recommendations derived from Table 5.

Completeness: Excellent

The answer covers all major aspects, including the specific matrices to adapt, justification for their selection, rank considerations, and a summary table showing validation accuracies for different configurations.

Clarity: Excellent

The answer is well-structured, logically organized, and easy to understand. Key points are clearly delineated, and technical terms are explained in the context.

Conciseness: Excellent

The answer is free of unnecessary repetition or verbosity, providing all essential information succinctly while avoiding extraneous details.

### Final Evaluation:

The answer fully satisfies all criteria and provides a high-quality response to the question.

Weighted Score: 100.0

Question 5: which WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Answer 5: Based on the empirical experiments outlined in the provided context, the optimal weight matrices to apply LoRA to within the Transformer architecture are **Wq (query)** and **Wv (value)**. These matrices yield the best performance on downstream tasks given a limited parameter budget.

### Key Findings:

1. **Adapting Wq and Wv**:

   - When both Wq and Wv are adapted, the validation accuracy is consistently higher compared to adapting a single matrix or other combinations of matrices.

   - For example, on WikiSQL and MultiNLI datasets, adapting **both Wq and Wv** results in better performance than adapting Wq or Wk alone.

2. **Rank Considerations**:

   - A low rank (e.g., r = 4) is often sufficient for capturing meaningful information in the weight update matrices (?W). This ensures efficiency in memory and computation while maintaining strong task performance.

3. **Adapting All Attention Matrices (Wq, Wk, Wv, Wo)**:

   - While adapting all attention matrices may slightly improve performance, it requires more trainable parameters and might not be as efficient as focusing on Wq and Wv.

4. **Practical Recommendation**:

   - If memory and computational efficiency are priorities, adapting **Wq and Wv** with a small rank (e.g., r = 4) is the most effective approach.

### Summary Table (Validation Accuracy):

| Weight Type | WikiSQL (%) | MultiNLI (%) |

Wq	70.4	91.0
Wk	70.0	90.8
Wv	73.0	91.0
Wo	73.2	91.3
Wq, Wv	**73.7**	**91.3**
Wq, Wk, Wv, Wo	**73.7**	**91.7**

In conclusion, **Wq and Wv** are the best matrices to adapt using LoRA for efficiency and performance on downstream tasks.

Response Latency: 9.01 seconds

Evaluation Latency: 5.07 seconds

[Prompt used for Q5]

You are a helpful assistant. Use the following history and context to answer.

History:

Q: What is a GPT pre-trained model?

A: A GPT pre-trained model is a type of large language model (LLM) built using the transformer deep learning architecture. It is trained using a method called **generative pretraining**, which is a form of unsupervised learning. In this process, the model is exposed to vast amounts of unlabeled text data and learns to detect patterns, relationships, and structures within the data. This pretraining enables the model to understand and generate natural language effectively.

The pre-trained model can then be fine-tuned on specific tasks, such as text generation, translation, summarization, or question answering, making it highly versatile for applications in natural language processing (NLP). Examples of such models include GPT-3 and GPT-4, which are widely used in tools like chatbots, virtual assistants, and content generation platforms.

Q: What advantages does it have compared with other pre-trained models?

A: Pre-trained models, such as large-scale pre-trained language models (PTMs) like BERT and GPT, offer distinct advantages compared with other pre-trained models. Here are some key advantages:

1. **Reduced Training Time and Computational Cost**: Pre-trained models come with weights and parameters already optimized

from prior training on large datasets. This eliminates the need to train a model from scratch, saving significant time and computational resources.

*2. **Generalization Ability**: Pre-trained models, especially large-scale ones, often generalize better across tasks because they are trained on diverse and extensive datasets. This makes them suitable for a wide range of applications, even without substantial fine-tuning.*

*3. **Transfer Learning**: These models are highly effective for transfer learning. You can fine-tune them on specific tasks with relatively small datasets, leveraging their pre-existing knowledge to achieve high performance.*

*4. **State-of-the-Art Performance**: Models like GPT and BERT have set benchmarks in many natural language processing (NLP) tasks, such as text classification, sentiment analysis, question answering, and more.*

*5. **Ease of Integration**: Pre-trained models are readily available through platforms like Hugging Face, OpenAI, and others. This makes it easier for developers to integrate advanced AI capabilities into their products without needing deep expertise in model architecture and training.*

*6. **Cost-Effectiveness**: By using pre-trained models, companies can avoid the costs associated with curating large datasets, training models, and maintaining infrastructure, making them a more economical option.*

*7. **Community Support and Updates**: Popular pre-trained models benefit from active developer communities and continuous updates, ensuring access to improvements, bug fixes, and new features.*

*8. **Versatility Across Domains**: Many pre-trained models can be adapted to tasks beyond their original training domain. For example, GPT models can be fine-tuned for creative writing, customer service, or technical documentation.*

These advantages make pre-trained models a powerful choice for incorporating AI into products and workflows compared to other pre-trained options.

Context:

guarantees that we do not introduce any additional latency during inference compared to a ?ne-tuned
model by construction.

4.2

APPLYING LORA TO TRANSFORMER

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the number of trainable parameters. In the Transformer architecture, there are four weight matrices in the self-attention module (Wq, Wk, Wv, Wo) and two in the MLP module. We treat Wq (or Wk, Wv) as a single matrix of dimension dmodel × dmodel, even though the output dimension is usually sliced into attention heads. We limit our study to only adapting the attention weights for downstream tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity and parameter-ef?ciency.We further study the effect on adapting different types of attention weight matrices in a Transformer in Section 7.1. We leave the empirical investigation of adapting the MLP layers, LayerNorm layers, and biases to a future work.

Practical Bene?ts and Limitations.

The most signi?cant bene?t comes from the reduction in

memory and storage usage. For a large Transformer trained with Adam, we reduce that VRAM usage by up to 2/3 if r ?dmodel as we do not need to store the optimizer states for the frozen parameters. On GPT-3 175B, we reduce the VRAM consumption during training from 1.2TB to 350GB. With r = 4 and only the query and value projection matrices being adapted, the checkpoint size is reduced by roughly 10,000× (from 350GB to 35MB)4. This allows us to train with signi?-cantly fewer GPUs and avoid I/O bottlenecks. Another bene?t is that we can switch between tasks while deployed at a much lower cost by only swapping the LoRA weights as opposed to all the parameters. This allows for the creation of many customized models that can be swapped in and out on the ?y on machines that store the pre-trained weights in VRAM. We also observe a 25% speedup during training on GPT-3 175B compared to full ?ne-tuning5 as we do not need to calculate the gradient for the vast majority of the parameters.

LoRA also has its limitations. For example, it is not straightforward to batch inputs to different tasks with different A and B in a single forward pass, if one chooses to absorb A and B into W to eliminate additional inference latency. Though it is possible to not merge the weights and dynamically choose the LoRA modules to use for samples in a batch for scenarios where latency is not critical.

5

EMPIRICAL EXPERIMENTS

We evaluate the downstream task performance of LoRA on RoBERTa (Liu et al., 2019), De-BERTa (He et al., 2021), and GPT-2 (Radford et al., b), before scaling up to GPT-3 175B (Brown et al., 2020). Our experiments cover a wide range of tasks, from natural language understanding (NLU) to generation (NLG). Speci?cally, we evaluate on the GLUE (Wang et al., 2019) benchmark *for RoBERTa and DeBERTa. We follow the setup of Li & Liang (2021) on GPT-2 for a direct com-*

parison and add WikiSQL (Zhong et al., 2017) (NL to SQL queries) and SAMSum (Gliwa et al.,

2019) (conversation summarization) for large-scale experiments on GPT-3. See Appendix C for

more details on the datasets we use. We use NVIDIA Tesla V100 for all experiments.

5.1

BASELINES

To compare with other baselines broadly, we replicate the setups used by prior work and reuse their

reported numbers whenever possible. This, however, means that some baselines might only appear

in certain experiments.

Fine-Tuning (FT) is a common approach for adaptation. During ?ne-tuning, the model is initialized

to the pre-trained weights and biases, and all model parameters undergo gradient updates.A simple

variant is to update only some layers while freezing others. We include one such baseline reported

in prior work (Li & Liang, 2021) on GPT-2, which adapts just the last two layers (FTTop2).

4We still need the 350GB model during deployment; however, storing 100 adapted models only requires

*350GB + 35MB * 100 ?354GB as opposed to 100 * 350GB ?35TB.*

5For GPT-3 175B, the training throughput for full ?ne-tuning is 32.5 tokens/s per V100 GPU; with the same

number of weight shards for model parallelism, the throughput is 43.1 tokens/s per V100 GPU for LoRA.

to maximize downstream performance? 2) Is the ?optimal? adaptation matrix ?W really rank-

de?cient? If so, what is a good rank to use in practice? 3) What is the connection between ?W and

W? Does ?W highly correlate with W? How large is ?W comparing to W?

We believe that our answers to question (2) and (3) shed light on the fundamental principles of using

pre-trained language models for downstream tasks, which is a critical topic in NLP.

7.1

WHICH WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Given a limited parameter budget, which types of weights should we adapt with LoRA to obtain

the best performance on downstream tasks? As mentioned in Section 4.2, we only consider weight

matrices in the self-attention module. We set a parameter budget of 18M (roughly 35MB if stored

in FP16) on GPT-3 175B, which corresponds to r = 8 if we adapt one type of attention weights or

r = 4 if we adapt two types, for all 96 layers. The result is presented in Table 5.

# of Trainable Parameters = 18M

Weight Type

Wq

Wk

Wv

Wo

Wq, Wk

Wq, Wv

Wq, Wk, Wv, Wo

Rank r

8

8

8

8

4

4

2

WikiSQL (±0.5%)

70.4

70.0

73.0

73.2

71.4

73.7

73.7

MultiNLI (±0.1%)

91.0

90.8

91.0

91.3

91.3

91.3

91.7

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both Wq and Wv gives the best performance overall. We ?nd the standard deviation across random seeds to be

consistent for a given dataset, which we report in the ?rst column.

Note that putting all the parameters in ?Wq or ?Wk results in signi?cantly lower performance,

while adapting both Wq and Wv yields the best result. This suggests that even a rank of four

captures enough information in ?W such that it is preferable to adapt more weight matrices than

adapting a single type of weights with a larger rank.

7.2

WHAT IS THE OPTIMAL RANK r FOR LORA?

We turn our attention to the effect of rank r on model performance.

We adapt {Wq, Wv},

{Wq, Wk, Wv, Wc}, and just Wq for a comparison.

Weight Type

r = 1

r = 2

r = 4

r = 8

r = 64

WikiSQL(±0.5%)

Wq

68.8

69.6

70.5

70.4

70.0

Wq, Wv

73.4

73.3

73.7

73.8

73.5

Wq, Wk, Wv, Wo

74.1

73.7

74.0

74.0

73.9

MultiNLI (±0.1%)

Wq

90.7

90.9

91.1

90.7

90.7

Wq, Wv

91.3

91.4

91.3

91.6

91.4

Wq, Wk, Wv, Wo

91.2

91.7

91.7

91.5

91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suf?ces for adapting both Wq and Wv on these datasets while training Wq alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

Table 6 shows that, surprisingly, LoRA already performs competitively with a very small r (more so for {Wq, Wv} than just Wq). This suggests the update matrix ?W could have a very small ?intrinsic rank?.6 To further support this ?nding, we check the overlap of the subspaces learned by different choices of r and by different random seeds. We argue that increasing r does not cover a more meaningful subspace, which suggests that a low-rank adaptation matrix is suf?cient.

6However, we do not expect a small r to work for every task or dataset. Consider the following thought experiment: if the downstream task were in a different language than the one used for pre-training, retraining the entire model (similar to LoRA with r = dmodel) could certainly outperform LoRA with a small r.

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.06

LR Schedule

Linear

RoBERTa base

LoRA

Batch Size

16

16

16

32

32

16

32

16

# Epochs

30

60

30

80

25

25

80

40

Learning Rate

5E-04

5E-04

4E-04

4E-04

4E-04

5E-04

5E-04

4E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

512

RoBERTa large

LoRA

Batch Size

4

4

4

4

4

4

8

8

# Epochs

10

10

20

20

10

20

20

30

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

128

512

128

512

512

512

512

RoBERTa large

LoRA?

Batch Size

4

# Epochs

10

10

20

20

10

20

20

10

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

RoBERTa large

AdptP (3M)?

Batch Size

32

# Epochs

10

20

20

20

10

20

20

20

Learning Rate

3E-05

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptP (0.8M)?

Batch Size

32

# Epochs

5

20

20

20

10

20

20

20

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

16

Max Seq. Len.

128

RoBERTa large

AdptH (6M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptH (0.8M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

8

Max Seq. Len.

128

Table 9: The hyperparameters we used for RoBERTa on the GLUE benchmark.

D.3

GPT-2

We train all of our GPT-2 models using AdamW (Loshchilov & Hutter, 2017) with a linear learning rate schedule for 5 epochs. We use the batch size, learning rate, and beam search beam size described in Li & Liang (2021). Accordingly, we also tune the above hyperparameters for LoRA. We report the *mean over 3 random seeds; the result for each run is taken from the best epoch. The hyperparameters used for LoRA in GPT-2 are listed in Table 11. For those used for other baselines, see Li & Liang (2021).*

D.4

GPT-3

For all GPT-3 experiments, we train using AdamW (Loshchilov & Hutter, 2017) for 2 epochs with a batch size of 128 samples and a weight decay factor of 0.1. We use a sequence length of 384 for

often introduce inference latency (Houlsby et al., 2019; Rebuf?et al., 2017) by extending model depth or reduce the model?s usable sequence length (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021) (Section 3). More importantly, these method often fail to match the ?ne-tuning baselines, posing a trade-off between ef?ciency and model quality.

We take inspiration from Li et al. (2018a); Aghajanyan et al. (2020) which show that the learned over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the change in weights during model adaptation also has a low ?intrinsic rank?, leading to our proposed Low-Rank Adaptation (LoRA) approach. LoRA allows us to train some dense layers in a neural network indirectly by optimizing rank decomposition matrices of the dense layers? change during adaptation instead, while keeping the pre-trained weights frozen, as shown in Figure 1. Using GPT-3 175B as an example, we show that a very low rank (i.e., r in Figure 1 can be one or two) suf?ces even when the full rank (i.e., d) is as high as 12,288, making LoRA both storage- and compute-ef?cient.

LoRA possesses several key advantages.

? A pre-trained model can be shared and used to build many small LoRA modules for different tasks. We can freeze the shared model and ef?ciently switch tasks by replacing the matrices A and B in Figure 1, reducing the storage requirement and task-switching overhead signi?cantly.

? LoRA makes training more ef?cient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most parameters. Instead, we only optimize the injected, much smaller low-rank matrices.

? Our simple linear design allows us to merge the trainable matrices with the frozen weights when deployed, introducing no inference latency compared to a fully ?ne-tuned model, by construction.

? LoRA is orthogonal to many prior methods and can be combined with many of them, such

as pre?x-tuning. We provide an example in Appendix E.

Terminologies and Conventions

We make frequent references to the Transformer architecture

and use the conventional terminologies for its dimensions.

We call the input and output di-

mension size of a Transformer layer dmodel.

We use Wq, Wk, Wv, and Wo to refer to the

query/key/value/output projection matrices in the self-attention module. W or W0 refers to a pre-

trained weight matrix and ?W its accumulated gradient update during adaptation. We use r to

denote the rank of a LoRA module. We follow the conventions set out by (Vaswani et al., 2017;

Brown et al., 2020) and use Adam (Loshchilov & Hutter, 2019; Kingma & Ba, 2017) for model

optimization and use a Transformer MLP feedforward dimension dffn = 4 × dmodel.

2

PROBLEM STATEMENT

While our proposal is agnostic to training objective, we focus on language modeling as our motivat-

ing use case. Below is a brief description of the language modeling problem and, in particular, the

maximization of conditional probabilities given a task-speci?c prompt.

Suppose we are given a pre-trained autoregressive language model P?(y|x) parametrized by ?.

For instance, P?(y|x) can be a generic multi-task learner such as GPT (Radford et al., b; Brown

et al., 2020) based on the Transformer architecture (Vaswani et al., 2017). Consider adapting this

pre-trained model to downstream conditional text generation tasks, such as summarization, machine

reading comprehension (MRC), and natural language to SQL (NL2SQL). Each downstream task is

represented by a training dataset of context-target pairs: Z = {(xi, yi)}i=1,..,N, where both xi and

yi are sequences of tokens. For example, in NL2SQL, xi is a natural language query and yi its

corresponding SQL command; for summarization, xi is the content of an article and yi its summary.

2

tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are

there more principled ways to do it? 4) Finally, the rank-de?ciency of ?W suggests that W could

be rank-de?cient as well, which can also be a source of inspiration for future works.

REFERENCES

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the

Effectiveness of Language Model Fine-Tuning. arXiv:2012.13255 [cs], December 2020. URL

http://arxiv.org/abs/2012.13255.

Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Ef?ciently, Going Beyond Kernels? In NeurIPS, 2019. Full version available at http://arxiv.org/abs/1905.10337.

Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. arXiv preprint arXiv:2001.04413, 2020a.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature puri?cation: How adversarial training performs robust deep learning. arXiv preprint arXiv:2005.10190, 2020b.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In ICML, 2019. Full version available at http://arxiv.org/abs/1811.

03962.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs], July 2020. URL http://arxiv.org/abs/2005.14165.

Jian-Feng Cai, Emmanuel J Cand`es, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. SIAM Journal on optimization, 20(4):1956?1982, 2010.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), 2017. doi: 10.18653/v1/s17-2001. URL http://dx.doi.org/10.18653/v1/S17-2001.

Ronan Collobert and Jason Weston. A uni?ed architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, ICML ?08, pp. 160?167, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc?Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019a.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep

Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs], May 2019b.

URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases.

In Proceedings of the Third International Workshop on Paraphrasing (IWP2005), 2005. URL

https://aclanthology.org/I05-5002.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg

challenge: Generating text from rdf data. In Proceedings of the 10th International Conference on

Natural Language Generation, pp. 124?133, 2017.

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.1

LR Schedule

Linear

DeBERTa XXL

LoRA

Batch Size

8

8

32

4

6

8

4

4

# Epochs

5

16

30

10

8

11

11

10

Learning Rate

1E-04

6E-05

2E-04

1E-04

1E-04

1E-04

2E-04

2E-04

Weight Decay

0

0.01

0.01

0

0.01

0.01

0.01

0.1

CLS Dropout

0.15

0

0

0.1

0.1

0.2

0.2

0.2

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

256

128

128

64

512

320

320

128

Table 10: The hyperparameters for DeBERTa XXL on tasks included in the GLUE benchmark.

Dataset

E2E

WebNLG

DART

Training

Optimizer

AdamW

Weight Decay

0.01

0.01

0.0

Dropout Prob

0.1

0.1

0.0

Batch Size

8

# Epoch

5

Warmup Steps

500

Learning Rate Schedule

Linear

Label Smooth

0.1

0.1

0.0

Learning Rate

0.0002

Adaptation

rq = rv = 4

LoRA ?

32

Inference

Beam Size

10

Length Penalty

0.9

0.8

0.8

no repeat ngram size

4

Table 11: The hyperparameters for GPT-2 LoRA on E2E, WebNLG and DART.

WikiSQL (Zhong et al., 2017), 768 for MNLI (Williams et al., 2018), and 2048 for SAMSum (Gliwa

et al., 2019). We tune learning rate for all method-dataset combinations. See Section D.4 for more

details on the hyperparameters used. For pre?x-embedding tuning, we ?nd the optimal lp and li *to be 256 and 8, respectively, totalling 3.2M trainable parameters. We use lp = 8 and li = 8 for pre?x-layer tuning with 20.2M trainable parameters to obtain the overall best performance. We present two parameter budgets for LoRA: 4.7M (rq = rv = 1 or rv = 2) and 37.7M (rq = rv = 8 or rq = rk = rv = ro = 2). We report the best validation performance from each run. The training hyperparameters used in our GPT-3 experiments are listed in Table 12.*

E

COMBINING LORA WITH PREFIX TUNING

LoRA can be naturally combined with existing pre?x-based approaches. In this section, we evaluate two combinations of LoRA and variants of pre?x-tuning on WikiSQL and MNLI.

LoRA+Pre?xEmbed (LoRA+PE) combines LoRA with pre?x-embedding tuning, where we insert lp + li special tokens whose embeddings are treated as trainable parameters. For more on pre?x-embedding tuning, see Section 5.1.

LoRA+Pre?xLayer (LoRA+PL) combines LoRA with pre?x-layer tuning. We also insert lp + li special tokens; however, instead of letting the hidden representations of these tokens evolve natu-

LORA: LOW-RANK ADAPTATION OF LARGE LAN-
GUAGE MODELS

Edward Hu?

Yelong Shen?

Phillip Wallis

Zeyuan Allen-Zhu

Yuanzhi Li

Shean Wang

Lu Wang

Weizhu Chen

Microsoft Corporation

{edwardhu, yeshe, phwallis, zeyuana,

yuanzhil, swang, luw, wzchen}@microsoft.com

yuanzhil@andrew.cmu.edu

(Version 2)

ABSTRACT

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As *we pre-train larger models, full ?ne-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example ? deploying independent instances of ?ne-tuned models, each with 175B parameters, is prohibitively expensive. We propose Low-Rank Adaptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B ?ne-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than ?ne-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, no additional inference latency. We also provide an empirical investigation into rank-de?ciency in language model adaptation, which sheds light on the ef?cacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at https://github.com/microsoft/LoRA.*

1

INTRODUCTION

Pretrained

Weights

????×?

x

h

?= 0

?= ?(0, ?2)

?

?

Pretrained

Weights

????×?

x

f(x)

?

Figure 1: Our reparametriza-

tion. We only train A and B.

Many applications in natural language processing rely on adapt-

ing one large-scale, pre-trained language model to multiple down-

stream applications. Such adaptation is usually done via ?ne-tuning,

which updates all the parameters of the pre-trained model. The ma-

jor downside of ?ne-tuning is that the new model contains as many

parameters as in the original model. As larger models are trained

every few months, this changes from a mere ?inconvenience? for

GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a

critical deployment challenge for GPT-3 (Brown et al., 2020) with

175 billion trainable parameters.1

Many sought to mitigate this by adapting only some parameters or

learning external modules for new tasks. This way, we only need

to store and load a small number of task-speci?c parameters in ad-

dition to the pre-trained model for each task, greatly boosting the

operational ef?ciency when deployed. However, existing techniques

?Equal contribution.

0Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.

1While GPT-3 175B achieves non-trivial performance with few-shot learning, ?ne-tuning boosts its perfor-

mance signi?cantly as shown in Appendix A.

1

arXiv:2106.09685v2 [cs.CL] 16 Oct 2021

Method

WebNLG

BLEU?

MET?

TER?

U

S

A

U

S

A

U

S

A

GPT-2 Medium

Fine-Tune (354M)

27.7

64.2

46.5

.30

.45

.38

.76

.33

.53

AdapterL (0.37M)

45.1

54.5

50.2

.36

.39

.38

.46

.40

.43

AdapterL (11M)

48.3

60.4

54.9

.38

.43

.41

.45

.35

.39

FTTop2 (24M)

18.9

53.6

36.0

.23

.38

.31

.99

.49

.72

Pre?x (0.35M)

45.6

62.9

55.1

.38

.44

.41

.49

.35

.40

LoRA (0.35M)

46.7±.4

62.1±.2

55.3±.2

.38

.44

.41

.46

.33

.39

GPT-2 Large

Fine-Tune (774M)

43.1

65.3

55.5

.38

.46

.42

.53

.33

.42

AdapterL (0.88M)

49.8±.0

61.1±.0

56.0±.0

.38

.43

.41

.44

.35

.39

AdapterL (23M)

49.2±.1

64.7±.2

57.7±.1

.39

.46

.43

.46

.33

.39

Pre?x (0.77M)

47.7

63.4

56.3

.39

.45

.42

.48

.34

.40

LoRA (0.77M)

48.4±.3

64.0±.3

57.0±.1

.39

.45

.42

.45

.32

.38

Table 14: GPT-2 with different adaptation methods on WebNLG. The variances of MET and TER are less than 0.01 for all the experiments we ran. ?U? indicates unseen categories, ?S? indicates seen categories, and ?A? indicates all categories in the test set of WebNLG.

F.2

ADDITIONAL EXPERIMENTS ON GPT-3

We present additional runs on GPT-3 with different adaptation methods in Table 15. The focus is on identifying the trade-off between performance and the number of trainable parameters.

F.3

LOW-DATA REGIME

To evaluate the performance of different adaptation approaches in the low-data regime. we randomly sample 100, 1k and 10k training examples from the full training set of MNLI to form the low-data MNLI-n tasks. In Table 16, we show the performance of different adaptation approaches on MNLI-n. To our surprise, Pre?xEmbed and Pre?xLayer performs very poorly on MNLI-100 dataset, with Pre?xEmbed performing only slightly better than random chance (37.6% vs. 33.3%). Pre?xLayer performs better than Pre?xEmbed but is still signi?cantly worse than Fine-Tune or LoRA on MNLI-100. The gap between pre?x-based approaches and LoRA/Fine-tuning becomes smaller as we in-

crease the number of training examples, which might suggest that pre?x-based approaches are not

suitable for low-data tasks in GPT-3. LoRA achieves better performance than ?ne-tuning on both

MNLI-100 and MNLI-Full, and comparable results on MNLI-1k and MNLI-10K considering the

(±0.3) variance due to random seeds.

The training hyperparameters of different adaptation approaches on MNLI-n are reported in Ta-

ble 17. We use a smaller learning rate for Pre?xLayer on the MNLI-100 set, as the training loss does

not decrease with a larger learning rate.

G

MEASURING SIMILARITY BETWEEN SUBSPACES

In this paper we use the measure ?(A, B, i, j) = ?(U i

A, U j

B) = ?U i?

A UB?2

F

min{i,j}

to measure the subspace

similarity between two column orthonormal matrices U i

A ?Rd×i and U j

B ?Rd×j, obtained by

taking columns of the left singular matrices of A and B. We point out that this similarity is simply

a reverse of the standard Projection Metric that measures distance between subspaces Ham & Lee

(2008).

Batch Size

32

16

1

Sequence Length

512

256

128

|?|

0.5M

11M

11M

Fine-Tune/LoRA

1449.4±0.8

338.0±0.6

19.8±2.7

AdapterL

1482.0±1.0 (+2.2%)

354.8±0.5 (+5.0%)

23.9±2.1 (+20.7%)

AdapterH

1492.2±1.0 (+3.0%)

366.3±0.5 (+8.4%)

25.8±2.2 (+30.3%)

Table 1: Infernece latency of a single forward pass in GPT-2 medium measured in milliseconds, averaged over 100 trials. We use an NVIDIA Quadro RTX8000. ?|?|? denotes the number of trainable parameters in adapter layers. AdapterL and AdapterH are two variants of adapter tuning, which we describe in Section 5.1. The inference latency introduced by adapter layers can be signi?cant in an online, short-sequence-length scenario. See the full study in Appendix B.

4

OUR METHOD

We describe the simple design of LoRA and its practical bene?ts. The principles outlined here apply to any dense layers in deep learning models, though we only focus on certain weights in Transformer language models in our experiments as the motivating use case.

4.1

LOW-RANK-PARAMETRIZED UPDATE MATRICES

A neural network contains many dense layers which perform matrix multiplication. The weight matrices in these layers typically have full-rank. When adapting to a speci?c task, Aghajanyan et al. (2020) shows that the pre-trained language models have a low ?instrisic dimension? and can still learn ef?ciently despite a random projection to a smaller subspace. Inspired by this, we hypothesize the updates to the weights also have a low ?intrinsic rank? during adaptation. For a pre-trained weight matrix W0 ?Rd×k, we constrain its update by representing the latter with a low-rank decomposition W0 + ?W = W0 + BA, where B ?Rd×r, A ?Rr×k, and the rank r ?min(d, k).

During training, W0 is frozen and does not receive gradient updates, while A and B contain trainable

parameters. Note both W0 and ?W = BA are multiplied with the same input, and their respective

output vectors are summed coordinate-wise. For h = W0x, our modi?ed forward pass yields:

h = W0x + ?Wx = W0x + BAx

(3)

We illustrate our reparametrization in Figure 1. We use a random Gaussian initialization for A and

zero for B, so ?W = BA is zero at the beginning of training. We then scale ?Wx by ?

r , where ?

is a constant in r. When optimizing with Adam, tuning ? is roughly the same as tuning the learning

rate if we scale the initialization appropriately. As a result, we simply set ? to the ?rst r we try

and do not tune it. This scaling helps to reduce the need to retune hyperparameters when we vary

r (Yang & Hu, 2021).

A Generalization of Full Fine-tuning.

A more general form of ?ne-tuning allows the training of

a subset of the pre-trained parameters. LoRA takes a step further and does not require the accumu-

lated gradient update to weight matrices to have full-rank during adaptation. This means that when

applying LoRA to all weight matrices and training all biases2, we roughly recover the expressive-

ness of full ?ne-tuning by setting the LoRA rank r to the rank of the pre-trained weight matrices. In

other words, as we increase the number of trainable parameters 3, training LoRA roughly converges

to training the original model, while adapter-based methods converges to an MLP and pre?x-based

methods to a model that cannot take long input sequences.

No Additional Inference Latency.

When deployed in production, we can explicitly compute and

store W = W0 + BA and perform inference as usual. Note that both W0 and BA are in Rd×k.

When we need to switch to another downstream task, we can recover W0 by subtracting BA and

then adding a different B?A?, a quick operation with very little memory overhead. Critically, this

2They represent a negligible number of parameters compared to weights.

3An inevitability when adapting to hard tasks.

4

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. GPT

Understands, Too. arXiv:2103.10385 [cs], March 2021. URL http://arxiv.org/abs/

2103.10385. arXiv: 2103.10385.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike

Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Ilya Loshchilov and Frank Hutter.

Decoupled weight decay regularization.

arXiv preprint

arXiv:1711.05101, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Ef?cient low-rank hypercomplex adapter layers, 2021.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, et al. Dart: Open-domain structured data record to text generation. arXiv preprint arXiv:2007.02871, 2020.

Jekaterina Novikova, Ond?rej Du?sek, and Verena Rieser. The e2e dataset: New challenges for end-to-end generation. arXiv preprint arXiv:1706.09254, 2017.

Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi.

Generalization guaran-

tees for neural networks via harnessing the low-rank structure of the jacobian. arXiv preprint arXiv:1906.05392, 2019.

Jonas Pfeiffer, Aishwarya Kamath, Andreas R¨uckl´e, Kyunghyun Cho, and Iryna Gurevych. Adapter-fusion: Non-destructive task composition for transfer learning, 2021.

Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. Semi-orthogonal low-rank matrix factorization for deep neural networks. In Interspeech, pp. 3743?3747, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. pp. 12, a.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. pp. 24, b.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don?t know: Unanswerable questions for squad. CoRR, abs/1806.03822, 2018. URL http://arxiv.org/abs/1806.03822.

Sylvestre-Alvise Rebuf?, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. arXiv:1705.08045 [cs, stat], November 2017. URL http://arxiv.org/abs/1705.08045. arXiv: 1705.08045.

Andreas R¨uckl´e, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and

Iryna Gurevych. Adapterdrop: On the ef?ciency of adapters in transformers, 2020.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In 2013 IEEE international conference on acoustics, speech and signal processing, pp. 6655?6659. IEEE, 2013.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 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, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1170.

Current Question:

which WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Answer:

Evaluation 6:

### Evaluation of the Answer

#### Relevance: **Excellent**

The answer directly addresses the question, "Why Self-Attention?" It provides a thorough explanation of the benefits and mechanisms of self-attention in neural networks, particularly in Transformers. Every point in the answer is relevant and aligned with the question's focus.

#### Faithfulness: **Excellent**

The answer is strongly supported by the provided context, referencing foundational works such as Vaswani et al. (2017) and real-world applications like GPT and BERT. Claims about self-attention's efficiency, scalability, and versatility are consistent with the cited literature and context.

#### Completeness: **Excellent**

The answer covers all major aspects of the question, including long-range dependency modeling, parallelism, dynamic weighting, scalability, architectural simplicity, transfer learning, and versatility across domains. It provides a comprehensive overview without leaving critical points unexplored.

#### Clarity: **Excellent**

The answer is well-structured, with clear headings and concise explanations for each point. It uses accessible language while maintaining technical rigor, making it easy to understand for a wide range of readers.

#### Conciseness: **Good**

While the answer is detailed and avoids unnecessary repetition, it could be slightly streamlined. Some points overlap in their emphasis on efficiency and scalability, which could be condensed for brevity without losing depth.

### Final Ratings:

*- **Relevance:** Excellent*

*- **Faithfulness:** Excellent*

*- **Completeness:** Excellent*

*- **Clarity:** Excellent*

*- **Conciseness:** Good*

Weighted Score: 98.0

Question 6: Why Self-Attention?

Answer 6: Self-attention is a foundational mechanism in modern neural networks, especially in architectures like Transformers (Vaswani et al., 2017), due to its ability to efficiently capture relationships between different elements of a sequence. Here are the key reasons why self-attention is so valuable:

*### 1. **Captures Long-Range Dependencies***

- Traditional neural network architectures, such as recurrent neural networks (RNNs), struggle to model long-range dependencies because they process sequences sequentially, leading to limitations in context retention. Self-attention, by contrast, computes relationships between all elements in a sequence in parallel, allowing it to capture dependencies regardless of their distance in the input.

*### 2. **Parallelism and Efficiency***

- Unlike RNNs, which process inputs sequentially, self-attention can process all elements of a sequence simultaneously. This parallelism makes self-attention highly efficient, especially when scaled to large datasets and models, as seen in architectures like GPT and BERT.

### 3. **Dynamic Weighting**

   - Self-attention assigns importance (weights) dynamically to different parts of the input sequence based on their relevance to the current context. For instance, in a language model, self-attention can focus on specific words or phrases that are most relevant to predicting the next word, enabling nuanced understanding and generation.

### 4. **Handles Variable-Length Inputs**

   - Self-attention mechanisms are inherently adaptable to sequences of varying length because they rely on pairwise comparisons between sequence elements. This flexibility is crucial for tasks like text generation and summarization, where input lengths can vary widely.

### 5. **Improves Representational Power**

   - By attending to all parts of the sequence simultaneously, self-attention allows the model to create richer, context-sensitive representations of each token in the input. This is particularly beneficial for tasks like machine translation, question answering, and language understanding.

### 6. **Scalability**

   - Self-attention scales efficiently with the number of layers and parameters, making it suitable for large-scale models like GPT-3 and other state-of-the-art systems. Techniques like multi-head attention further improve its ability to capture diverse aspects of relationships within the data.

### 7. **Simplifies Architecture**

   - Self-attention removes the need for recurrence or convolution, simplifying the model architecture while achieving superior performance. This simplicity has allowed Transformer-based models to dominate benchmarks in natural language processing (NLP), such as GLUE and SuperGLUE.

### 8. **Enables Transfer Learning**

   - The contextual embeddings learned through self-attention in pre-trained models (e.g., BERT, GPT) can be fine-tuned efficiently *for downstream tasks, making self-attention a cornerstone of transfer learning in NLP.*

*### 9. **Versatility Across Domains***

- While self-attention originated in NLP, it has proven effective in other domains like computer vision (e.g., Vision Transformers) and speech processing, showcasing its adaptability and general-purpose utility.

In summary, self-attention is critical because it provides an efficient, scalable, and versatile way to model complex dependencies within data, enabling breakthroughs in tasks that require understanding and generating sequences. It is "all you need" for building powerful and general-purpose models, as highlighted by Vaswani et al. (2017).

Response Latency: 10.79 seconds

Evaluation Latency: 6.92 seconds

[Prompt used for Q6]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez,
?ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Proceedings of the 31st In-
ternational Conference on Neural Information Processing Systems, pp. 6000?6010, 2017.
Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman.
Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.
Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer
Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language
understanding systems, 2020.
Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments.
arXiv preprint arXiv:1805.12471, 2018.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sen-

tence understanding through inference.

In Proceedings of the 2018 Conference of the North

American Chapter of the Association for Computational Linguistics: Human Language Technolo-

gies, Volume 1 (Long Papers), pp. 1112?1122, New Orleans, Louisiana, June 2018. Association

for Computational Linguistics. doi: 10.18653/v1/N18-1101. URL https://www.aclweb.

org/anthology/N18-1101.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi,

Pierric Cistac, Tim Rault, R´emi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick

von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gug-

ger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art

natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in

Natural Language Processing: System Demonstrations, pp. 38?45, Online, October 2020. As-

sociation for Computational Linguistics. URL https://www.aclweb.org/anthology/

2020.emnlp-demos.6.

Greg Yang and Edward J. Hu.

Feature Learning in In?nite-Width Neural Networks.

arXiv:2011.14522 [cond-mat], May 2021. URL http://arxiv.org/abs/2011.14522.

arXiv: 2011.14522.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bit?t: Simple parameter-ef?cient ?ne-tuning

for transformer-based masked language-models, 2021.

Yu Zhang, Ekapol Chuangsuwanich, and James Glass. Extracting deep neural network bottleneck

features using low-rank matrix factorization. In 2014 IEEE international conference on acoustics,

speech and signal processing (ICASSP), pp. 185?189. IEEE, 2014.

Yong Zhao, Jinyu Li, and Yifan Gong. Low-rank plus diagonal adaptation for deep neural networks.

In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),

pp. 5005?5009. IEEE, 2016.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from

natural language using reinforcement learning. CoRR, abs/1709.00103, 2017. URL http://

arxiv.org/abs/1709.00103.

A

LARGE LANGUAGE MODELS STILL NEED PARAMETER UPDATES

Few-shot learning, or prompt engineering, is very advantageous when we only have a handful of *training samples. However, in practice, we can often afford to curate a few thousand or more training examples for performance-sensitive applications. As shown in Table 8, ?ne-tuning improves the model performance drastically compared to few-shot learning on datasets large and small. We take the GPT-3 few-shot result on RTE from the GPT-3 paper (Brown et al., 2020). For MNLI-matched, we use two demonstrations per class and six in-context examples in total.*

tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ?W suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

REFERENCES

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. arXiv:2012.13255 [cs], December 2020. URL http://arxiv.org/abs/2012.13255.

Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Ef?ciently, Going Beyond Kernels? In NeurIPS, 2019. Full version available at http://arxiv.org/abs/1905.10337.

Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. arXiv preprint arXiv:2001.04413, 2020a.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature puri?cation: How adversarial training performs robust deep learning. arXiv preprint arXiv:2005.10190, 2020b.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In ICML, 2019. Full version available at http://arxiv.org/abs/1811.
03962.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhari-wal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs], July 2020. URL http://arxiv.org/abs/2005.14165.

Jian-Feng Cai, Emmanuel J Cand`es, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. SIAM Journal on optimization, 20(4):1956?1982, 2010.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task

1: Semantic textual similarity multilingual and crosslingual focused evaluation. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), 2017. doi: 10.18653/ v1/s17-2001. URL http://dx.doi.org/10.18653/v1/S17-2001.

Ronan Collobert and Jason Weston. A uni?ed architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, ICML ?08, pp. 160?167, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc?Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019a.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs], May 2019b. URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005), 2005. URL https://aclanthology.org/I05-5002.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg challenge: Generating text from rdf data. In Proceedings of the 10th International Conference on Natural Language Generation, pp. 124?133, 2017.

guarantees that we do not introduce any additional latency during inference compared to a ?ne-tuned model by construction.

4.2 APPLYING LORA TO TRANSFORMER

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the number of trainable parameters. In the Transformer architecture, there are four weight matrices in the self-attention module (Wq, Wk, Wv, Wo) and two in the MLP module. We treat Wq (or Wk, Wv) as a single matrix of dimension dmodel × dmodel, even though the output dimension is usually sliced into attention heads. We limit our study to only adapting the attention weights for downstream tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity and parameter-ef?ciency.We further study the effect on adapting different types of attention weight *matrices in a Transformer in Section 7.1. We leave the empirical investigation of adapting the MLP*

layers, LayerNorm layers, and biases to a future work.

Practical Bene?ts and Limitations.

The most signi?cant bene?t comes from the reduction in

memory and storage usage. For a large Transformer trained with Adam, we reduce that VRAM

usage by up to 2/3 if r ?dmodel as we do not need to store the optimizer states for the frozen

parameters. On GPT-3 175B, we reduce the VRAM consumption during training from 1.2TB to

350GB. With r = 4 and only the query and value projection matrices being adapted, the checkpoint

size is reduced by roughly 10,000× (from 350GB to 35MB)4. This allows us to train with signi?-

cantly fewer GPUs and avoid I/O bottlenecks. Another bene?t is that we can switch between tasks

while deployed at a much lower cost by only swapping the LoRA weights as opposed to all the

parameters. This allows for the creation of many customized models that can be swapped in and out

on the ?y on machines that store the pre-trained weights in VRAM. We also observe a 25% speedup

during training on GPT-3 175B compared to full ?ne-tuning5 as we do not need to calculate the

gradient for the vast majority of the parameters.

LoRA also has its limitations. For example, it is not straightforward to batch inputs to different tasks

with different A and B in a single forward pass, if one chooses to absorb A and B into W to eliminate

additional inference latency. Though it is possible to not merge the weights and dynamically choose

the LoRA modules to use for samples in a batch for scenarios where latency is not critical.

5

EMPIRICAL EXPERIMENTS

We evaluate the downstream task performance of LoRA on RoBERTa (Liu et al., 2019), De-

BERTa (He et al., 2021), and GPT-2 (Radford et al., b), before scaling up to GPT-3 175B (Brown

et al., 2020). Our experiments cover a wide range of tasks, from natural language understanding

(NLU) to generation (NLG). Speci?cally, we evaluate on the GLUE (Wang et al., 2019) benchmark

for RoBERTa and DeBERTa. We follow the setup of Li & Liang (2021) on GPT-2 for a direct com-

parison and add WikiSQL (Zhong et al., 2017) (NL to SQL queries) and SAMSum (Gliwa et al.,

2019) (conversation summarization) for large-scale experiments on GPT-3. See Appendix C for

more details on the datasets we use. We use NVIDIA Tesla V100 for all experiments.

5.1

BASELINES

To compare with other baselines broadly, we replicate the setups used by prior work and reuse their

reported numbers whenever possible. This, however, means that some baselines might only appear *in certain experiments.*

Fine-Tuning (FT) is a common approach for adaptation. During ?ne-tuning, the model is initialized

to the pre-trained weights and biases, and all model parameters undergo gradient updates.A simple

variant is to update only some layers while freezing others. We include one such baseline reported

in prior work (Li & Liang, 2021) on GPT-2, which adapts just the last two layers (FTTop2).

4We still need the 350GB model during deployment; however, storing 100 adapted models only requires

*350GB + 35MB * 100 ?354GB as opposed to 100 * 350GB ?35TB.*

5For GPT-3 175B, the training throughput for full ?ne-tuning is 32.5 tokens/s per V100 GPU; with the same

number of weight shards for model parallelism, the throughput is 43.1 tokens/s per V100 GPU for LoRA.

5

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. When do neural

networks outperform kernel methods? arXiv preprint arXiv:2006.13409, 2020.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-

annotated dialogue dataset for abstractive summarization. CoRR, abs/1911.12237, 2019. URL

http://arxiv.org/abs/1911.12237.

Lars Grasedyck, Daniel Kressner, and Christine Tobler.

A literature survey of low-rank tensor

approximation techniques. GAMM-Mitteilungen, 36(1):53?78, 2013.

Jihun Ham and Daniel D. Lee. Grassmann discriminant analysis: a unifying view on subspace-based

learning. In ICML, pp. 376?383, 2008. URL https://doi.org/10.1145/1390156.

1390204.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. WARP: Word-level Adversarial

ReProgramming. arXiv:2101.00121 [cs], December 2020. URL http://arxiv.org/abs/

2101.00121. arXiv: 2101.00121.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert

with disentangled attention, 2021.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe,

Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Ef?cient Transfer Learning

for NLP. arXiv:1902.00751 [cs, stat], June 2019. URL http://arxiv.org/abs/1902.

00751.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks

with low rank expansions. arXiv preprint arXiv:1405.3866, 2014.

Mikhail Khodak, Neil Tenenholtz, Lester Mackey, and Nicol`o Fusi. Initialization and regularization *of factorized neural layers, 2021.*

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang,

Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional

computation and automatic sharding, 2020.

Brian Lester, Rami Al-Rfou, and Noah Constant. The Power of Scale for Parameter-Ef?cient Prompt

Tuning. arXiv:2104.08691 [cs], April 2021. URL http://arxiv.org/abs/2104.08691.

arXiv: 2104.08691.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the Intrinsic Di-

mension of Objective Landscapes.

arXiv:1804.08838 [cs, stat], April 2018a.

URL http:

//arxiv.org/abs/1804.08838. arXiv: 1804.08838.

Xiang Lisa Li and Percy Liang. Pre?x-Tuning: Optimizing Continuous Prompts for Generation.

arXiv:2101.00190 [cs], January 2021. URL http://arxiv.org/abs/2101.00190.

Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient

descent on structured data. In Advances in Neural Information Processing Systems, 2018.

Yuanzhi Li, Yingyu Liang, and Andrej Risteski. Recovery guarantee of weighted low-rank ap-

proximation via alternating minimization. In International Conference on Machine Learning, pp.

2358?2367. PMLR, 2016.

Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized

matrix sensing and neural networks with quadratic activations. In Conference On Learning The-

ory, pp. 2?47. PMLR, 2018b.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. Exploring versatile generative language model

via parameter-ef?cient transfer learning. In Findings of the Association for Computational Lin-

guistics: EMNLP 2020, pp. 441?459, Online, November 2020. Association for Computational

Linguistics.

doi: 10.18653/v1/2020.?ndings-emnlp.41.

URL https://aclanthology.

org/2020.findings-emnlp.41.

0.0

0.2

0.4

0.6

0.8

1.0

1

2

3

4

5

6

7

8

Layer 1

i

Wq

Wv

Wq

Wv

1

2

3

4

5

6

7

8

Layer 32

i

1

2

3

4

5

6

7

8

Layer 64

i

1

6

12

18

23

29

35

40

46

52

58

j

1

2

3

4

5

6

7

8

Layer 96

i

1

6

12

18

23

29

35

40

46

52

58

j

1

2

3

4

5

6

7

8

j

1

2

3

4

5

6

7

8

j

(Ar = 8, Ar = 64, i, j)

Figure 6: Normalized subspace similarity between the column vectors of Ar=8 and Ar=64 for both

?Wq and ?Wv from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

H.4

AMPLIFICATION FACTOR

One can naturally consider a feature ampli?cation factor as the ratio

??W ?F

?U ?W V ??F , where U and V

are the left- and right-singular matrices of the SVD decomposition of ?W. (Recall UU ?WV ?V *gives the ?projection? of W onto the subspace spanned by ?W.)*

Intuitively, when ?W mostly contains task-speci?c directions, this quantity measures how much of them are ampli?ed by ?W. As shown in Section 7.3, for r = 4, this ampli?cation factor is as large as 20. In other words, there are (generally speaking) four feature directions in each layer (out of the entire feature space from the pre-trained model W), that need to be ampli?ed by a very large factor 20, in order to achieve our reported accuracy for the downstream speci?c task. And, one should expect a very different set of feature directions to be ampli?ed for each different downstream task.

One may notice, however, for r = 64, this ampli?cation factor is only around 2, meaning that most directions learned in ?W with r = 64 are not being ampli?ed by much. This should not be surprising, and in fact gives evidence (once again) that the intrinsic rank needed to represent the ?task-speci?c directions? (thus for model adaptation) is low. In contrast, those directions in the rank-4 version of ?W (corresponding to r = 4) are ampli?ed by a much larger factor 20.

25

to maximize downstream performance? 2) Is the ?optimal? adaptation matrix ?W really rank-de?cient? If so, what is a good rank to use in practice? 3) What is the connection between ?W and W? Does ?W highly correlate with W? How large is ?W comparing to W?

We believe that our answers to question (2) and (3) shed light on the fundamental principles of using pre-trained language models for downstream tasks, which is a critical topic in NLP.

7.1

WHICH WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Given a limited parameter budget, which types of weights should we adapt with LoRA to obtain the best performance on downstream tasks? As mentioned in Section 4.2, we only consider weight matrices in the self-attention module. We set a parameter budget of 18M (roughly 35MB if stored in FP16) on GPT-3 175B, which corresponds to r = 8 if we adapt one type of attention weights or r = 4 if we adapt two types, for all 96 layers. The result is presented in Table 5.

# of Trainable Parameters = 18M

Weight Type

Wq

Wk

Wv

Wo

Wq, Wk

Wq, Wv

Wq, Wk, Wv, Wo

Rank r

8

8

8

8

4

4

2

WikiSQL (±0.5%)

70.4

70.0

73.0

73.2

71.4

73.7

73.7

MultiNLI (±0.1%)

91.0

90.8

91.0

91.3

91.3

91.3

91.7

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both Wq and Wv gives the best performance overall. We ?nd the standard deviation across random seeds to be consistent for a given dataset, which we report in the ?rst column.

Note that putting all the parameters in ?Wq or ?Wk results in signi?cantly lower performance, while adapting both Wq and Wv yields the best result. This suggests that even a rank of four captures enough information in ?W such that it is preferable to adapt more weight matrices than adapting a single type of weights with a larger rank.

7.2

WHAT IS THE OPTIMAL RANK r FOR LORA?

We turn our attention to the effect of rank r on model performance.

We adapt {Wq, Wv},

{Wq, Wk, Wv, Wc}, and just Wq for a comparison.

Weight Type

r = 1

r = 2

r = 4

r = 8

r = 64

WikiSQL(±0.5%)

Wq

68.8

69.6

70.5

70.4

70.0

Wq, Wv

73.4

73.3

73.7

73.8

73.5

Wq, Wk, Wv, Wo

74.1

73.7

74.0

74.0

73.9

MultiNLI (±0.1%)

Wq

90.7

90.9

91.1

90.7

90.7

Wq, Wv

91.3

91.4

91.3

91.6

91.4

Wq, Wk, Wv, Wo

91.2

91.7

91.7

91.5

91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a

rank as small as one suf?ces for adapting both Wq and Wv on these datasets while training Wq alone

needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

Table 6 shows that, surprisingly, LoRA already performs competitively with a very small r (more

so for {Wq, Wv} than just Wq). This suggests the update matrix ?W could have a very small

?intrinsic rank?.6 To further support this ?nding, we check the overlap of the subspaces learned by

different choices of r and by different random seeds. We argue that increasing r does not cover a

more meaningful subspace, which suggests that a low-rank adaptation matrix is suf?cient.

6However, we do not expect a small r to work for every task or dataset. Consider the following thought

experiment: if the downstream task were in a different language than the one used for pre-training, retraining

the entire model (similar to LoRA with r = dmodel) could certainly outperform LoRA with a small r.

often introduce inference latency (Houlsby et al., 2019; Rebuf?et al., 2017) by extending model

depth or reduce the model?s usable sequence length (Li & Liang, 2021; Lester et al., 2021; Ham-

bardzumyan et al., 2020; Liu et al., 2021) (Section 3). More importantly, these method often fail to

match the ?ne-tuning baselines, posing a trade-off between ef?ciency and model quality.

We take inspiration from Li et al. (2018a); Aghajanyan et al. (2020) which show that the learned *over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the*

change in weights during model adaptation also has a low ?intrinsic rank?, leading to our proposed

Low-Rank Adaptation (LoRA) approach. LoRA allows us to train some dense layers in a neural

network indirectly by optimizing rank decomposition matrices of the dense layers? change during

adaptation instead, while keeping the pre-trained weights frozen, as shown in Figure 1. Using GPT-3

175B as an example, we show that a very low rank (i.e., r in Figure 1 can be one or two) suf?ces even

when the full rank (i.e., d) is as high as 12,288, making LoRA both storage- and compute-ef?cient.

LoRA possesses several key advantages.

? A pre-trained model can be shared and used to build many small LoRA modules for dif-

ferent tasks. We can freeze the shared model and ef?ciently switch tasks by replacing the

matrices A and B in Figure 1, reducing the storage requirement and task-switching over-

head signi?cantly.

? LoRA makes training more ef?cient and lowers the hardware barrier to entry by up to 3

times when using adaptive optimizers since we do not need to calculate the gradients or

maintain the optimizer states for most parameters. Instead, we only optimize the injected,

much smaller low-rank matrices.

? Our simple linear design allows us to merge the trainable matrices with the frozen weights

when deployed, introducing no inference latency compared to a fully ?ne-tuned model, by

construction.

? LoRA is orthogonal to many prior methods and can be combined with many of them, such

as pre?x-tuning. We provide an example in Appendix E.

Terminologies and Conventions

We make frequent references to the Transformer architecture

and use the conventional terminologies for its dimensions.

We call the input and output di-

mension size of a Transformer layer dmodel.

We use Wq, Wk, Wv, and Wo to refer to the

query/key/value/output projection matrices in the self-attention module. W or W0 refers to a pre-

trained weight matrix and ?W its accumulated gradient update during adaptation. We use r to

denote the rank of a LoRA module. We follow the conventions set out by (Vaswani et al., 2017;

Brown et al., 2020) and use Adam (Loshchilov & Hutter, 2019; Kingma & Ba, 2017) for model

optimization and use a Transformer MLP feedforward dimension dffn = 4 × dmodel.

PROBLEM STATEMENT

While our proposal is agnostic to training objective, we focus on language modeling as our motivat-

ing use case. Below is a brief description of the language modeling problem and, in particular, the

maximization of conditional probabilities given a task-speci?c prompt.

Suppose we are given a pre-trained autoregressive language model P?(y|x) parametrized by ?.

For instance, P?(y|x) can be a generic multi-task learner such as GPT (Radford et al., b; Brown

et al., 2020) based on the Transformer architecture (Vaswani et al., 2017). Consider adapting this

pre-trained model to downstream conditional text generation tasks, such as summarization, machine

reading comprehension (MRC), and natural language to SQL (NL2SQL). Each downstream task is

represented by a training dataset of context-target pairs: Z = {(xi, yi)}i=1,..,N, where both xi and

yi are sequences of tokens. For example, in NL2SQL, xi is a natural language query and yi its

corresponding SQL command; for summarization, xi is the content of an article and yi its summary.

2

During full ?ne-tuning, the model is initialized to pre-trained weights ?0 and updated to ?0 + ??

by repeatedly following the gradient to maximize the conditional language modeling objective:

max

?

X

(x,y)?Z

|y|

X

t=1

log (P?(yt|x, y<t))

(1)

One of the main drawbacks for full ?ne-tuning is that for each downstream task, we learn a different

set of parameters ?? whose dimension |??| equals |?0|. Thus, if the pre-trained model is large

(such as GPT-3 with |?0| ?175 Billion), storing and deploying many independent instances of

?ne-tuned models can be challenging, if at all feasible.

In this paper, we adopt a more parameter-ef?cient approach, where the task-speci?c parameter

increment ?? = ??(?) is further encoded by a much smaller-sized set of parameters ? with

|?| ?|?0|. The task of ?nding ?? thus becomes optimizing over ?:

max

?

X

(x,y)?Z

|y|

X

t=1

log

p?0+??(?)(yt|x, y<t)

(2)

In the subsequent sections, we propose to use a low-rank representation to encode ?? that is both compute- and memory-ef?cient. When the pre-trained model is GPT-3 175B, the number of trainable parameters |?| can be as small as 0.01% of |?0|.

3

AREN?T EXISTING SOLUTIONS GOOD ENOUGH?

The problem we set out to tackle is by no means new. Since the inception of transfer learning, dozens of works have sought to make model adaptation more parameter- and compute-ef?cient. See Section 6 for a survey of some of the well-known works. Using language modeling as an example, there are two prominent strategies when it comes to ef?cient adaptations: adding adapter layers (Houlsby et al., 2019; Rebuf?et al., 2017; Pfeiffer et al., 2021; R¨uckl´e et al., 2020) or optimizing some forms of the input layer activations (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021). However, both strategies have their limitations, especially in a large-scale and latency-sensitive production scenario.

Adapter Layers Introduce Inference Latency

There are many variants of adapters. We focus

on the original design by Houlsby et al. (2019) which has two adapter layers per Transformer block and a more recent one by Lin et al. (2020) which has only one per block but with an additional LayerNorm (Ba et al., 2016). While one can reduce the overall latency by pruning layers or exploiting multi-task settings (R¨uckl´e et al., 2020; Pfeiffer et al., 2021), there is no direct ways to bypass the extra compute in adapter layers. This seems like a non-issue since adapter layers are designed to have few parameters (sometimes <1% of the original model) by having a small bottleneck dimension, which limits the FLOPs they can add. However, large neural networks rely on hardware parallelism to keep the latency low, and adapter layers have to be processed sequentially. This makes a difference in the online inference setting where the batch size is typically as small as one. In a *generic scenario without model parallelism, such as running inference on GPT-2 (Radford et al., b) medium on a single GPU, we see a noticeable increase in latency when using adapters, even with a very small bottleneck dimension (Table 1).*

This problem gets worse when we need to shard the model as done in Shoeybi et al. (2020); Lep-ikhin et al. (2020), because the additional depth requires more synchronous GPU operations such as AllReduce and Broadcast, unless we store the adapter parameters redundantly many times.

Directly Optimizing the Prompt is Hard

The other direction, as exempli?ed by pre?x tuning (Li & Liang, 2021), faces a different challenge. We observe that pre?x tuning is dif?cult to optimize and that its performance changes non-monotonically in trainable parameters, con?rming similar observations in the original paper. More fundamentally, reserving a part of the sequence length for adaptation necessarily reduces the sequence length available to process a downstream task, which we suspect makes tuning the prompt less performant compared to other methods. We defer the study on task performance to Section 5.

3

Method

WebNLG

BLEU?

MET?

TER?

U

S

A

U

S

A

U

S

A

GPT-2 Medium

Fine-Tune (354M)

27.7

64.2

46.5

.30

.45

.38

.76

.33

.53

AdapterL (0.37M)

45.1

54.5

50.2

.36

.39

.38

.46

.40

.43

AdapterL (11M)

48.3

60.4

54.9

.38

.43

.41

.45

.35

.39

FTTop2 (24M)

18.9

53.6

36.0

.23

.38

.31

.99

.49

.72

Pre?x (0.35M)

45.6

62.9

55.1

.38

.44

.41

.49

.35

.40

LoRA (0.35M)

46.7±.4

62.1±.2

55.3±.2

.38

.44

.41

.46

.33

.39

GPT-2 Large

Fine-Tune (774M)

43.1

65.3

55.5

.38

.46

.42

.53

.33

.42

AdapterL (0.88M)

49.8±.0

61.1±.0

56.0±.0

.38

.43

.41

.44

.35

.39

AdapterL (23M)

49.2±.1

64.7±.2

57.7±.1

.39

.46

.43

.46

.33

.39

Pre?x (0.77M)

47.7

63.4

56.3

.39

.45

.42

.48

.34

.40

LoRA (0.77M)

48.4±.3

64.0±.3

57.0±.1

.39

.45

.42

.45

.32

.38

Table 14: GPT-2 with different adaptation methods on WebNLG. The variances of MET and TER are less than 0.01 for all the experiments we ran. ?U? indicates unseen categories, ?S? indicates seen categories, and ?A? indicates all categories in the test set of WebNLG.

F.2

ADDITIONAL EXPERIMENTS ON GPT-3

We present additional runs on GPT-3 with different adaptation methods in Table 15. The focus is on identifying the trade-off between performance and the number of trainable parameters.

F.3

LOW-DATA REGIME

To evaluate the performance of different adaptation approaches in the low-data regime. we randomly sample 100, 1k and 10k training examples from the full training set of MNLI to form the low-data MNLI-n tasks. In Table 16, we show the performance of different adaptation approaches on MNLI-n. To our surprise, Pre?xEmbed and Pre?xLayer performs very poorly on MNLI-100 dataset, with Pre?xEmbed performing only slightly better than random chance (37.6% vs. 33.3%). Pre?xLayer performs better than Pre?xEmbed but is still signi?cantly worse than Fine-Tune or LoRA on MNLI-100. The gap between pre?x-based approaches and LoRA/Fine-tuning becomes smaller as we increase the number of training examples, which might suggest that pre?x-based approaches are not suitable for low-data tasks in GPT-3. LoRA achieves better performance than ?ne-tuning on both MNLI-100 and MNLI-Full, and comparable results on MNLI-1k and MNLI-10K considering the (±0.3) variance due to random seeds.

The training hyperparameters of different adaptation approaches on MNLI-n are reported in Table 17. We use a smaller learning rate for Pre?xLayer on the MNLI-100 set, as the training loss does not decrease with a larger learning rate.

G

MEASURING SIMILARITY BETWEEN SUBSPACES

In this paper we use the measure ?(A, B, i, j) = ?(U i

A, U j

B) = ?U i?

A UB?2

F

min{i,j}

to measure the subspace

similarity between two column orthonormal matrices U i

A ?Rd×i and U j

B ?Rd×j, obtained by

taking columns of the left singular matrices of A and B. We point out that this similarity is simply

a reverse of the standard Projection Metric that measures distance between subspaces Ham & Lee

(2008).

22

Current Question:

Why Self-Attention?

Answer:

Evaluation 7:

### Evaluation of the Answer:

***Relevance:** Excellent*

The answer directly addresses the question by identifying the model with the highest average score and stating the score.

***Faithfulness:** Excellent*

The answer is strictly supported by the given context, accurately referencing the data in Table 1.

***Completeness:** Excellent*

The answer covers all major aspects of the question, listing all models and their respective average scores before identifying the highest.

***Clarity:** Excellent*

The answer is well-structured and easy to understand, with clear enumeration and direct identification of the best-performing model.

***Conciseness:** Excellent*

The answer avoids unnecessary repetition and verbosity, providing the information succinctly.

Weighted Score: 100.0

Question 7: Among the five models in Table 1, which model has the highest average score and what is the highest average score?

*Answer 7: In **Table 1**, the five models listed are:*

*1. **Pre-OpenAI SOTA**: Average score = 74*

*2. **BiLSTM+ELMo+Attn**: Average score = 71*

*3. **OpenAI GPT**: Average score = 75.1*

*4. **BERT BASE (Single)**: Average score = 79.6*

*5. **BERT LARGE (Single)**: Average score = 82.1*

*Among these models, **BERT LARGE (Single)** has the highest average score, which is **82.1**.*

Response Latency: 5.63 seconds

Evaluation Latency: 4.48 seconds

[Prompt used for Q7]

You are a helpful assistant. Use the following history and context to answer.

```
    History:



    Context:
    [Sheet: Sheet1] Row 25: R.M.Reader(Ensemble)
             NaN
             NaN
            81.2
            87.9
            82.3
            88.5
             NaN
             NaN
             NaN
             NaN
             NaN
[Sheet: Sheet1] Row 29: BERT LARGE(Ensemble)
             NaN
             NaN
```

85.8

91.8

-

-

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 45: unet(Ensemble)

NaN

NaN

-

-

71.4

74.9

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 19: Top Leaderboard Systems (Dec 10th, 2018)

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1]

Unnamed: 0	*Unnamed: 1*	*Unnamed: 2*	*Unnamed: 3*	*Unnamed: 4*	*Unnamed: 5*	*Unnamed: 6*	*Unnamed: 7*	*Unnamed: 8*	*Unnamed: 9*	*Unnamed: 10*	*Unnamed: 11*
System	*NaN*	*NaN*	*MNLI-(m/mm)*	*QQP*	*QNLI*	*SST-2*	*CoLA*	*STS-B*	*MRPC*	*RTE*	*Average*
NaN	*NaN*	*NaN*	*392k*	*363k*	*108k*	*67k*	*8.5k*	*5.7k*	*3.5k*	*2.5k*	*-*
Pre-OpenAI SOTA	*NaN*	*NaN*	*80.6/80.1*	*66.1*	*82.3*	*93.2*	*35*	*81*	*86*	*61.7*	*74*
BiLSTM+ELMo+Attn	*NaN*	*NaN*	*76.4/76.1*	*64.8*	*79.8*	*90.4*	*36*	*73.3*	*84.9*	*56.8*	*71*
OpenAI GPT	*NaN*	*NaN*	*82.1/81.4*	*70.3*	*87.4*	*91.3*	*45.4*	*80*	*82.3*	*56*	*75.1*
BERT BASE(Single)	*NaN*	*NaN*	*84.6/83.4*	*71.2*	*90.5*	*93.5*	*52.1*	*85.8*	*88.9*	*66.4*	*79.6*
BERT BLARGE	*NaN*	*NaN*	*86.7/85.9*	*72.1*	*92.7*	*94.9*	*60.5*	*86.5*	*89.3*	*70.1*	*82.1*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples.The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components. *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

System NaN NaN Dev NaN Test NaN

NaN NaN NaN NaN NaN

NaN NaN NaN EM F1 EM F1

NaN NaN NaN NaN NaN

Top Leaderboard Systems (Dec 10th, 2018) NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

Human NaN NaN - - 82.3 91.2

NaN NaN NaN NaN NaN

#1 Ensemble - nlnet NaN NaN - - 86 91.7

NaN NaN NaN NaN NaN

#2 Ensemble - QANet NaN NaN - - 84.5 90.5

NaN NaN NaN NaN NaN

Published NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

BiDAF+ELMo(Single)	*NaN*	*NaN*	*-*	*85.6*	*-*	*85.8*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
R.M.Reader(Ensemble)	*NaN*	*NaN*	*81.2*	*87.9*	*82.3*	*88.5*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Ours	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT BASE(Single)	*NaN*	*NaN*	*80.8*	*88.5*	*-*	*-*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Single)	*NaN*	*NaN*	*84.1*	*90.9*	*-*	*-*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Ensemble)	*NaN*	*NaN*	*85.8*	*91.8*	*-*	*-*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Shl.+TriviaQA)	*NaN*	*NaN*	*84.2*	*91.1*	*85.1*	*91.8*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Ens.+TriviaQA)	*NaN*	*NaN*	*86.2*	*92.2*	*87.4*	*93.2*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*

NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN		

Table 2:

SQuAD 1.1 results. The BERT ensemble is 7x systems\nwhich use different pre-training checkpoints and fine-tuning\nseeds.

NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN		
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN		
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN		
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN		
System	NaN	NaN	Dev	NaN	Test	NaN
NaN	NaN	NaN	NaN	NaN		
NaN	NaN	NaN	EM	F1	EM	F1
NaN	NaN	NaN	NaN	NaN		

Top Leaderboard Systems (Dec 10th, 2018)	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Human	*NaN*	*NaN*	*86.3*	*89*	*86.9*	*89.5*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
#1 Single-MIR-MRC(F-Net)	*NaN*	*NaN*	*-*	*-*	*74.8*	*78*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
#2 Single-nlnet	*NaN*	*NaN*	*-*	*-*	*74.2*	*77.1*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Published	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
unet(Ensemble)	*NaN*	*NaN*	*-*	*-*	*71.4*	*74.9*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
SLQA+(Single)	*NaN*	*NaN*	*-*	*-*	*71.4*	*74.4*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Ours	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Single)	*NaN*	*NaN*	*78.7*	*81.9*	*80*						

83.1 NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components. NaN NaN

NaN NaN NaN NaN NaN NaN NaN NaN NaN

[Sheet: Sheet1] Row 40: Top Leaderboard Systems (Dec 10th, 2018)

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 22: #2 Ensemble - QANet

NaN

NaN

-

-

84.5

90.5

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 33: Table 2: SQuAD 1.1 results. The BERT ensemble i...

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 1: System

NaN

NaN

MNLI-(m/mm)

QQP

QNLI

SST-2

CoLA

STS-B

MRPC

RTE

Average

[Sheet: Sheet1] Row 21: #1 Ensemble - nlnet

NaN

NaN

-

-

86

91.7

NaN

NaN

NaN

NaN

NaN

Current Question:

Among the five models in Table 1, which model has the highest average score and what is the highest average score?

Answer:

Evaluation 8:

### Evaluation of the Answer

Relevance:** **Excellent

The answer directly addresses the question by explaining what LoRA fine-tuning is, its purpose, and its key features. It remains focused on the concept and provides a detailed explanation tailored to the topic.

Faithfulness:** **Excellent

The answer is strictly supported by the provided context. It accurately summarizes the details about LoRA, such as its use of low-rank matrices, parameter efficiency, application in Transformers, and scalability to large models like GPT-3.

Completeness:** **Excellent

The answer comprehensively covers all major aspects of LoRA fine-tuning, including its methodology, advantages, application, and practical benefits. It also gives examples to clarify its usage.

Clarity:** **Excellent

The answer is well-structured and easy to understand. It uses clear headings and concise explanations to break down complex concepts, ensuring readability.

Conciseness:** **Excellent

The answer avoids unnecessary repetition and verbosity while providing sufficient detail to explain the concept thoroughly. It balances depth and brevity effectively.

### Final Ratings

*- **Relevance:** Excellent*

*- **Faithfulness:** Excellent*

*- **Completeness:** Excellent*

*- **Clarity:** Excellent*

*- **Conciseness:** Excellent*

Weighted Score: 100.0

Question 8: what's LoRA fine-tuning?

Answer 8: LoRA (Low-Rank Adaptation) fine-tuning is a method designed to adapt pre-trained large language models (LLMs) to downstream tasks in a parameter-efficient and computationally efficient way. Instead of fine-tuning all the parameters of the pre-trained model, LoRA focuses on injecting trainable, low-rank matrices into the model's architecture, specifically targeting certain weight layers (e.g., attention weights in the Transformer architecture). This approach reduces the number of trainable parameters while maintaining high performance on downstream tasks.

*### **Key Features of LoRA Fine-Tuning:***

*1. **Low-Rank Updates:***

- LoRA assumes that the changes in model weights required for task-specific adaptation have a low intrinsic rank. Instead of directly updating the full weight matrices, LoRA learns rank-decomposed matrices.

- For example, if a weight matrix \(W \) is adapted, LoRA expresses the update as \(\Delta W = A \times B \), where \(A \) and \(B \) are low-rank matrices (with rank \(r \) much smaller than the dimension of \(W \)).

*2. **Frozen Pre-trained Weights:***

- The original weights of the pre-trained model remain frozen during fine-tuning. LoRA only adds and updates the low-rank matrices \(A \) and \(B \), which are significantly smaller in size compared to the full model parameters.

*3. **Parameter Efficiency:***

- LoRA significantly reduces the number of trainable parameters. For example, in GPT-3 175B, LoRA can reduce trainable

parameters by orders of magnitude. This makes it easier to adapt large models on hardware with limited memory.

4. **Performance and Scalability:**

   - LoRA achieves performance comparable to or better than full fine-tuning, even in low-data regimes. It scales well to large models like GPT-3 and works efficiently across various tasks.

5. **No Inference Latency Increase:**

   - During deployment, the low-rank matrices \(A \) and \(B \) can be merged into the original weight matrices, ensuring no additional inference latency compared to fully fine-tuned models.

6. **Orthogonality to Other Adaptation Methods:**

   - LoRA can be combined with other methods like prefix-tuning or adapter-tuning to further improve performance.

### **Advantages of LoRA Fine-Tuning:**

- **Reduced Computational and Memory Costs:** By freezing most of the model parameters, LoRA reduces the memory and computation required during training.

- **Task-Specific Adaptation:** It allows easy switching between tasks by loading different low-rank matrices for each task while keeping the core model unchanged.

- **Scalability:** LoRA scales effectively to extremely large models like GPT-3 175B.

- **Sample Efficiency:** It performs well in scenarios with limited training data, making it suitable for low-data regimes.

### **How LoRA is Applied in Transformers:**

- LoRA is typically applied to the **query (Wq)** and **value (Wv)** projection matrices in the self-attention modules of Transformers. These matrices are crucial for task-specific adaptation, and adapting them provides strong performance gains with minimal trainable parameters.

### **Example Use Case:**

For a large pre-trained model like GPT-3, instead of fine-tuning all 175 billion parameters, LoRA adds trainable matrices \(A \) and \(B \) to the query and value projection matrices, which might amount to only a few million trainable parameters. This enables efficient adaptation to tasks like summarization, text classification, or SQL query generation, while drastically reducing the hardware requirements.

In summary, LoRA fine-tuning is a powerful, efficient technique for adapting large language models to specific tasks, enabling researchers and practitioners to leverage the capabilities of massive pre-trained models with minimal computational overhead.

Response Latency: 11.55 seconds

Evaluation Latency: 6.34 seconds

[Prompt used for Q8]
You are a helpful assistant. Use the following history and context to answer.

History:

Context:

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.06

LR Schedule

Linear

RoBERTa base

LoRA

Batch Size

16

16

16

32

32

16

32

16

# Epochs

30

60

30

80

25

25

80

40

Learning Rate

5E-04

5E-04

4E-04

4E-04

4E-04

5E-04

5E-04

4E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

512

RoBERTa large

LoRA

Batch Size

4

4

4

4

4

4

8

8

# Epochs

10

10

20

20

10

20

20

30

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

128

512

128

512

512

512

512

RoBERTa large

LoRA?

Batch Size

4

# Epochs

10

10

20

20

10

20

20

10

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

RoBERTa large

AdptP (3M)?

Batch Size

32

# Epochs

10

20

20

20

10

20

20

20

Learning Rate

3E-05

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptP (0.8M)?

Batch Size

32

# Epochs

5

20

20

20

10

20

20

20

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

16

Max Seq. Len.

128

RoBERTa large

AdptH (6M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptH (0.8M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

8

Max Seq. Len.

128

Table 9: The hyperparameters we used for RoBERTa on the GLUE benchmark.

D.3

GPT-2

We train all of our GPT-2 models using AdamW (Loshchilov & Hutter, 2017) with a linear learning rate schedule for 5 epochs. We use the batch size, learning rate, and beam search beam size described in Li & Liang (2021). Accordingly, we also tune the above hyperparameters for LoRA. We report the mean over 3 random seeds; the result for each run is taken from the best epoch. The hyperparameters used for LoRA in GPT-2 are listed in Table 11. For those used for other baselines, see Li & Liang (2021).

D.4

GPT-3

For all GPT-3 experiments, we train using AdamW (Loshchilov & Hutter, 2017) for 2 epochs with a batch size of 128 samples and a weight decay factor of 0.1. We use a sequence length of 384 for

tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ?W suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

REFERENCES

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. arXiv:2012.13255 [cs], December 2020. URL *http://arxiv.org/abs/2012.13255.*

Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Ef?ciently, Going Beyond Kernels? In NeurIPS, 2019. Full version available at http://arxiv.org/abs/1905.10337.

Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. arXiv preprint arXiv:2001.04413, 2020a.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature puri?cation: How adversarial training performs robust deep learning. arXiv preprint arXiv:2005.10190, 2020b.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In ICML, 2019. Full version available at http://arxiv.org/abs/1811.

03962.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhari-wal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs], July 2020. URL http://arxiv.org/abs/2005.14165.

Jian-Feng Cai, Emmanuel J Cand`es, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. SIAM Journal on optimization, 20(4):1956?1982, 2010.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), 2017. doi: 10.18653/v1/s17-2001. URL http://dx.doi.org/10.18653/v1/S17-2001.

Ronan Collobert and Jason Weston. A uni?ed architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, ICML ?08, pp. 160?167, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc?Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019a.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs], May 2019b. URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005), 2005. URL https://aclanthology.org/I05-5002.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg challenge: Generating text from rdf data. In Proceedings of the 10th International Conference on Natural Language Generation, pp. 124?133, 2017.

Method	*Hyperparameters*	*# Trainable Parameters*	*WikiSQL*	*MNLI-m*
Fine-Tune	-	*175B*	*73.8*	*89.5*
Pre?xEmbed	*lp = 32, li = 8*	*0.4 M*	*55.9*	*84.9*
	lp = 64, li = 8	*0.9 M*	*58.7*	*88.1*
	lp = 128, li = 8	*1.7 M*	*60.6*	*88.0*

lp = 256, li = 8

3.2 M

63.1

88.6

lp = 512, li = 8

6.4 M

55.9

85.8

Pre?xLayer

lp = 2, li = 2

5.1 M

68.5

89.2

lp = 8, li = 0

10.1 M

69.8

88.2

lp = 8, li = 8

20.2 M

70.1

89.5

lp = 32, li = 4

44.1 M

66.4

89.6

lp = 64, li = 0

76.1 M

64.9

87.9

AdapterH

r = 1

7.1 M

71.9

89.8

r = 4

21.2 M

73.2

91.0

r = 8

40.1 M

73.2

91.5

r = 16

77.9 M

73.2

91.5

r = 64

304.4 M

72.6

91.5

LoRA

rv = 2

4.7 M

73.4

91.7

rq = rv = 1

4.7 M

73.4

91.3

rq = rv = 2

9.4 M

73.3

91.4

rq = rk = rv = ro = 1

9.4 M

74.1

91.2

rq = rv = 4

18.8 M

73.7

91.3

rq = rk = rv = ro = 2

18.8 M

73.7

91.7

rq = rv = 8

37.7 M

73.8

91.6

rq = rk = rv = ro = 4

37.7 M

74.0

91.7

rq = rv = 64

301.9 M

73.6

91.4

rq = rk = rv = ro = 64

603.8 M

73.9

91.4

LoRA+PE

rq = rv = 8, lp = 8, li = 4

37.8 M

75.0

91.4

rq = rv = 32, lp = 8, li = 4

151.1 M

75.9

91.1

rq = rv = 64, lp = 8, li = 4

302.1 M

76.2

91.3

LoRA+PL

rq = rv = 8, lp = 8, li = 4

52.8 M

72.9

90.2

Table 15: Hyperparameter analysis of different adaptation approaches on WikiSQL and MNLI. Both pre?x-embedding tuning (Pre?xEmbed) and pre?x-layer tuning (Pre?xLayer) perform worse as we increase the number of trainable parameters, while LoRA?s performance stabilizes. Performance is measured in validation accuracy.

Method	*MNLI(m)-100*	*MNLI(m)-1k*	*MNLI(m)-10k*	*MNLI(m)-392K*
GPT-3 (Fine-Tune)	*60.2*	*85.8*	*88.9*	*89.5*
GPT-3 (Pre?xEmbed)	*37.6*	*75.2*	*79.5*	*88.6*
GPT-3 (Pre?xLayer)	*48.3*	*82.5*	*85.9*	

89.6

GPT-3 (LoRA)

63.8

85.6

89.2

91.7

Table 16: Validation accuracy of different methods on subsets of MNLI using GPT-3 175B. MNLI-n describes a subset with n training examples. We evaluate with the full validation set. LoRA performs exhibits favorable sample-ef?ciency compared to other methods, including ?ne-tuning.

To be concrete, let the singular values of U i?

A U j

B to be ?1, ?2, · · · , ?p where p = min{i, j}. We

know that the Projection Metric Ham & Lee (2008) is de?ned as:

d(U i

A, U j

B) =

v

u

u

tp ?

p

X

i=1

?2

i ?[0, ?p]

23

often introduce inference latency (Houlsby et al., 2019; Rebuf?et al., 2017) by extending model depth or reduce the model?s usable sequence length (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021) (Section 3). More importantly, these method often fail to match the ?ne-tuning baselines, posing a trade-off between ef?ciency and model quality.

We take inspiration from Li et al. (2018a); Aghajanyan et al. (2020) which show that the learned over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the change in weights during model adaptation also has a low ?intrinsic rank?, leading to our proposed

Low-Rank Adaptation (LoRA) approach. LoRA allows us to train some dense layers in a neural

network indirectly by optimizing rank decomposition matrices of the dense layers? change during

adaptation instead, while keeping the pre-trained weights frozen, as shown in Figure 1. Using GPT-3

175B as an example, we show that a very low rank (i.e., r in Figure 1 can be one or two) suf?ces even

when the full rank (i.e., d) is as high as 12,288, making LoRA both storage- and compute-ef?cient.

LoRA possesses several key advantages.

? A pre-trained model can be shared and used to build many small LoRA modules for dif-

ferent tasks. We can freeze the shared model and ef?ciently switch tasks by replacing the

matrices A and B in Figure 1, reducing the storage requirement and task-switching over-

head signi?cantly.

? LoRA makes training more ef?cient and lowers the hardware barrier to entry by up to 3

times when using adaptive optimizers since we do not need to calculate the gradients or

maintain the optimizer states for most parameters. Instead, we only optimize the injected,

much smaller low-rank matrices.

? Our simple linear design allows us to merge the trainable matrices with the frozen weights

when deployed, introducing no inference latency compared to a fully ?ne-tuned model, by

construction.

? LoRA is orthogonal to many prior methods and can be combined with many of them, such

as pre?x-tuning. We provide an example in Appendix E.

Terminologies and Conventions

We make frequent references to the Transformer architecture

and use the conventional terminologies for its dimensions.

We call the input and output di-

mension size of a Transformer layer dmodel.

We use Wq, Wk, Wv, and Wo to refer to the

query/key/value/output projection matrices in the self-attention module. W or W0 refers to a pre-

trained weight matrix and ?W its accumulated gradient update during adaptation. We use r to

denote the rank of a LoRA module. We follow the conventions set out by (Vaswani et al., 2017;

Brown et al., 2020) and use Adam (Loshchilov & Hutter, 2019; Kingma & Ba, 2017) for model

optimization and use a Transformer MLP feedforward dimension dffn = 4 × dmodel.

2

PROBLEM STATEMENT

While our proposal is agnostic to training objective, we focus on language modeling as our motivat-

ing use case. Below is a brief description of the language modeling problem and, in particular, the

maximization of conditional probabilities given a task-speci?c prompt.

Suppose we are given a pre-trained autoregressive language model P?(y|x) parametrized by ?.

For instance, P?(y|x) can be a generic multi-task learner such as GPT (Radford et al., b; Brown

et al., 2020) based on the Transformer architecture (Vaswani et al., 2017). Consider adapting this

pre-trained model to downstream conditional text generation tasks, such as summarization, machine

reading comprehension (MRC), and natural language to SQL (NL2SQL). Each downstream task is

represented by a training dataset of context-target pairs: Z = {(xi, yi)}i=1,..,N, where both xi and

yi are sequences of tokens. For example, in NL2SQL, xi is a natural language query and yi its

corresponding SQL command; for summarization, xi is the content of an article and yi its summary.

2

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.1

LR Schedule

Linear

DeBERTa XXL

LoRA

Batch Size

8

8

32

4

6

8

4

4

# Epochs

5

16

30

10

8

11

11

10

Learning Rate

1E-04

6E-05

2E-04

1E-04

1E-04

1E-04

2E-04

2E-04

Weight Decay

0

0.01

0.01

0

0.01

0.01

0.01

0.1

CLS Dropout

0.15

0

0

0.1

0.1

0.2

0.2

0.2

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

256

128

128

64

512

320

320

128

Table 10: The hyperparameters for DeBERTa XXL on tasks included in the GLUE benchmark.

Dataset

E2E

WebNLG

DART

Training

Optimizer

AdamW

Weight Decay

0.01

0.01

0.0

Dropout Prob

0.1

0.1

0.0

Batch Size

8

# Epoch

5

Warmup Steps

500

Learning Rate Schedule

Linear

Label Smooth

0.1

0.1

0.0

Learning Rate

0.0002

Adaptation

rq = rv = 4

LoRA ?

32

Inference

Beam Size

10

Length Penalty

0.9

0.8

0.8

no repeat ngram size

4

Table 11: The hyperparameters for GPT-2 LoRA on E2E, WebNLG and DART.

WikiSQL (Zhong et al., 2017), 768 for MNLI (Williams et al., 2018), and 2048 for SAMSum (Gliwa *et al., 2019). We tune learning rate for all method-dataset combinations. See Section D.4 for more*

details on the hyperparameters used. For pre?x-embedding tuning, we ?nd the optimal lp and li

to be 256 and 8, respectively, totalling 3.2M trainable parameters. We use lp = 8 and li = 8 for

pre?x-layer tuning with 20.2M trainable parameters to obtain the overall best performance. We

present two parameter budgets for LoRA: 4.7M (rq = rv = 1 or rv = 2) and 37.7M (rq = rv = 8

or rq = rk = rv = ro = 2). We report the best validation performance from each run. The training

hyperparameters used in our GPT-3 experiments are listed in Table 12.

E

COMBINING LORA WITH PREFIX TUNING

LoRA can be naturally combined with existing pre?x-based approaches. In this section, we evaluate

two combinations of LoRA and variants of pre?x-tuning on WikiSQL and MNLI.

LoRA+Pre?xEmbed (LoRA+PE) combines LoRA with pre?x-embedding tuning, where we insert

lp + li special tokens whose embeddings are treated as trainable parameters. For more on pre?x-

embedding tuning, see Section 5.1.

LoRA+Pre?xLayer (LoRA+PL) combines LoRA with pre?x-layer tuning. We also insert lp + li

special tokens; however, instead of letting the hidden representations of these tokens evolve natu-

guarantees that we do not introduce any additional latency during inference compared to a ?ne-tuned

model by construction.

4.2

APPLYING LORA TO TRANSFORMER

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the

number of trainable parameters. In the Transformer architecture, there are four weight matrices in

the self-attention module (Wq, Wk, Wv, Wo) and two in the MLP module. We treat Wq (or Wk, Wv)

as a single matrix of dimension dmodel × dmodel, even though the output dimension is usually sliced

into attention heads. We limit our study to only adapting the attention weights for downstream

tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity

and parameter-ef?ciency.We further study the effect on adapting different types of attention weight

matrices in a Transformer in Section 7.1. We leave the empirical investigation of adapting the MLP

layers, LayerNorm layers, and biases to a future work.

Practical Bene?ts and Limitations.

The most signi?cant bene?t comes from the reduction in

memory and storage usage. For a large Transformer trained with Adam, we reduce that VRAM *usage by up to 2/3 if r ?dmodel as we do not need to store the optimizer states for the frozen parameters. On GPT-3 175B, we reduce the VRAM consumption during training from 1.2TB to 350GB. With r = 4 and only the query and value projection matrices being adapted, the checkpoint size is reduced by roughly 10,000× (from 350GB to 35MB)4. This allows us to train with signi?-cantly fewer GPUs and avoid I/O bottlenecks. Another bene?t is that we can switch between tasks while deployed at a much lower cost by only swapping the LoRA weights as opposed to all the parameters. This allows for the creation of many customized models that can be swapped in and out on the ?y on machines that store the pre-trained weights in VRAM. We also observe a 25% speedup during training on GPT-3 175B compared to full ?ne-tuning5 as we do not need to calculate the gradient for the vast majority of the parameters.*

LoRA also has its limitations. For example, it is not straightforward to batch inputs to different tasks with different A and B in a single forward pass, if one chooses to absorb A and B into W to eliminate additional inference latency. Though it is possible to not merge the weights and dynamically choose the LoRA modules to use for samples in a batch for scenarios where latency is not critical.

5

EMPIRICAL EXPERIMENTS

We evaluate the downstream task performance of LoRA on RoBERTa (Liu et al., 2019), De-BERTa (He et al., 2021), and GPT-2 (Radford et al., b), before scaling up to GPT-3 175B (Brown et al., 2020). Our experiments cover a wide range of tasks, from natural language understanding (NLU) to generation (NLG). Speci?cally, we evaluate on the GLUE (Wang et al., 2019) benchmark for RoBERTa and DeBERTa. We follow the setup of Li & Liang (2021) on GPT-2 for a direct com-parison and add WikiSQL (Zhong et al., 2017) (NL to SQL queries) and SAMSum (Gliwa et al., 2019) (conversation summarization) for large-scale experiments on GPT-3. See Appendix C for more details on the datasets we use. We use NVIDIA Tesla V100 for all experiments.

5.1

BASELINES

To compare with other baselines broadly, we replicate the setups used by prior work and reuse their reported numbers whenever possible. This, however, means that some baselines might only appear in certain experiments.

Fine-Tuning (FT) is a common approach for adaptation. During ?ne-tuning, the model is initialized to the pre-trained weights and biases, and all model parameters undergo gradient updates.A simple variant is to update only some layers while freezing others. We include one such baseline reported in prior work (Li & Liang, 2021) on GPT-2, which adapts just the last two layers (FTTop2).

4We still need the 350GB model during deployment; however, storing 100 adapted models only requires

*350GB + 35MB * 100 ?354GB as opposed to 100 * 350GB ?35TB.*

5For GPT-3 175B, the training throughput for full ?ne-tuning is 32.5 tokens/s per V100 GPU; with the same

number of weight shards for model parallelism, the throughput is 43.1 tokens/s per V100 GPU for LoRA.

5

Hyperparameters

Fine-Tune

PreEmbed

PreLayer

BitFit

AdapterH

LoRA

Optimizer

AdamW

Batch Size

128

# Epoch

2

Warmup Tokens

250,000

LR Schedule

Linear

Learning Rate

5.00E-06

5.00E-04

1.00E-04

1.6E-03

1.00E-04

2.00E-04

Table 12: The training hyperparameters used for different GPT-3 adaption methods. We use the

same hyperparameters for all datasets after tuning learning rate.

rally, we replace them after every Transformer block with an input agnostic vector. Thus, both the

embeddings and subsequent Transformer block activations are treated as trainable parameters. For more on pre?x-layer tuning, see Section 5.1.

In Table 15, we show the evaluation results of LoRA+PE and LoRA+PL on WikiSQL and MultiNLI.

First of all, LoRA+PE signi?cantly outperforms both LoRA and pre?x-embedding tuning on

WikiSQL, which indicates that LoRA is somewhat orthogonal to pre?x-embedding tuning. On

MultiNLI, the combination of LoRA+PE doesn?t perform better than LoRA, possibly because LoRA

on its own already achieves performance comparable to the human baseline. Secondly, we notice

that LoRA+PL performs slightly worse than LoRA even with more trainable parameters. We at-

tribute this to the fact that pre?x-layer tuning is very sensitive to the choice of learning rate and thus

makes the optimization of LoRA weights more dif?cult in LoRA+PL.

F

ADDITIONAL EMPIRICAL EXPERIMENTS

F.1

ADDITIONAL EXPERIMENTS ON GPT-2

We also repeat our experiment on DART (Nan et al., 2020) and WebNLG (Gardent et al., 2017)

following the setup of Li & Liang (2021). The result is shown in Table 13. Similar to our result

on E2E NLG Challenge, reported in Section 5, LoRA performs better than or at least on-par with

pre?x-based approaches given the same number of trainable parameters.

Method

# Trainable

DART

Parameters

BLEU?

MET?

TER?

GPT-2 Medium

Fine-Tune

354M

46.2

0.39

0.46

AdapterL

0.37M

42.4

0.36

0.48

AdapterL

11M

45.2

0.38

0.46

FTTop2

24M

41.0

0.34

0.56

PrefLayer

0.35M

46.4

0.38

0.46

LoRA

0.35M

47.1±.2

0.39

0.46

GPT-2 Large

Fine-Tune

774M

47.0

0.39

0.46

AdapterL

0.88M

45.7±.1

0.38

0.46

AdapterL

23M

47.1±.1

0.39

0.45

PrefLayer

0.77M

46.7

0.38

0.45

LoRA

0.77M

47.5±.1

0.39

0.45

Table 13: GPT-2 with different adaptation methods on DART. The variances of MET and TER are less than 0.01 for all adaption approaches.

Method

WebNLG

BLEU?

MET?

TER?

U

S

A

U

S

A

U

S

A

GPT-2 Medium

Fine-Tune (354M)

27.7

64.2

46.5

.30

.45

.38

.76

.33

.53

AdapterL (0.37M)

45.1

54.5

50.2

.36

.39

.38

.46

.40

.43

AdapterL (11M)

48.3

60.4

54.9

.38

.43

.41

.45

.35

.39

FTTop2 (24M)

18.9

53.6

36.0

.23

.38

.31

.99

.49

.72

Pre?x (0.35M)

45.6

62.9

55.1

.38

.44

.41

.49

.35

.40

LoRA (0.35M)

46.7±.4

62.1±.2

55.3±.2

.38

.44

.41

.46

.33

.39

GPT-2 Large

Fine-Tune (774M)

43.1

65.3

55.5

.38

.46

.42

.53

.33

.42

AdapterL (0.88M)

49.8±.0

61.1±.0

56.0±.0

.38

.43

.41

.44

.35

.39

AdapterL (23M)

49.2±.1

64.7±.2

57.7±.1

.39

.46

.43

.46

.33

.39

Pre?x (0.77M)

47.7

63.4

56.3

.39

.45

.42

.48

.34

.40

LoRA (0.77M)

48.4±.3

64.0±.3

57.0±.1

.39

.45

.42

.45

.32

.38

Table 14: GPT-2 with different adaptation methods on WebNLG. The variances of MET and TER are less than 0.01 for all the experiments we ran. ?U? indicates unseen categories, ?S? indicates seen categories, and ?A? indicates all categories in the test set of WebNLG.

F.2 ADDITIONAL EXPERIMENTS ON GPT-3

We present additional runs on GPT-3 with different adaptation methods in Table 15. The focus is on identifying the trade-off between performance and the number of trainable parameters.

F.3 LOW-DATA REGIME

To evaluate the performance of different adaptation approaches in the low-data regime. we randomly sample 100, 1k and 10k training examples from the full training set of MNLI to form the low-data MNLI-n tasks. In Table 16, we show the performance of different adaptation approaches on MNLI-n. To our surprise, Pre?xEmbed and Pre?xLayer performs very poorly on MNLI-100 dataset, with Pre?xEmbed performing only slightly better than random chance (37.6% vs. 33.3%). Pre?xLayer performs better than Pre?xEmbed but is still signi?cantly worse than Fine-Tune or LoRA on MNLI-100. The gap between pre?x-based approaches and LoRA/Fine-tuning becomes smaller as we increase the number of training examples, which might suggest that pre?x-based approaches are not suitable for low-data tasks in GPT-3. LoRA achieves better performance than ?ne-tuning on both MNLI-100 and MNLI-Full, and comparable results on MNLI-1k and MNLI-10K considering the (±0.3) variance due to random seeds.

The training hyperparameters of different adaptation approaches on MNLI-n are reported in Ta-

ble 17. We use a smaller learning rate for Pre?xLayer on the MNLI-100 set, as the training loss does

not decrease with a larger learning rate.

G

MEASURING SIMILARITY BETWEEN SUBSPACES

In this paper we use the measure ?(A, B, i, j) = ?(U i

A, U j

B) = ?U i?

A UB?2

F

min{i,j}

to measure the subspace

similarity between two column orthonormal matrices U i

A ?Rd×i and U j

B ?Rd×j, obtained by

taking columns of the left singular matrices of A and B. We point out that this similarity is simply

a reverse of the standard Projection Metric that measures distance between subspaces Ham & Lee

(2008).

22

to maximize downstream performance? 2) Is the ?optimal? adaptation matrix ?W really rank-

de?cient? If so, what is a good rank to use in practice? 3) What is the connection between ?W and

W? Does ?W highly correlate with W? How large is ?W comparing to W?

We believe that our answers to question (2) and (3) shed light on the fundamental principles of using

pre-trained language models for downstream tasks, which is a critical topic in NLP.

7.1

WHICH WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Given a limited parameter budget, which types of weights should we adapt with LoRA to obtain

the best performance on downstream tasks? As mentioned in Section 4.2, we only consider weight

matrices in the self-attention module. We set a parameter budget of 18M (roughly 35MB if stored

in FP16) on GPT-3 175B, which corresponds to r = 8 if we adapt one type of attention weights or

r = 4 if we adapt two types, for all 96 layers. The result is presented in Table 5.

# of Trainable Parameters = 18M

Weight Type

Wq

Wk

Wv

Wo

Wq, Wk

Wq, Wv

Wq, Wk, Wv, Wo

Rank r

8

8

8

8

4

4

2

WikiSQL (±0.5%)

70.4

70.0

73.0

73.2

71.4

73.7

73.7

MultiNLI (±0.1%)

91.0

90.8

91.0

91.3

91.3

91.3

91.7

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both Wq and Wv gives the best performance overall. We ?nd the standard deviation across random seeds to be

consistent for a given dataset, which we report in the ?rst column.

Note that putting all the parameters in ?Wq or ?Wk results in signi?cantly lower performance,

while adapting both Wq and Wv yields the best result. This suggests that even a rank of four

captures enough information in ?W such that it is preferable to adapt more weight matrices than

adapting a single type of weights with a larger rank.

7.2

WHAT IS THE OPTIMAL RANK r FOR LORA?

We turn our attention to the effect of rank r on model performance.

We adapt {Wq, Wv},

{Wq, Wk, Wv, Wc}, and just Wq for a comparison.

Weight Type

r = 1

r = 2

r = 4

r = 8

r = 64

WikiSQL(±0.5%)

Wq

68.8

69.6

70.5

70.4

70.0

Wq, Wv

73.4

73.3

73.7

73.8

73.5

Wq, Wk, Wv, Wo

74.1

73.7

74.0

74.0

73.9

MultiNLI (±0.1%)

Wq

90.7

90.9

91.1

90.7

90.7

Wq, Wv

91.3

91.4

91.3

91.6

91.4

Wq, Wk, Wv, Wo

91.2

91.7

91.7

91.5

91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a

rank as small as one suf?ces for adapting both Wq and Wv on these datasets while training Wq alone

needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

Table 6 shows that, surprisingly, LoRA already performs competitively with a very small r (more

so for {Wq, Wv} than just Wq). This suggests the update matrix ?W could have a very small

?intrinsic rank?.6 To further support this ?nding, we check the overlap of the subspaces learned by

different choices of r and by different random seeds. We argue that increasing r does not cover a

more meaningful subspace, which suggests that a low-rank adaptation matrix is suf?cient.

6However, we do not expect a small r to work for every task or dataset. Consider the following thought

experiment: if the downstream task were in a different language than the one used for pre-training, retraining

the entire model (similar to LoRA with r = dmodel) could certainly outperform LoRA with a small r.

Model&Method	*# Trainable*	*WikiSQL*	*MNLI-m*	*SAMSum*
	Parameters	*Acc. (%)*	*Acc. (%)*	*R1/R2/RL*
GPT-3 (FT)	*175,255.8M*	*73.8*	*89.5*	*52.0/28.0/44.5*
GPT-3 (BitFit)	*14.2M*	*71.3*	*91.0*	*51.3/27.4/43.5*
GPT-3 (PreEmbed)	*3.2M*	*63.1*	*88.6*	*48.3/24.2/40.5*
GPT-3 (PreLayer)	*20.2M*	*70.1*	*89.5*	*50.8/27.3/43.5*
GPT-3 (AdapterH)	*7.1M*	*71.9*	*89.8*	

				53.0/28.9/44.8
GPT-3 (AdapterH)	*40.1M*	*73.2*	*91.5*	*53.2/29.0/45.1*
GPT-3 (LoRA)	*4.7M*	*73.4*	*91.7*	*53.8/29.8/45.9*
GPT-3 (LoRA)	*37.7M*	*74.0*	*91.6*	*53.4/29.2/45.1*

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full ?ne-tuning. The results on WikiSQL have a ?uctuation around ±0.5%, MNLI-m around ±0.1%, and SAMSum around ±0.2/±0.2/±0.1 for the three metrics.

5.5 SCALING UP TO GPT-3 175B

As a ?nal stress test for LoRA, we scale up to GPT-3 with 175 billion parameters. Due to the high training cost, we only report the typical standard deviation for a given task over random seeds, as opposed to providing one for every entry. See Section D.4 for details on the hyperparameters used.

As shown in Table 4, LoRA matches or exceeds the ?ne-tuning baseline on all three datasets. Note that not all methods bene?t monotonically from having more trainable parameters, as shown in Figure 2. We observe a signi?cant performance drop when we use more than 256 special tokens for pre?x-embedding tuning or more than 32 special tokens for pre?x-layer tuning. This corroborates similar observations in Li & Liang (2021). While a thorough investigation into this phenomenon is out-of-scope for this work, we suspect that having more special tokens causes the input distribution to shift further away from the pre-training data distribution. Separately, we investigate the *performance of different adaptation approaches in the low-data regime in Section F.3.*

6

7

8

9

10

11

log10 # Trainable Parameters

0.55

0.60

0.65

0.70

0.75

Validation Accuracy

WikiSQL

Method

Fine-Tune

PrefixEmbed

PrefixLayer

Adapter(H)

LoRA

6

7

8

9

10

11

log10 # Trainable Parameters

0.84

0.86

0.88

0.90

0.92

MultiNLI-matched

Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance. See Section F.2 for more details on the plotted data points.

6

RELATED WORKS

Transformer Language Models.

Transformer (Vaswani et al., 2017) is a sequence-to-sequence

architecture that makes heavy use of self-attention. Radford et al. (a) applied it to autoregressive lan-

guage modeling by using a stack of Transformer decoders. Since then, Transformer-based language

models have dominated NLP, achieving the state-of-the-art in many tasks. A new paradigm emerged

with BERT (Devlin et al., 2019b) and GPT-2 (Radford et al., b) ? both are large Transformer lan-

8

Current Question:

what's LoRA fine-tuning?

Answer:

Evaluation 9:

### Evaluation of the Answer

#### **Relevance:** Excellent

The answer directly addresses the comparison between LoRA and full fine-tuning, covering all key aspects such as parameters, performance, efficiency, and trade-offs.

#### **Faithfulness:** Excellent

The answer is strictly supported by the provided context, drawing on specific data points like trainable parameters, accuracy results, and efficiency implications from the tables and text.

#### **Completeness:** Excellent

The answer covers all major aspects of the comparison, such as trainable parameters, performance, efficiency, low-data regime effectiveness, and trade-offs, providing a thorough analysis.

#### **Clarity:** Excellent

The explanation is well-structured and easy to understand, with clear headings and concise descriptions for each aspect of the comparison.

#### **Conciseness:** Excellent

The answer avoids unnecessary repetition, providing detailed yet succinct information on each point.

### Final Ratings

- **Relevance:** Excellent

- **Faithfulness:** Excellent

*- **Completeness:** Excellent*

*- **Clarity:** Excellent*

*- **Conciseness:** Excellent*

Weighted Score: 100.0

Question 9: compare it to full fine-tuning

*Answer 9: When comparing **LoRA** (Low-Rank Adaptation) to **full fine-tuning**, the main differences lie in the **number of trainable parameters**, **performance**, and **efficiency**. Here's a detailed comparison based on the provided data:*

---

*### **Number of Trainable Parameters***

*- **Full Fine-Tuning**: Requires updating all parameters of the model. For large models like GPT-3 (175B parameters), this approach involves modifying 175B parameters, which is computationally expensive and memory-intensive.*

*- **LoRA**: Focuses on low-rank updates to specific weight matrices, drastically reducing the number of trainable parameters. For example, LoRA configurations often use only **4.7M - 37.7M trainable parameters**, which is **0.01%-0.02%** of the size of GPT-3.*

---

*### **Performance***

*- **Full Fine-Tuning**: Provides high performance across tasks due to complete model adaptation. For instance, on MNLI-m (matched dataset), full fine-tuning achieves **89.5% accuracy**.*

*- **LoRA**: Achieves comparable or even superior performance with significantly fewer parameters. On MNLI-m:*

* - LoRA (rv = 2) achieves **91.7% accuracy**, outperforming full fine-tuning.*

- Even LoRA with smaller configurations (e.g., rv = 1 or rq = rv = 2) achieves competitive results (91.3%-91.7%).

### **Efficiency**

- **Full Fine-Tuning**: Requires storing and deploying separate fine-tuned models for each task, which is challenging for large models like GPT-3. For example, deploying multiple large fine-tuned models is not scalable due to storage and inference latency issues.

- **LoRA**: Is highly parameter-efficient and introduces minimal additional inference latency. LoRA updates can be merged into the original model weights during inference, making it much more practical for production use.

### **Low-Data Regime**

- **Full Fine-Tuning**: Performs well with sufficient training data but may not be optimal in low-data scenarios.

- **LoRA**: Demonstrates **better sample efficiency** compared to full fine-tuning in low-data regimes. For example, on MNLI-100 (100 training examples), LoRA achieves **63.8% accuracy**, outperforming fine-tuning's **60.2% accuracy**.

### **Trade-offs**

- **Full Fine-Tuning**: Best suited for scenarios where task-specific performance is paramount and resources are not a concern.

- **LoRA**: Ideal for resource-constrained environments, enabling adaptation without the need for extensive computational or storage overhead while maintaining competitive performance.

---

*### **Conclusion***

*LoRA offers **significant advantages** over full fine-tuning in terms of efficiency, scalability, and sample efficiency, while maintaining comparable or even superior performance in many cases. For large-scale models like GPT-3, LoRA is especially attractive due to its ability to adapt models with minimal hardware cost and storage requirements.*

Response Latency: 11.75 seconds

Evaluation Latency: 5.26 seconds

[Prompt used for Q9]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

Method

Hyperparameters

# Trainable Parameters

WikiSQL

MNLI-m

Fine-Tune

-

175B

73.8

89.5

Pre?xEmbed

lp = 32, li = 8

0.4 M

55.9

84.9

lp = 64, li = 8

0.9 M

58.7

88.1

lp = 128, li = 8

1.7 M

60.6

88.0

lp = 256, li = 8

3.2 M

63.1

88.6

lp = 512, li = 8

6.4 M

55.9

85.8

Pre?xLayer

lp = 2, li = 2

5.1 M

68.5

89.2

lp = 8, li = 0

10.1 M

69.8

88.2

lp = 8, li = 8

20.2 M

70.1

89.5

lp = 32, li = 4

44.1 M

66.4

89.6

lp = 64, li = 0

76.1 M

64.9

87.9

AdapterH

r = 1

7.1 M

71.9

89.8

r = 4

21.2 M

73.2

91.0

r = 8

40.1 M

73.2

91.5

r = 16

77.9 M

73.2

91.5

r = 64

304.4 M

72.6

91.5

LoRA

rv = 2

4.7 M

73.4

91.7

rq = rv = 1

4.7 M

73.4

91.3

rq = rv = 2

9.4 M

73.3

91.4

rq = rk = rv = ro = 1

9.4 M

74.1

91.2

rq = rv = 4

18.8 M

73.7

91.3

rq = rk = rv = ro = 2

18.8 M

73.7

91.7

rq = rv = 8

37.7 M

73.8

91.6

rq = rk = rv = ro = 4

37.7 M

74.0

91.7

rq = rv = 64

301.9 M

73.6

91.4

rq = rk = rv = ro = 64

603.8 M

73.9

91.4

LoRA+PE

rq = rv = 8, lp = 8, li = 4

37.8 M

75.0

91.4

rq = rv = 32, lp = 8, li = 4

151.1 M

75.9

91.1

rq = rv = 64, lp = 8, li = 4

302.1 M

76.2

91.3

LoRA+PL

rq = rv = 8, lp = 8, li = 4

52.8 M

72.9

90.2

Table 15: Hyperparameter analysis of different adaptation approaches on WikiSQL and MNLI. Both pre?x-embedding tuning (Pre?xEmbed) and pre?x-layer tuning (Pre?xLayer) perform worse as we increase the number of trainable parameters, while LoRA?s performance stabilizes. Performance is measured in validation accuracy.

Method

MNLI(m)-100

MNLI(m)-1k

MNLI(m)-10k

MNLI(m)-392K

GPT-3 (Fine-Tune)

60.2

85.8

88.9

89.5

GPT-3 (Pre?xEmbed)

37.6

75.2

79.5

88.6

GPT-3 (Pre?xLayer)

48.3

82.5

85.9

89.6

GPT-3 (LoRA)

63.8

85.6

89.2

91.7

Table 16: Validation accuracy of different methods on subsets of MNLI using GPT-3 175B. MNLI-

n describes a subset with n training examples. We evaluate with the full validation set. LoRA

performs exhibits favorable sample-ef?ciency compared to other methods, including ?ne-tuning.

To be concrete, let the singular values of U i?

A U j

B to be ?1, ?2, · · · , ?p where p = min{i, j}. We

know that the Projection Metric Ham & Lee (2008) is de?ned as:

d(U i

A, U j

B) =

v

u

u

tp ?

p

X

i=1

?2

i ?[0, ?p]

23

tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are

there more principled ways to do it? 4) Finally, the rank-de?ciency of ?W suggests that W could *be rank-de?cient as well, which can also be a source of inspiration for future works.*

REFERENCES

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. arXiv:2012.13255 [cs], December 2020. URL http://arxiv.org/abs/2012.13255.

Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Ef?ciently, Going Beyond Kernels? In NeurIPS, 2019. Full version available at http://arxiv.org/abs/1905.10337.

Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. arXiv preprint arXiv:2001.04413, 2020a.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature puri?cation: How adversarial training performs robust deep learning. arXiv preprint arXiv:2005.10190, 2020b.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In ICML, 2019. Full version available at http://arxiv.org/abs/1811.

03962.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhari-wal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs], July 2020. URL http://arxiv.org/abs/2005.14165.

Jian-Feng Cai, Emmanuel J Cand`es, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. SIAM Journal on optimization, 20(4):1956?1982, 2010.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), 2017. doi: 10.18653/ v1/s17-2001. URL http://dx.doi.org/10.18653/v1/S17-2001.

Ronan Collobert and Jason Weston. A uni?ed architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, ICML ?08, pp. 160?167, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc?Aurelio Ranzato, and Nando de Freitas. Predicting

parameters in deep learning, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep

bidirectional transformers for language understanding, 2019a.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep

Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs], May 2019b.

URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases.

In Proceedings of the Third International Workshop on Paraphrasing (IWP2005), 2005. URL

https://aclanthology.org/I05-5002.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg

challenge: Generating text from rdf data. In Proceedings of the 10th International Conference on

Natural Language Generation, pp. 124?133, 2017.

Method

WebNLG

BLEU?

MET?

TER?

U

S

A

U

S

A

U

S

A

GPT-2 Medium

Fine-Tune (354M)

27.7

64.2

46.5

.30

.45

.38

.76

.33

.53

AdapterL (0.37M)

45.1

54.5

50.2

.36

.39

.38

.46

.40

.43

AdapterL (11M)

48.3

60.4

54.9

.38

.43

.41

.45

.35

.39

FTTop2 (24M)

18.9

53.6

36.0

.23

.38

.31

.99

.49

.72

Pre?x (0.35M)

45.6

62.9

55.1

.38

.44

.41

.49

.35

.40

LoRA (0.35M)

46.7±.4

62.1±.2

55.3±.2

.38

.44

.41

.46

.33

.39

GPT-2 Large

Fine-Tune (774M)

43.1

65.3

55.5

.38

.46

.42

.53

.33

.42

AdapterL (0.88M)

49.8±.0

61.1±.0

56.0±.0

.38

.43

.41

.44

.35

.39

AdapterL (23M)

49.2±.1

64.7±.2

57.7±.1

.39

.46

.43

.46

.33

.39

Pre?x (0.77M)

47.7

63.4

56.3

.39

.45

.42

.48

.34

.40

LoRA (0.77M)

48.4±.3

64.0±.3

57.0±.1

.39

.45

.42

.45

.32

.38

Table 14: GPT-2 with different adaptation methods on WebNLG. The variances of MET and TER are less than 0.01 for all the experiments we ran. ?U? indicates unseen categories, ?S? indicates seen categories, and ?A? indicates all categories in the test set of WebNLG.

F.2

ADDITIONAL EXPERIMENTS ON GPT-3

We present additional runs on GPT-3 with different adaptation methods in Table 15. The focus is on identifying the trade-off between performance and the number of trainable parameters.

F.3

LOW-DATA REGIME

To evaluate the performance of different adaptation approaches in the low-data regime. we randomly sample 100, 1k and 10k training examples from the full training set of MNLI to form the low-data MNLI-n tasks. In Table 16, we show the performance of different adaptation approaches on MNLI-n. To our surprise, Pre?xEmbed and Pre?xLayer performs very poorly on MNLI-100 dataset, with Pre?xEmbed performing only slightly better than random chance (37.6% vs. 33.3%). Pre?xLayer performs better than Pre?xEmbed but is still signi?cantly worse than Fine-Tune or LoRA on MNLI-100. The gap between pre?x-based approaches and LoRA/Fine-tuning becomes smaller as we increase the number of training examples, which might suggest that pre?x-based approaches are not suitable for low-data tasks in GPT-3. LoRA achieves better performance than ?ne-tuning on both MNLI-100 and MNLI-Full, and comparable results on MNLI-1k and MNLI-10K considering the (±0.3) variance due to random seeds.

The training hyperparameters of different adaptation approaches on MNLI-n are reported in Table 17. We use a smaller learning rate for Pre?xLayer on the MNLI-100 set, as the training loss does not decrease with a larger learning rate.

G

MEASURING SIMILARITY BETWEEN SUBSPACES

In this paper we use the measure ?(A, B, i, j) = ?(U i

A, U j

B) = ?U i?

A UB?2

F

min{i,j}

to measure the subspace

similarity between two column orthonormal matrices U i

A ?Rd×i and U j

B ?Rd×j, obtained by

taking columns of the left singular matrices of A and B. We point out that this similarity is simply

a reverse of the standard Projection Metric that measures distance between subspaces Ham & Lee

(2008).

22

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.06

LR Schedule

Linear

RoBERTa base

LoRA

Batch Size

16

16

16

32

32

16

32

16

# Epochs

30

60

30

80

25

25

80

40

Learning Rate

5E-04

5E-04

4E-04

4E-04

4E-04

5E-04

5E-04

4E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

512

RoBERTa large

LoRA

Batch Size

4

4

4

4

4

4

8

8

# Epochs

10

10

20

20

10

20

20

30

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

128

512

128

512

512

512

512

RoBERTa large

LoRA?

Batch Size

4

# Epochs

10

10

20

20

10

20

20

10

Learning Rate

3E-04

4E-04

3E-04

2E-04

2E-04

3E-04

4E-04

2E-04

LoRA Con?g.

rq = rv = 8

LoRA ?

16

Max Seq. Len.

128

RoBERTa large

AdptP (3M)?

Batch Size

32

# Epochs

10

20

20

20

10

20

20

20

Learning Rate

3E-05

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptP (0.8M)?

Batch Size

32

# Epochs

5

20

20

20

10

20

20

20

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

16

Max Seq. Len.

128

RoBERTa large

AdptH (6M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-05

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

64

Max Seq. Len.

128

RoBERTa large

AdptH (0.8M)?

Batch Size

32

# Epochs

10

5

10

10

5

20

20

10

Learning Rate

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

3E-04

Bottleneck r

8

Max Seq. Len.

128

Table 9: The hyperparameters we used for RoBERTa on the GLUE benchmark.

D.3

GPT-2

We train all of our GPT-2 models using AdamW (Loshchilov & Hutter, 2017) with a linear learning rate schedule for 5 epochs. We use the batch size, learning rate, and beam search beam size described in Li & Liang (2021). Accordingly, we also tune the above hyperparameters for LoRA. We report the mean over 3 random seeds; the result for each run is taken from the best epoch. The hyperparameters used for LoRA in GPT-2 are listed in Table 11. For those used for other baselines, see Li & Liang (2021).

D.4

GPT-3

For all GPT-3 experiments, we train using AdamW (Loshchilov & Hutter, 2017) for 2 epochs with a batch size of 128 samples and a weight decay factor of 0.1. We use a sequence length of 384 for

During full ?ne-tuning, the model is initialized to pre-trained weights ?0 and updated to ?0 + ?? by repeatedly following the gradient to maximize the conditional language modeling objective:

max

?

X

(x,y)?Z

|y|

X

t=1

log (P?(yt|x, y<t))

(1)

One of the main drawbacks for full ?ne-tuning is that for each downstream task, we learn a different

set of parameters ?? whose dimension |??| equals |?0|. Thus, if the pre-trained model is large

(such as GPT-3 with |?0| ?175 Billion), storing and deploying many independent instances of

?ne-tuned models can be challenging, if at all feasible.

In this paper, we adopt a more parameter-ef?cient approach, where the task-speci?c parameter

increment ?? = ??(?) is further encoded by a much smaller-sized set of parameters ? with

|?| ?|?0|. The task of ?nding ?? thus becomes optimizing over ?:

max

?

X

(x,y)?Z

|y|

X

t=1

log

p?0+??(?)(yt|x, y<t)

(2)

In the subsequent sections, we propose to use a low-rank representation to encode ?? that is both

compute- and memory-ef?cient. When the pre-trained model is GPT-3 175B, the number of train-

able parameters |?| can be as small as 0.01% of |?0|.

3

AREN?T EXISTING SOLUTIONS GOOD ENOUGH?

The problem we set out to tackle is by no means new. Since the inception of transfer learning, dozens

of works have sought to make model adaptation more parameter- and compute-ef?cient. See Sec-

tion 6 for a survey of some of the well-known works. Using language modeling as an example, there

are two prominent strategies when it comes to ef?cient adaptations: adding adapter layers (Houlsby

et al., 2019; Rebuf?et al., 2017; Pfeiffer et al., 2021; R¨uckl´e et al., 2020) or optimizing some forms

of the input layer activations (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020;

Liu et al., 2021). However, both strategies have their limitations, especially in a large-scale and

latency-sensitive production scenario.

Adapter Layers Introduce Inference Latency

There are many variants of adapters. We focus

on the original design by Houlsby et al. (2019) which has two adapter layers per Transformer block

and a more recent one by Lin et al. (2020) which has only one per block but with an additional

LayerNorm (Ba et al., 2016). While one can reduce the overall latency by pruning layers or exploit-

ing multi-task settings (R¨uckl´e et al., 2020; Pfeiffer et al., 2021), there is no direct ways to bypass

the extra compute in adapter layers. This seems like a non-issue since adapter layers are designed

to have few parameters (sometimes <1% of the original model) by having a small bottleneck di-

mension, which limits the FLOPs they can add. However, large neural networks rely on hardware

parallelism to keep the latency low, and adapter layers have to be processed sequentially. This makes

a difference in the online inference setting where the batch size is typically as small as one. In a

generic scenario without model parallelism, such as running inference on GPT-2 (Radford et al., b)

medium on a single GPU, we see a noticeable increase in latency when using adapters, even with a

very small bottleneck dimension (Table 1).

This problem gets worse when we need to shard the model as done in Shoeybi et al. (2020); Lep-

ikhin et al. (2020), because the additional depth requires more synchronous GPU operations such as

AllReduce and Broadcast, unless we store the adapter parameters redundantly many times.

Directly Optimizing the Prompt is Hard

The other direction, as exempli?ed by pre?x tuning (Li

& Liang, 2021), faces a different challenge. We observe that pre?x tuning is dif?cult to optimize

and that its performance changes non-monotonically in trainable parameters, con?rming similar

observations in the original paper. More fundamentally, reserving a part of the sequence length for

adaptation necessarily reduces the sequence length available to process a downstream task, which

we suspect makes tuning the prompt less performant compared to other methods. We defer the study

on task performance to Section 5.

Model & Method # Trainable

Parameters MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B Avg.

*RoBbase (FT)**

125.0M

87.6

94.8

90.2

63.6

92.8

91.9

78.7

91.2

86.4

*RoBbase (BitFit)**

0.1M

84.7

93.7

92.7

62.0

91.8

84.0

81.5

90.8

85.2

*RoBbase (AdptD)**

0.3M 87.1±.0 94.2±.1 88.5±1.1 60.8±.4 93.1±.1 90.2±.0 71.5±2.7 89.7±.3 84.4

*RoBbase (AdptD)**

0.9M 87.3±.1 94.7±.3 88.4±.1

62.6±.9 93.0±.2 90.6±.0 75.9±2.2 90.3±.1 85.4

RoBbase (LoRA)

0.3M 87.5±.3 95.1±.2 89.7±.7 63.4±1.2 93.3±.3 90.8±.1 86.6±.7

91.5±.2 87.2

*RoBlarge (FT)**

355.0M

90.2

96.4

90.9

68.0

94.7

92.2

86.6

92.4

88.9

RoBlarge (LoRA)

0.8M 90.6±.2 96.2±.5 90.9±1.2 68.2±1.9 94.9±.3 91.6±.1 87.4±2.5 92.6±.2 89.0

RoBlarge (AdptP)?

3.0M 90.2±.3 96.1±.3 90.2±.7 68.3±1.0 94.8±.2 91.9±.1 83.8±2.9 92.1±.7 88.4

RoBlarge (AdptP)?

0.8M 90.5±.3 96.6±.2 89.7±1.2 67.8±2.5 94.8±.3 91.7±.2 80.1±2.9 91.9±.4 87.9

RoBlarge (AdptH)?

6.0M 89.9±.5 96.2±.3 88.7±2.9 66.5±4.4 94.7±.2 92.1±.1 83.4±1.1 91.0±1.7 87.8

RoBlarge (AdptH)?

0.8M 90.3±.3 96.3±.5 87.7±1.7 66.3±2.0 94.7±.2 91.5±.1 72.9±2.9 91.5±.5 86.4

RoBlarge (LoRA)?

0.8M 90.6±.2 96.2±.5 90.2±1.0 68.2±1.9 94.8±.3 91.6±.2 85.2±1.1 92.3±.5 88.6

*DeBXXL (FT)**

1500.0M

91.8

97.2

92.0

72.0

96.0

92.7

93.9

92.9

91.1

DeBXXL (LoRA)

4.7M 91.9±.2 96.9±.2 92.6±.6 72.4±1.1 96.0±.1 92.9±.1 94.9±.4

93.0±.2 91.3

*Table 2: RoBERTabase, RoBERTalarge, and DeBERTaXXL with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew?s correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. ? indicates runs con?gured in a setup similar to Houlsby et al. (2019) for a fair comparison.*

Bias-only or BitFit is a baseline where we only train the bias vectors while freezing everything else. Contemporarily, this baseline has also been studied by BitFit (Zaken et al., 2021).

Pre?x-embedding tuning (PreEmbed) inserts special tokens among the input tokens. These special tokens have trainable word embeddings and are generally not in the model?s vocabulary. Where to place such tokens can have an impact on performance. We focus on ?pre?xing?, which prepends such tokens to the prompt, and ?in?xing?, which appends to the prompt; both are discussed in Li & Liang (2021). We use lp (resp. li) denote the number of pre?x (resp. in?x) tokens. The number of trainable parameters is |?| = dmodel × (lp + li).

Pre?x-layer tuning (PreLayer) is an extension to pre?x-embedding tuning. Instead of just learning the word embeddings (or equivalently, the activations after the embedding layer) for some special tokens, we learn the activations after every Transformer layer. The activations computed from previous layers are simply replaced by trainable ones. The resulting number of trainable parameters is |?| = L × dmodel × (lp + li), where L is the number of Transformer layers.

Adapter tuning as proposed in Houlsby et al. (2019) inserts adapter layers between the self-attention module (and the MLP module) and the subsequent residual connection. There are two fully connected layers with biases in an adapter layer with a nonlinearity in between. We call this original design AdapterH. Recently, Lin et al. (2020) proposed a more ef?cient design with the adapter layer applied only after the MLP module and after a LayerNorm. We call it AdapterL. This is very similar to another deign proposed in Pfeiffer et al. (2021), which we call AdapterP. We also include another baseline call AdapterDrop (R¨uckl´e et al., 2020) which drops some adapter layers for greater ef?ciency (AdapterD). We cite numbers from prior works whenever possible to maximize the number of baselines we compare with; they are in rows with an asterisk (*) in the ?rst column. In all cases, we have |?| = ?LAdpt ×(2×dmodel ×r+r+dmodel)+2× ?LLN ×dmodel where ?LAdpt is the number of adapter layers and ?LLN the number of trainable LayerNorms (e.g., in AdapterL).

LoRA adds trainable pairs of rank decomposition matrices in parallel to existing weight matrices. As mentioned in Section 4.2, we only apply LoRA to Wq and Wv in most experiments for simplicity.

The number of trainable parameters is determined by the rank r and the shape of the original weights:

|?| = 2 × ?LLoRA × dmodel × r, where ?LLoRA is the number of weight matrices we apply LoRA to.

6

0.0

0.2

0.4

0.6

0.8

1.0

1

2

3

4

5

6

7

8

Layer 1

i

Wq

Wv

Wq

Wv

1

2

3

4

5

6

7

8

Layer 32

i

1

2

3

4

5

6

7

8

Layer 64

i

1

6

12

18

23

29

35

40

46

52

58

j

1

2

3

4

5

6

7

8

Layer 96

i

1

6

12

18

23

29

35

40

46

52

58

j

1

2

3

4

5

6

7

8

j

1

2

3

4

5

6

7

8

j

(Ar = 8, Ar = 64, i, j)

Figure 6: Normalized subspace similarity between the column vectors of Ar=8 and Ar=64 for both

?Wq and ?Wv from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

H.4

AMPLIFICATION FACTOR

One can naturally consider a feature ampli?cation factor as the ratio

??W ?F

?U ?W V ??F , where U and V

are the left- and right-singular matrices of the SVD decomposition of ?W. (Recall UU ?WV ?V

gives the ?projection? of W onto the subspace spanned by ?W.)

Intuitively, when ?W mostly contains task-speci?c directions, this quantity measures how much of

them are ampli?ed by ?W. As shown in Section 7.3, for r = 4, this ampli?cation factor is as large

as 20. In other words, there are (generally speaking) four feature directions in each layer (out of the

entire feature space from the pre-trained model W), that need to be ampli?ed by a very large factor

20, in order to achieve our reported accuracy for the downstream speci?c task. And, one should

expect a very different set of feature directions to be ampli?ed for each different downstream task.

One may notice, however, for r = 64, this ampli?cation factor is only around 2, meaning that

most directions learned in ?W with r = 64 are not being ampli?ed by much. This should not

be surprising, and in fact gives evidence (once again) that the intrinsic rank needed to represent

the ?task-speci?c directions? (thus for model adaptation) is low. In contrast, those directions in the

rank-4 version of ?W (corresponding to r = 4) are ampli?ed by a much larger factor 20.

25

guage models trained on a large amount of text ? where ?ne-tuning on task-speci?c data after pre-

training on general domain data provides a signi?cant performance gain compared to training on

task-speci?c data directly. Training larger Transformers generally results in better performance and

remains an active research direction. GPT-3 (Brown et al., 2020) is the largest single Transformer

language model trained to-date with 175B parameters.

Prompt Engineering and Fine-Tuning.

While GPT-3 175B can adapt its behavior with just a

few additional training examples, the result depends heavily on the input prompt (Brown et al.,

2020). This necessitates an empirical art of composing and formatting the prompt to maximize a

model?s performance on a desired task, which is known as prompt engineering or prompt hacking.

Fine-tuning retrains a model pre-trained on general domains to a speci?c task Devlin et al. (2019b);

Radford et al. (a). Variants of it include learning just a subset of the parameters Devlin et al. (2019b);

Collobert & Weston (2008), yet practitioners often retrain all of them to maximize the downstream

performance. However, the enormity of GPT-3 175B makes it challenging to perform ?ne-tuning in

the usual way due to the large checkpoint it produces and the high hardware barrier to entry since it *has the same memory footprint as pre-training.*

Parameter-Ef?cient Adaptation.

Many have proposed inserting adapter layers between existing

layers in a neural network (Houlsby et al., 2019; Rebuf?et al., 2017; Lin et al., 2020). Our method

uses a similar bottleneck structure to impose a low-rank constraint on the weight updates. The

key functional difference is that our learned weights can be merged with the main weights during

inference, thus not introducing any latency, which is not the case for the adapter layers (Section 3).

A comtenporary extension of adapter is COMPACTER (Mahabadi et al., 2021), which essentially

parametrizes the adapter layers using Kronecker products with some predetermined weight sharing

scheme. Similarly, combining LoRA with other tensor product-based methods could potentially

improve its parameter ef?ciency, which we leave to future work. More recently, many proposed

optimizing the input word embeddings in lieu of ?ne-tuning, akin to a continuous and differentiable

generalization of prompt engineering (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al.,

2020; Liu et al., 2021). We include comparisons with Li & Liang (2021) in our experiment section.

However, this line of works can only scale up by using more special tokens in the prompt, which

take up available sequence length for task tokens when positional embeddings are learned.

Low-Rank Structures in Deep Learning.

Low-rank structure is very common in machine learn-

ing. A lot of machine learning problems have certain intrinsic low-rank structure (Li et al., 2016;

Cai et al., 2010; Li et al., 2018b; Grasedyck et al., 2013). Moreover, it is known that for many

deep learning tasks, especially those with a heavily over-parametrized neural network, the learned

neural network will enjoy low-rank properties after training (Oymak et al., 2019). Some prior works

even explicitly impose the low-rank constraint when training the original neural network (Sainath

et al., 2013; Povey et al., 2018; Zhang et al., 2014; Jaderberg et al., 2014; Zhao et al., 2016; Kho-

dak et al., 2021; Denil et al., 2014); however, to the best of our knowledge, none of these works

considers low-rank update to a frozen model for adaptation to downstream tasks. In theory liter-

ature, it is known that neural networks outperform other classical learning methods, including the

corresponding (?nite-width) neural tangent kernels (Allen-Zhu et al., 2019; Li & Liang, 2018) when

the underlying concept class has certain low-rank structure (Ghorbani et al., 2020; Allen-Zhu & Li,

2019; Allen-Zhu & Li, 2020a). Another theoretical result in Allen-Zhu & Li (2020b) suggests that

low-rank adaptations can be useful for adversarial training. In sum, we believe that our proposed

low-rank adaptation update is well-motivated by the literature.

UNDERSTANDING THE LOW-RANK UPDATES

Given the empirical advantage of LoRA, we hope to further explain the properties of the low-rank adaptation learned from downstream tasks. Note that the low-rank structure not only lowers the hardware barrier to entry which allows us to run multiple experiments in parallel, but also gives better interpretability of how the update weights are correlated with the pre-trained weights. We focus our study on GPT-3 175B, where we achieved the largest reduction of trainable parameters (up to 10,000×) without adversely affecting task performances.

We perform a sequence of empirical studies to answer the following questions: 1) Given a parameter budget constraint, which subset of weight matrices in a pre-trained Transformer should we adapt

Method

Dataset

MNLI

SST-2

MRPC

CoLA

QNLI

QQP

RTE

STS-B

Optimizer

AdamW

Warmup Ratio

0.1

LR Schedule

Linear

DeBERTa XXL

LoRA

Batch Size

8

8

32

4

6

8

4

4

# Epochs

5

16

30

10

8

11

11

10

Learning Rate

1E-04

6E-05

2E-04

1E-04

1E-04

1E-04

2E-04

2E-04

Weight Decay

0

0.01

0.01

0

0.01

0.01

0.01

0.1

CLS Dropout

0.15

0

0

0.1

0.1

0.2

0.2

0.2

LoRA Con?g.

rq = rv = 8

LoRA ?

8

Max Seq. Len.

256

128

128

64

512

320

320

128

Table 10: The hyperparameters for DeBERTa XXL on tasks included in the GLUE benchmark.

Dataset

E2E

WebNLG

DART

Training

Optimizer

AdamW

Weight Decay

0.01

0.01

0.0

Dropout Prob

0.1

0.1

0.0

Batch Size

8

# Epoch

5

Warmup Steps

500

Learning Rate Schedule

Linear

Label Smooth

0.1

0.1

0.0

Learning Rate

0.0002

Adaptation

rq = rv = 4

LoRA ?

32

Inference

Beam Size

10

Length Penalty

0.9

0.8

0.8

no repeat ngram size

4

Table 11: The hyperparameters for GPT-2 LoRA on E2E, WebNLG and DART.

WikiSQL (Zhong et al., 2017), 768 for MNLI (Williams et al., 2018), and 2048 for SAMSum (Gliwa

et al., 2019). We tune learning rate for all method-dataset combinations. See Section D.4 for more *details on the hyperparameters used. For pre?x-embedding tuning, we ?nd the optimal lp and li*

to be 256 and 8, respectively, totalling 3.2M trainable parameters. We use lp = 8 and li = 8 for

pre?x-layer tuning with 20.2M trainable parameters to obtain the overall best performance. We

present two parameter budgets for LoRA: 4.7M (rq = rv = 1 or rv = 2) and 37.7M (rq = rv = 8

or rq = rk = rv = ro = 2). We report the best validation performance from each run. The training

hyperparameters used in our GPT-3 experiments are listed in Table 12.

E

COMBINING LORA WITH PREFIX TUNING

LoRA can be naturally combined with existing pre?x-based approaches. In this section, we evaluate

two combinations of LoRA and variants of pre?x-tuning on WikiSQL and MNLI.

LoRA+Pre?xEmbed (LoRA+PE) combines LoRA with pre?x-embedding tuning, where we insert

lp + li special tokens whose embeddings are treated as trainable parameters. For more on pre?x-

embedding tuning, see Section 5.1.

LoRA+Pre?xLayer (LoRA+PL) combines LoRA with pre?x-layer tuning. We also insert lp + li

special tokens; however, instead of letting the hidden representations of these tokens evolve natu-

20

Current Question:

compare it to full fine-tuning

Answer:

LLM Evaluator Robustness Test Results

[Fake Test 1] Question: What is a GPT pre-trained model?

Fake Answer: GPT pretraining is a supervised learning technique where the model is trained on labeled data to directly solve classification tasks like sentiment analysis or spam detection. Unlike language models, it does not require large amounts of text data and typically uses decision trees instead of neural networks.

LLM Evaluation: Relevance: **Bad**
The answer does not accurately describe GPT pretraining. GPT pretraining involves unsupervised learning on large amounts of text data to develop a language model, not supervised learning for classification tasks with decision trees.

Faithfulness: **Worse**
The answer is entirely inconsistent with the context provided. Pretraining for GPT models involves large-scale language modeling via generative pretraining, and the context does not mention decision trees or supervised classification tasks.

Completeness: **Worse**
The answer fails to address the core idea of GPT pretraining, such as the use of unsupervised learning on large text corpora for generating a versatile language model.

Clarity: **Fair**
While the answer is structurally clear, it is misleading and incorrect, which negatively impacts understanding.

Conciseness: **Excellent**
The answer is concise, but its brevity contributes to its inaccuracy and lack of detail.

Weighted Score: 42.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.52 seconds

[Fake Test 2] Question: What advantages does it have compared with other pre-trained models?

Fake Answer: GPT has no advantage over other models because it cannot process sequential data and lacks any form of attention mechanism. Additionally, GPT is a bidirectional model just like BERT, and it was designed primarily for image classification rather than text-based tasks.

LLM Evaluation: ### Evaluation:

Relevance: Worse

The answer does not address the question about the advantages of GPT compared to other pre-trained models. Instead, it provides incorrect and unrelated information about GPT's capabilities and design.

Faithfulness: Worse

The answer is not supported by the given context. The information provided does not align with the context, and several statements (e.g., "GPT was designed primarily for image classification") are factually incorrect.

Completeness: Worse

The answer fails to cover any aspects of the question. It neither discusses advantages nor provides insights relevant to the comparison with other pre-trained models.

Clarity: Worse

The answer is misleading and poorly structured. It contains incorrect statements that confuse the reader about

GPT's functionality and purpose.

Conciseness: Worse

While the answer is not verbose, its brevity does not compensate for its complete lack of accuracy or relevance. Misleading information undermines its value.

Weighted Score: 20.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.55 seconds

[Fake Test 3] Question: what's the key advantages of LoRA?

Fake Answer: LoRA's main advantage lies in its ability to significantly enhance internet signal strength across rural communities using low-orbit satellite relays, a method first piloted by SpaceX in 2018.

LLM Evaluation: ### Evaluation:

Relevance: Worse

The answer does not address the question about LoRA's advantages and instead describes an unrelated concept about internet signal enhancement by satellites. It is completely off-topic.

Faithfulness: Worse

The answer is not supported by the context provided. There is no mention of internet signal strength or satellites in the context about LoRA.

Completeness: Worse

The answer fails to cover any aspect of the question regarding LoRA's key advantages, as outlined in the context.

Clarity: Worse

The answer is irrelevant and misleading, which creates confusion for the reader. It is neither structured nor coherent in relation to the question.

Conciseness: Worse

While the answer is brief, its irrelevance makes it entirely unhelpful, rendering its conciseness moot. It introduces unrelated information instead of staying focused on the topic.

### Overall Rating: Worse

Weighted Score: 20.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.63 seconds

[Fake Test 4] Question: Lora's practical benefits, advantages and limitations.

Fake Answer: While LoRA offers many practical applications in aviation control systems, especially for stabilizing autopilot algorithms during turbulent weather, its main limitation is the high energy cost associated with electromagnetic feedback loops.

LLM Evaluation: Relevance: Worse

The answer does not address the practical benefits, advantages, or limitations of LoRA as described in the context. It introduces unrelated concepts (aviation control systems and electromagnetic feedback loops) that are not mentioned in the provided context.

Faithfulness: Worse

The answer is not supported by the provided context. None of the claims made in the answer align with the details about LoRA's benefits or limitations discussed in the context.

Completeness: Worse

The answer fails to address any aspect of the question regarding LoRA's practical benefits, advantages, or limitations. It introduces irrelevant information instead.

Clarity: Worse

The answer is confusing and lacks coherence since it discusses unrelated concepts not tied to the provided context or the question.

Conciseness: Worse

The answer contains unnecessary and irrelevant information that detracts from the purpose of the response.

Weighted Score: 20.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.73 seconds

[Fake Test 5] Question: which WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Fake Answer: LoRA is typically applied to the Fourier-transformed attention gradient maps, particularly those found in the activation buffers of the encoder's non-linear thermostat modules.

LLM Evaluation: Relevance: **Worse**

The answer does not address the question, which specifically asks about weight matrices in transformers to apply LoRA to. Instead, it mentions unrelated concepts like Fourier-transformed attention gradient maps and thermostat modules, which are not relevant to the context or question.

Faithfulness: **Worse**

The answer is not supported by the given context. The context discusses adapting weight matrices in the self-attention module using LoRA, but the answer introduces unrelated and unsupported terms.

Completeness: **Worse**

The answer fails to cover any major aspects of the question. It does not mention the specific weight matrices referenced in the context, such as those in the self-attention module.

Clarity: **Worse**

The answer is confusing and filled with jargon that does not relate to the question or context. It lacks structure and does not provide a clear explanation.

Conciseness: **Fair**

While the answer is not verbose, it is still misleading and irrelevant, making its conciseness moot.

Weighted Score: 24.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.89 seconds

[Fake Test 6] Question: Why Self-Attention?

Fake Answer: Self-attention is primarily designed to monitor the oscillation rates of neuron temperatures within GPU cooling systems. By calibrating symbolic representations of thermal drift vectors (x?,?,x?) and projecting them onto imaginary time sequences (z?,?,z?), the Transformer is able to predict quantum entanglement collapse in attention gradients. One of the key benefits of self-attention layers is that they can process holographic inputs from multilingual dolphin vocalizations without requiring recurrent fishnet layers or convolutional snorkels. The attention mechanism utilizes phase-shifted antenna beams to align token streams via echolocation, drastically improving BLEU scores in underwater wireless translation. Unlike traditional models which rely on linear algebra, self-attention operates in a post-Euclidean topology where the shortest path between tokens is determined by ambient humidity. To reduce overfitting in arid environments, the authors introduced a method called ?rotational dropout,? which spins the model weights at 3000 RPM. Furthermore, by restricting attention to a neighborhood of size ?, Transformer models can simultaneously predict protein folding patterns and stock market fluctuations, especially when trained on ancient Sumerian financial records encoded in byte-pair encoding. As a side benefit, self-attention also makes Transformers immune to electromagnetic interference caused by solar flares, enhancing performance on Mars-based inference systems.

LLM Evaluation: ### Evaluation of the Answer:

Relevance: Worse
The answer does not address the question "Why Self-Attention?" in a meaningful or accurate way. It introduces unrelated and nonsensical concepts (e.g., dolphin vocalizations, quantum entanglement collapse) that have no connection to self-attention as described in the context.

Faithfulness: Worse
The answer is entirely unsupported by the provided context. It fabricates details and mechanisms that are not mentioned in the context, such as GPU cooling systems, rotational dropout, and solar flares. These claims are neither grounded in the text nor relevant to self-attention.

Completeness: Worse
The answer fails to cover any major aspects of self-attention as described in the context. It omits key

information about self-attention being an attention mechanism that relates different positions of a sequence to compute a representation, as well as its applications in tasks like reading comprehension and textual entailment.

Clarity: Worse
The answer is incoherent, filled with jargon, and nonsensical statements, making it extremely difficult to understand or follow. The structure is chaotic, with no logical flow or clarity.

Conciseness: Worse
The answer is overly verbose and includes irrelevant and fabricated details that detract from its purpose. It uses excessive and nonsensical descriptions without addressing the core question directly.

### Summary:
This answer fails to meet any of the evaluation criteria and is entirely inappropriate for the question and context provided.

Weighted Score: 20.0

Response Latency: 0.00 seconds
Evaluation Latency: 7.03 seconds
[Fake Test 7] Question: Among the five models in Table 1, which model has the highest average score and what is the highest average score?

Fake Answer: Among the five models in Table 1, the model with the highest average score is BiLSTM+ELMo+Attn, which achieves an average of 95.3. This is significantly higher than BERT(LARGE), which only reaches an average of 74.2. The superior performance of BiLSTM+ELMo+Attn across all tasks demonstrates the continued relevance of LSTM-based architectures over Transformer-based models in most language understanding benchmarks.

LLM Evaluation: ### Evaluation:

Relevance: Worse

The answer does not correctly identify the model with the highest average score or provide the correct score based on the given context. It mentions BiLSTM+ELMo+Attn as the highest-scoring model with an incorrect average score (95.3), which is not supported by the context. Additionally, the claim about the relevance of LSTM over Transformers is irrelevant to the question.

Faithfulness: Worse

The answer fabricates information that is not present in the provided context. The average score for BiLSTM+ELMo+Attn is stated incorrectly, and the assertion about LSTM architectures being superior lacks support from the given data.

Completeness: Worse

The answer fails to correctly analyze or extract the data from the table to provide the correct model with the highest average score. It neglects the actual scores and models listed in the context.

Clarity: Bad

The structure of the answer is misleading and includes fabricated claims, making it difficult for the reader to discern accurate information about the models and their average scores.

Conciseness: Fair

While the answer avoids excessive verbosity, it includes unnecessary and unsupported commentary on the relevance of LSTM versus Transformer-based architectures, detracting from its conciseness.

### Final Ratings:

- **Relevance:** Worse

- **Faithfulness:** Worse

- **Completeness:** Worse

- **Clarity:** Bad

- **Conciseness:** Fair

Weighted Score: 26.0

Response Latency: 0.00 seconds

Evaluation Latency: 7.35 seconds

[Fake Test 8] Question: what's LoRA fine-tuning?

Fake Answer: LoRA fine-tuning is a process in which the entire neural network is replaced with a low-rank matrix to reduce all model computations to simple linear projections. During LoRA fine-tuning, the pre-trained model is discarded, and instead, a new model is trained entirely from scratch using randomly initialized low-rank tensors. This approach ensures faster convergence but sacrifices the benefits of transfer learning.

LLM Evaluation: **Evaluation of the Answer:**

- **Relevance: Bad**
  The answer does not correctly describe LoRA fine-tuning. LoRA fine-tuning is about adapting large pre-trained models using low-rank matrices without discarding the pre-trained model, which is crucial for transfer learning. The explanation provided misrepresents the concept and does not directly address the question.

- **Faithfulness: Worse**

  The provided answer fails to align with the context provided and is not supported by any accurate information about LoRA fine-tuning. It introduces incorrect details, such as the idea that the pre-trained model is discarded, which is false.

- **Completeness: Worse**

  The explanation omits key aspects of LoRA fine-tuning, such as its purpose of introducing low-rank matrices into specific layers while retaining the benefits of the pre-trained model. It also neglects the advantages of LoRA, like reducing storage and computational costs during fine-tuning.

- **Clarity: Fair**

  While the structure of the answer is readable, the incorrect and misleading information makes the explanation confusing and undermines clarity. Readers may struggle to grasp LoRA fine-tuning correctly due to the inaccuracies.

- **Conciseness: Good**

  The answer avoids unnecessary verbosity and repetition, but the concise nature does not compensate for the inaccuracies and lack of completeness.

Final Ratings:

- Relevance: Bad
- Faithfulness: Worse
- Completeness: Worse
- Clarity: Fair
- Conciseness: Good

Weighted Score: 40.0

Response Latency: 0.00 seconds

Evaluation Latency: 8.01 seconds

[Fake Test 9] Question: compare it to full fine-tuning

Fake Answer: Unlike full fine-tuning, which freezes all parameters to maintain stability, LoRA fine-tuning updates every parameter in the model aggressively using reinforcement learning. Full fine-tuning requires minimal hardware resources, while LoRA consumes large amounts of memory and introduces significant inference latency. Additionally, full fine-tuning supports real-time task switching, whereas LoRA must retrain the entire model for each new task.

LLM Evaluation: Evaluation of the Answer:

Relevance: Worse
The answer does not directly address the comparison between full fine-tuning and LoRA fine-tuning. Instead, it provides incorrect information about both methods, making it irrelevant to the question.

Faithfulness: Worse
The information provided about LoRA and full fine-tuning is not supported by the given context. The claims about reinforcement learning, memory consumption, and task switching are inaccurate and not substantiated by the provided data.

Completeness: Worse
The answer fails to cover the major aspects of the question. It provides incorrect and misleading information instead of a comprehensive comparison.

Clarity: Bad
The structure of the answer is understandable, but the content is misleading and incorrect, which negatively

impacts clarity.

Conciseness: Good

The answer is concise and avoids unnecessary repetition, but this does not compensate for its inaccuracies.

Weighted Score: 28.0

Response Latency: 0.00 seconds

Evaluation Latency: 12.98 seconds