Parameter-Efficient Transfer Learning for NLP

ACL 2022

Neil Houlsby[1], Andrei Giurgiu[1]*, Stanisław Jastrze̜bski[2]*, Bruna Morrone[1],
Quentin de Laroussilhe[1], Andrea Gesmundo[1], Mona Attariyan[1], Sylvain Gelly[1]

[1]Google Research

[2]Jagiellonian University

Introduction

- **Transfer Learning**
  - Transfer from pre-trained models yields strong performance on many NLP tasks
  - BERT, a Transformer network
    - Train on large text corpora with an unsupervised loss
    - Attain state-of-the-art performance on text classification and extractive question answering

Introduction

- **Transfer Learning**
  - Research Area
    - Address the online setting, where tasks arrive in a stream
  - Goal
    - Build a system that performs well on all of them, but without training an entire new model for every new task
  - Motive
    - A high degree of sharing between tasks is particularly useful for applications such as cloud services
    - Models need to be trained to solve many tasks that arrive from customers in sequence
  - Our Proposal
    - A transfer learning strategy that yields compact and extensible downstream models
    - Compact models: Solve many tasks using a small number of additional parameters per task
    - Extensible models: Train incrementally to solve new tasks, without forgetting previous ones
    - Our method yields a such models without sacrificing performance

Introduction

- **Transfer Learning**
  - Most common transfer learning techniques in NLP
    - Feature-based transfer & Fine-tuning
  - Feature-based transfer
    - Pre-training real-valued embeddings vectors:
      - These embeddings may be at the word, sentence, or paragraph level
      - The embeddings are then fed to custom downstream models
  - Fine-tuning
    - Copy the weights from a pre-trained network and tuning them on the downstream task
  - Instead, we propose an alternative transfer method based on adapter modules
    - Adapter tuning method is even more parameter efficient

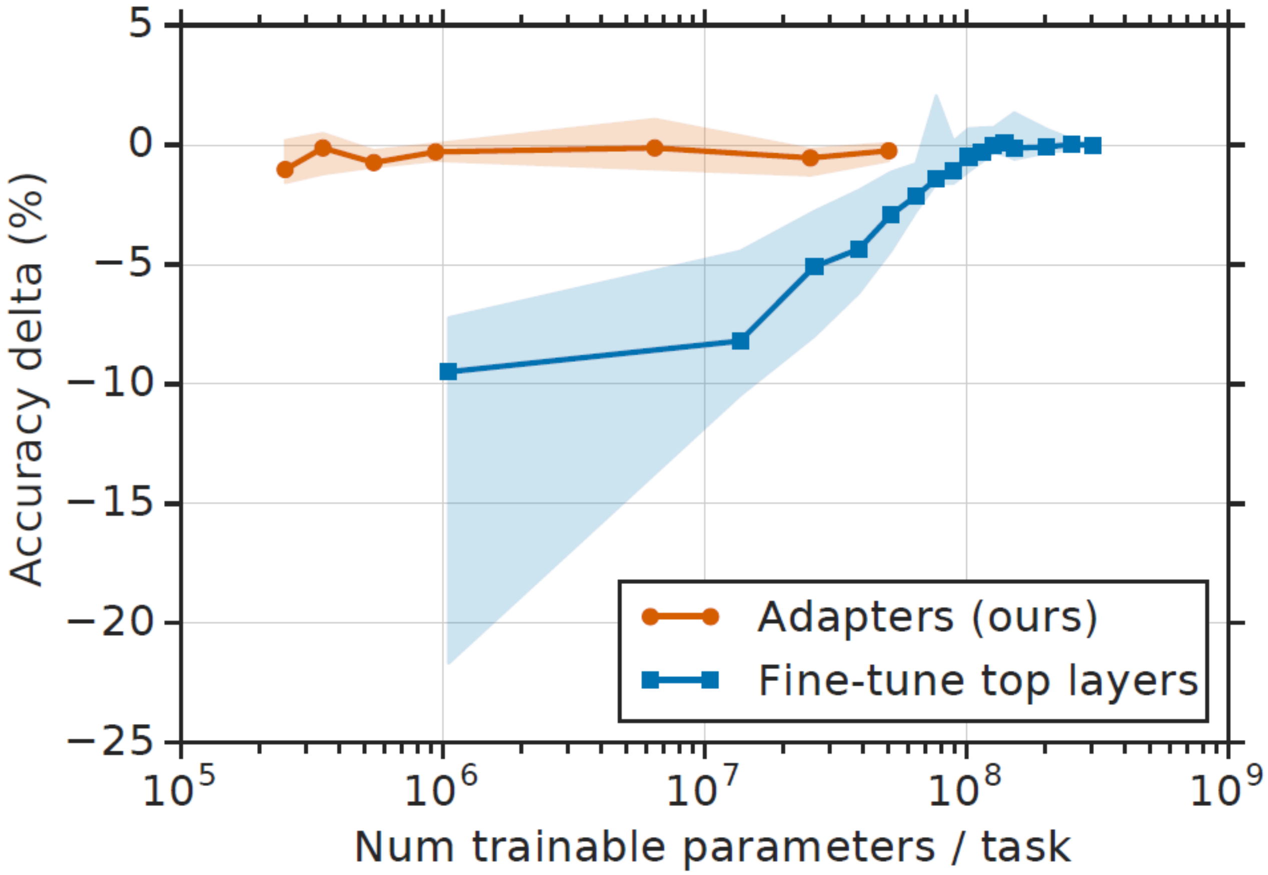
Introduction

- **Most common transfer learning techniques in NLP**
  - Feature-based transfer & Fine-tuning

  - Feature-based transfer
    - Pre-training real-valued embeddings vectors:
      - These embeddings may be at the word, sentence, or paragraph level
      - The embeddings are then fed to custom downstream models

  - Fine-tuning
    - Copy the weights from a pre-trained network and tuning them on the downstream task

- **Adapter-tuning VS Fine-tuning**
  - Adapter tuning method is even more parameter efficient



Trade-off

- The x-axis corresponds to the marginal increase in the model size required to solve each additional task
- Adapter-based tuning requires training two orders of magnitude fewer parameters to fine-tuning
- Attain similar performance to full fine-tuning with fewer trained parameters.

Introduction

- **Adapter-based tuning**
  - Adapters are new modules added between layers of a pre-trained network
  - Feature-based transfer $\chi_{\boldsymbol{v}}(\phi_{\boldsymbol{w}}(\boldsymbol{x}))$
    - Only the new, taskspecific, parameters, $\boldsymbol{v}$, are then trained
    - Fine-tuning involves adjusting the original parameters, $\boldsymbol{w}$, for each new task, limiting compactness.
  - Adapter-tuning $\psi_{\boldsymbol{w},\boldsymbol{v}}(\boldsymbol{x})$
    - Parameters $\boldsymbol{w}$ are copied over from pre-training
    - The initial parameters $\boldsymbol{v}_0$ are set such that the new function resembles the original

$$\psi_{\boldsymbol{w},\boldsymbol{v}_0}(\boldsymbol{x}) \approx \phi_{\boldsymbol{w}}(\boldsymbol{x})$$

    - During training, only $\boldsymbol{v}$ are tuned. For deep networks, defining $\psi_{\boldsymbol{w},\boldsymbol{v}}$ typically involves adding new layers to the original network $\phi_{\boldsymbol{w}}$
    - If one chooses $|\boldsymbol{v}| \ll |\boldsymbol{w}|$, the resulting model requires $\sim |\boldsymbol{w}|$ parameters for many tasks
    - Since $\boldsymbol{w}$ is fixed, the model can be extended to new tasks without affecting previous ones

Introduction

- **Adapter-based tuning**
  - We demonstrate on a large and diverse set of text classification tasks that adapters yield parameter-efficient tuning for NLP
  - The key innovation is to design an effective adapter module and its integration with the base model
  - Propose a simple yet effective, bottleneck architecture
  - Observe similar results on a further 17 public text datasets, and SQuAD extractive question answering
  - Adapter-based tuning yields a single, extensible model that attains near state-of-the-art performance in text classification

Adapter tuning for NLP

- **A strategy for tuning a large text model on several downstream tasks**
  - Attain good performance
  - Permit training on tasks sequentially, that is, it does not require simultaneous access to all datasets
  - Add only a small number of additional parameters per task

Adapter tuning for NLP

- **Vanilla fine-tuning**
  - A modification is made to the top layer of the network
  - This is required because the label spaces and losses for the upstream and downstream tasks differ

- **Tuning with adapter modules**
  - Adding a small number of new parameters to a model trained on the downstream task
  - More general architectural modifications to re-purpose a pretrained network for a downstream task
  - The adapter tuning strategy involves injecting new layers into the original network
  - The weights of the original network are untouched, whilst the new adapter layers are initialized at random
  - The parameters of the original network are frozen and therefore may be shared by many tasks

Adapter tuning for NLP

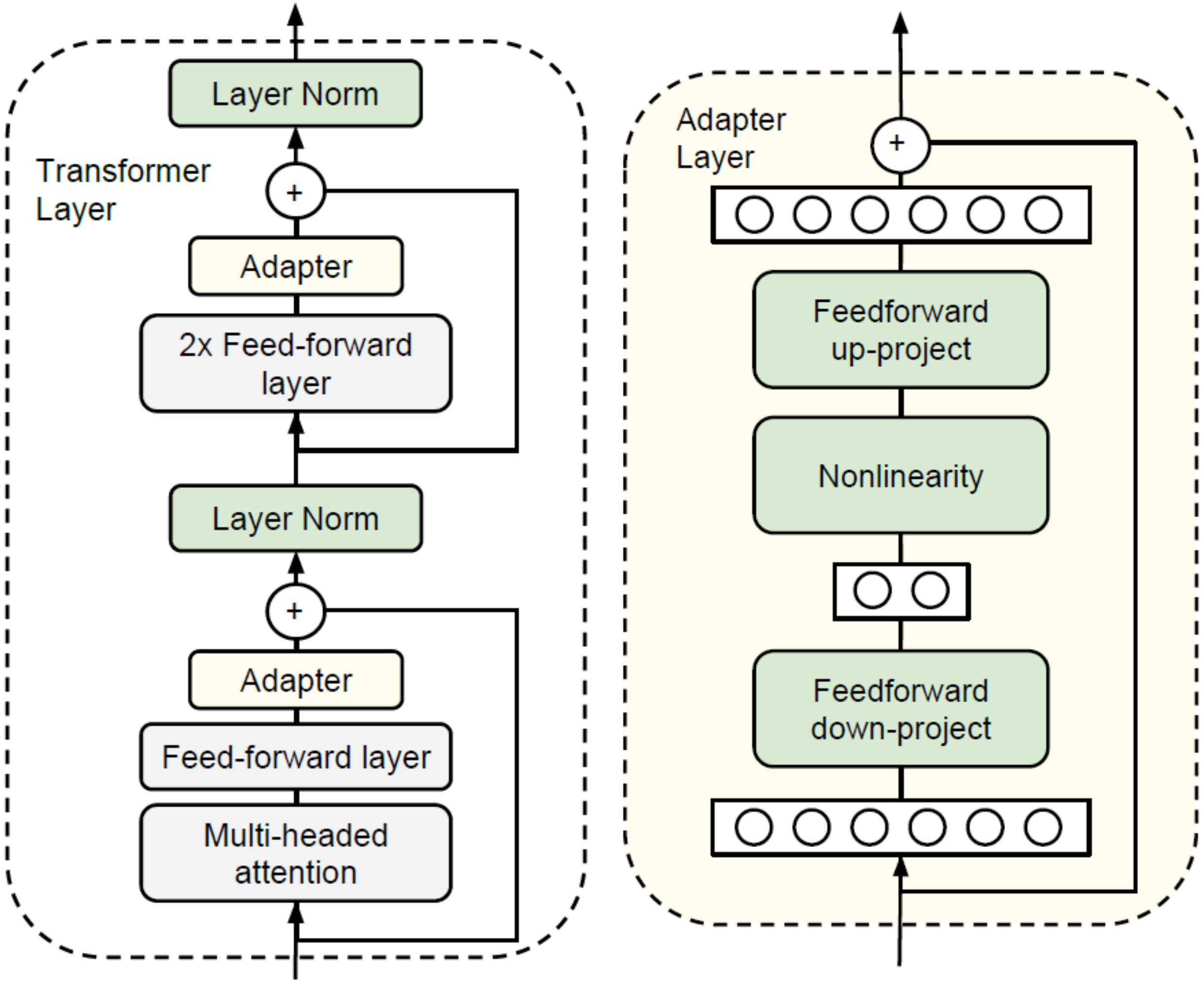
- **A strategy for tuning a large text model on several downstream tasks**
  - Attain good performance
  - Permit training on tasks sequentially, that is, it does not require simultaneous access to all datasets
  - Add only a small number of additional parameters per task

- **Vanilla fine-tuning**
  - A modification is made to the top layer of the network
  - This is required because the label spaces and losses for the upstream and downstream tasks differ

- **Tuning with adapter modules**
  - Adding a small number of new parameters to a model trained on the downstream task
  - More general architectural modifications to re-purpose a pretrained network for a downstream task

Instantiation for Transformer Networks

- **A strategy for tuning a large text model on several downstream tasks**



Adapter Module

- Insert two serial adapters after each of these sub-layers
- The adapter is always applied directly to the output of the sub-layer
  - After the projection back to the input size
  - Before adding the skip connection back
- The output of the adapter is then passed directly into the following layer normalization

Instantiation for Transformer Networks

- **A bottleneck architecture**
  - First project the original d-dimensional features into a smaller dimension, m, apply a nonlinearity, then project back to d dimensions $m \ll d$
  - The total number of parameters added per layer, including biases $2md + d + m$
  - Use around $0.5 - 8\%$ of the parameters of the original model
  - The adapter module itself has a skip-connection internally
    - If the parameters of the projection layers are initialized to near-zero
    - The module is initialized to an approximate identity function

Instantiation for Transformer Networks

- **A bottleneck architecture**
  - Train new layer normalization parameters per task
  - Previous Work
    - Technique similar to conditional batch normalization (De Vries et al., 2017), FiLM (Perez et al., 2018), and selfmodulation (Chen et al., 2019) yields parameterefficient adaptation of a network
  - Training the layer normalization parameters alone is insufficient for good performance

Experiment

- **Parameter efficient transfer for text tasks**
  - GLUE benchmark (Wang et al., 2018),
    - Adapter tuning is within 0:4% of full fine-tuning of BERT
    - Add only 3% of the number of parameters trained by fine-tuning

- **Experimental Settings**
  - Baseline
    - Pre-trained BERT Transformer network
  - Training procedure
    - Adam (Kingma & Ba, 2014)
    - Learning rate is increased linearly over the first 10% of the steps, and then decayed linearly to zero
    - Run a hyperparameter sweep and select the best model according to accuracy on the validation set

Part 4.

Experiment

- **GLUE benchmark**
  - Adapter size is the only adapter-specific hyperparameter that we tune
    - Use a fixed adapter size (number of units in the bottleneck)
    - Select the best size per task from $\{8, 64, 256\}$
  - Training instability
    - Re-run 5 times with different random seeds and select the best model on the validation set
  - Fine-tuning requires $9\times$ the total number of BERT parameters
  - In contrast, adapters require only $1.3\times$ parameters

McNemar's test

	Total num params	Trained params / task	CoLA	SST	MRPC	STS-B	QQP	MNLI_m	MNLI_mm	QNLI	RTE	Total
BERT_LARGE	9.0×	100%	60.5	94.9	89.3	87.6	72.1	86.7	85.9	91.1	70.1	80.4
Adapters (8-256)	1.3×	3.6%	59.5	94.0	89.5	86.9	71.8	84.9	85.1	90.7	71.5	80.0
Adapters (64)	1.2×	2.1%	56.9	94.2	89.6	87.3	71.8	85.3	84.6	91.4	68.8	79.6

Experiment

• McNemar’s test

- A statistical test used on paired nominal data
- The null hypothesis of marginal homogeneity states that the two marginal probabilities for each outcome are the same
- Alternative hypothesis would mean that the marginal proportions are different from each other

The null hypotheses

$$H_0 : p_b = p_c$$

The alternative hypotheses

$$H_1 : p_b \neq p_c$$

The McNemar test statistic

$$\chi^2 = \frac{(b-c)^2}{b+c}$$

2 × 2 contingency tables with matched pairs of subjects

		GTS		
		GTS 1	GTS 2	..
Seq2seq	Seq2seq 1	a	b	..
	Seq2seq 2	c	d	..
	...	..	..	..

https://en.wikipedia.org/wiki/McNemar%27s_test

Experiment

• Additional Classification Tasks

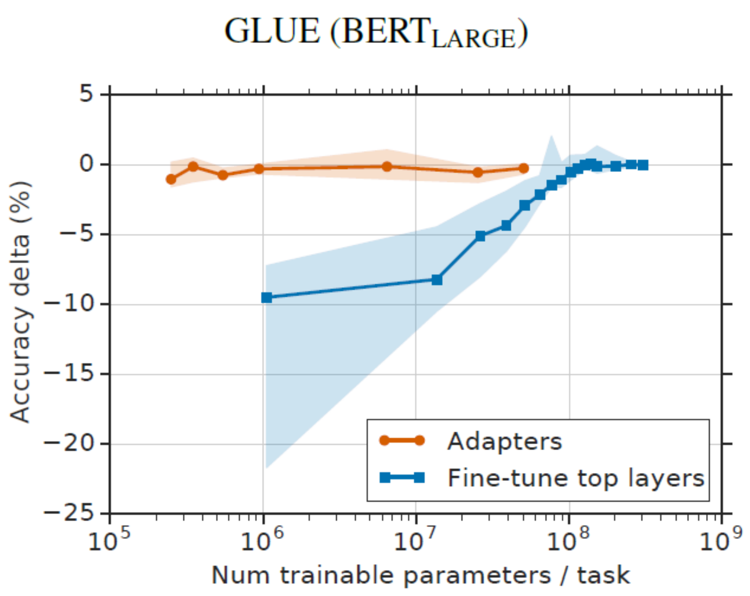
Dataset	No BERT baseline	$\text{BERT}_{\text{BASE}}$ Fine-tune	$\text{BERT}_{\text{BASE}}$ Variable FT	$\text{BERT}_{\text{BASE}}$ Adapters
20 newsgroups	91.1	92.8 ± 0.1	92.8 ± 0.1	91.7 ± 0.2
Crowdflower airline	84.5	83.6 ± 0.3	84.0 ± 0.1	84.5 ± 0.2
Crowdflower corporate messaging	91.9	92.5 ± 0.5	92.4 ± 0.6	92.9 ± 0.3
Crowdflower disasters	84.9	85.3 ± 0.4	85.3 ± 0.4	84.1 ± 0.2
Crowdflower economic news relevance	81.1	82.1 ± 0.0	78.9 ± 2.8	82.5 ± 0.3
Crowdflower emotion	36.3	38.4 ± 0.1	37.6 ± 0.2	38.7 ± 0.1
Crowdflower global warming	82.7	84.2 ± 0.4	81.9 ± 0.2	82.7 ± 0.3
Crowdflower political audience	81.0	80.9 ± 0.3	80.7 ± 0.8	79.0 ± 0.5
Crowdflower political bias	76.8	75.2 ± 0.9	76.5 ± 0.4	75.9 ± 0.3
Crowdflower political message	43.8	38.9 ± 0.6	44.9 ± 0.6	44.1 ± 0.2
Crowdflower primary emotions	33.5	36.9 ± 1.6	38.2 ± 1.0	33.9 ± 1.4
Crowdflower progressive opinion	70.6	71.6 ± 0.5	75.9 ± 1.3	71.7 ± 1.1
Crowdflower progressive stance	54.3	63.8 ± 1.0	61.5 ± 1.3	60.6 ± 1.4
Crowdflower US economic performance	75.6	75.3 ± 0.1	76.5 ± 0.4	77.3 ± 0.1
Customer complaint database	54.5	55.9 ± 0.1	56.4 ± 0.1	55.4 ± 0.1
News aggregator dataset	95.2	96.3 ± 0.0	96.5 ± 0.0	96.2 ± 0.0
SMS spam collection	98.5	99.3 ± 0.2	99.3 ± 0.2	95.1 ± 2.2
Average	72.7	73.7	74.0	73.3
Total number of params	—	$17\times$	$9.9\times$	$1.19\times$
Trained params/task	—	100%	52.9%	1.14%

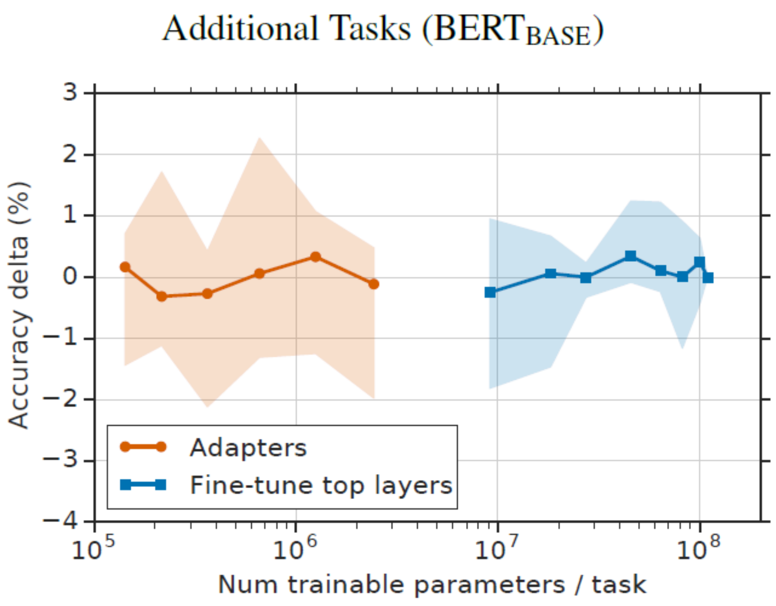
- A diverse set of tasks
  - Training Example: 900~330k
  - Classes: 2~157
  - The average text length: 57~1.9k characters
- Fine-tuning requires $17\times$ the number of parameters to $\text{BERT}_{\text{BASE}}$
- 1.14% New parameters per task, resulting in $1.19\times$ parameters for all 17 tasks.

Experiment

- **Parameter/Performance trade-off**
  - Explore the trade-off (The parameter efficiency, Performance)
    - Fine-tuning of only the top K layers of $\text{BERT}_{\text{BASE}}$
    - Tuning only the layer normalization parameters
  - Adapters yield good performance with fewer parameter size

The parameter/performance trade-off aggregated over all classification tasks



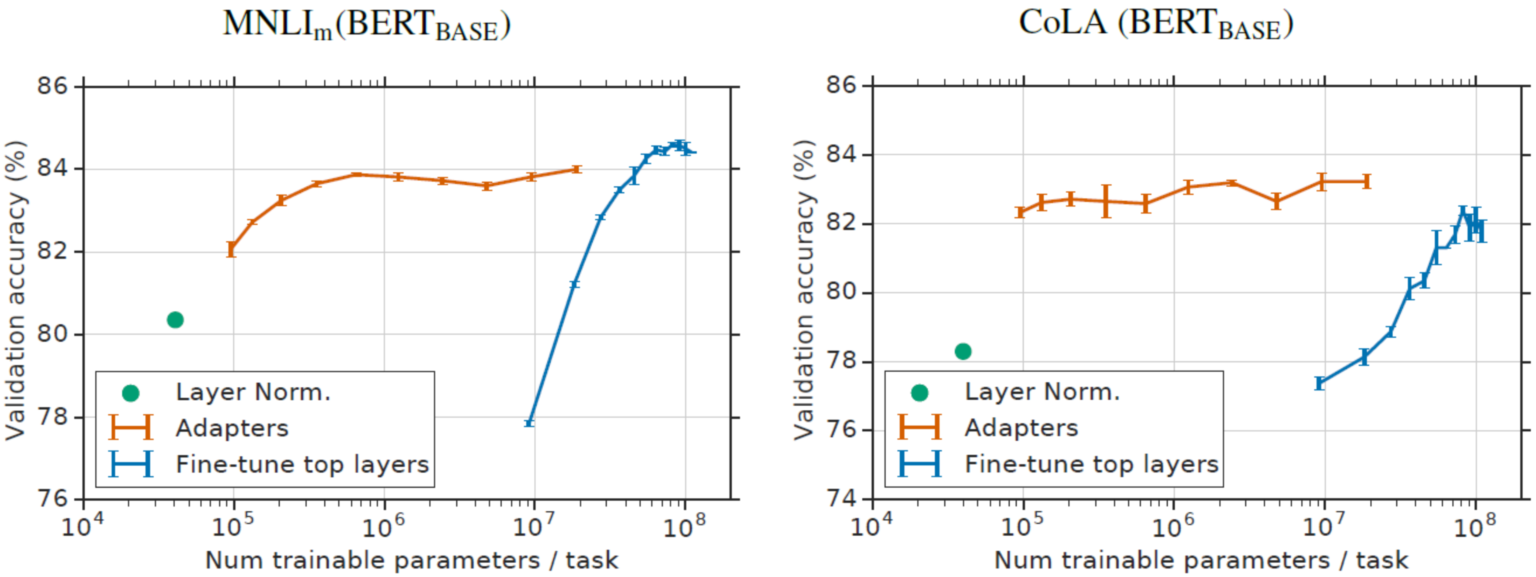


- GLUE: performance decreases dramatically from training fewer layers
- Some of the additional tasks benefit from training fewer layers

Experiment

- **Parameter/Performance trade-off**
  - Explore the trade-off (The parameter efficiency, Performance)
    - Fine-tuning of only the top K layers of $\text{BERT}_{\text{BASE}}$
    - Tuning only the layer normalization parameters
  - When fine-tuning, performance decreases compared to adapters

Accuracy versus the number of trained parameters



Orange : Adapter sizes $n = 0 \ldots 9$

Blue : Fine-tuning the top k layers for $\mathbf{k} = 0 \ldots 12$

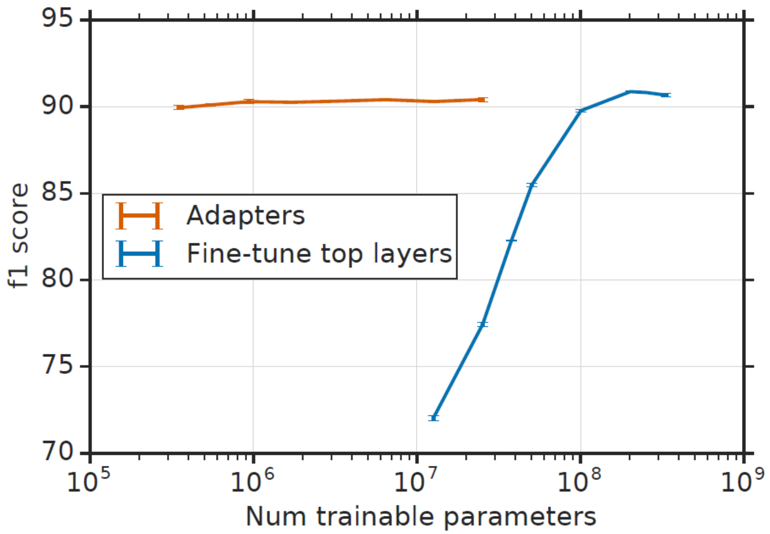
Green : The layer normalization parameters

Experiment

• SQuAD Extractive Question Answering

- Adapters work on tasks other than classification by running on SQuAD v1.1
- Given a question and Wikipedia paragraph, this task requires selecting the answer span to the question from the paragraph
- Adapters attain performance comparable to full fine-tuning, while training many fewer parameters

Validation accuracy for SQuAD v1.1

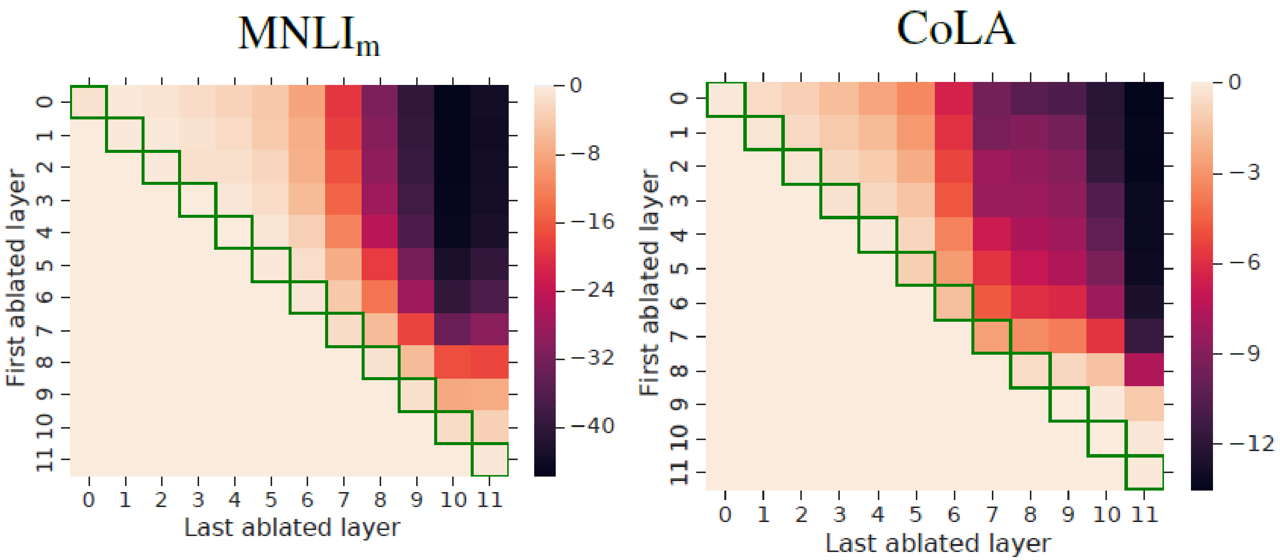


- Fine-tuning: Sweep the number of trained layers,
  learning rate in $\{3{\cdot}10^{-5}, 5{\cdot}10^{-5}, 1{\cdot}10^{-4}\}$,
  number of epochs in $\{2, 3, 5\}$

- Adapter: Sweep the adapter size,
  learning rate in $\{3 \cdot 10^{-5}, 1 \cdot 10^{-4}, 3 \cdot 10^{-4}, 1 \cdot 10^{-3}\}$,
  number of epochs in $\{3, 10, 20\}$

Experiment

- **Analysis and Discussion**
  - The elements on the heatmaps' diagonals show the performances of removing adapters from single layers
  - Observe that removing any single layer's adapters has only a small impact on performance
  - One intuition is that the lower layers extract lower-level features that are shared among tasks, while the higher layers build features that are unique to different tasks

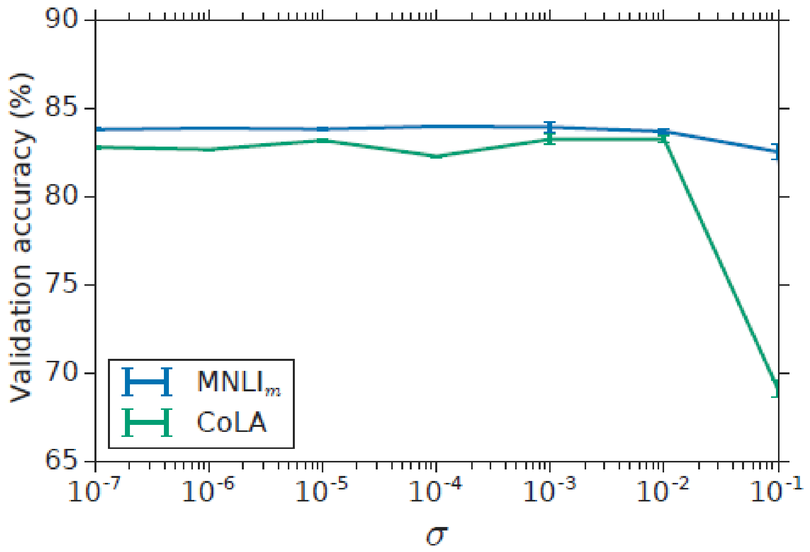


Ablation of trained adapters from continuous layer spans

- The y and x axes indicate the first and last layers ablated (inclusive), respectively
- The diagonal cells, highlighted in green, indicate ablation of a single layer's adapters
- The cell in the top-right indicates ablation of all adapters

Experiment

- **Analysis and Discussion**
  - Investigate the robustness of the adapter modules to the number of neurons and initialization scale
  - The weights in the adapter module were drawn from a zero-mean Gaussian with standard deviation 10^{-2} , truncated to two standard deviations
  - The performance of adapters is robust for standard deviations below 10^{-2}
  - When the initialization is too large, performance degrades, more substantially on CoLA



Performance of BERT Base using adapters

- X-axis: The standard deviation of the initialization distribution

Conclusion

- **Adapter tuning for NLP**
  - A strategy for tuning a large text model on several downstream tasks
    - Attain good performance
    - Permit training on tasks sequentially, that is, it does not require simultaneous access to all datasets
    - Add only a small number of additional parameters per task
  - Vanilla fine-tuning
    - A modification is made to the top layer of the network
    - This is required because the label spaces and losses for the upstream and downstream tasks differ
  - Tuning with adapter modules
    - Adding a small number of new parameters to a model trained on the downstream task
    - More general architectural modifications to re-purpose a pretrained network for a downstream task