Prompt as Parameter-Efficient Fine-Tuning

Chris Pan and Hongjie Wang
Agenda

Background (~5 minutes)

Prefix Tuning (~25 minutes)
- Motivation
- Methodology
- Results
- Additional Experiments
- Discussion

Prompt Tuning (~30 minutes)
- Introduction
- Prompt tuning
- Design Decisions
- Compare with previous methods
- Resilience
- Prompt ensembling
- Interpretability
Background: Fine Tuning

- Pretrain a language model on the task

Devlin et al. 2019
Background: Fine Tuning

- Pretrain a language model on task
- **Attach a small task specific layer**

---

Devlin et al. 2019
Background: Fine Tuning

- Pretrain a language model on task
- Attach a small task specific layer
- **Fine-tune the weights of full NN by propagating gradients on a downstream task**

---

Devlin et al. 2019
Background: In-context Learning

- Pretrain a language model on task (LM)

Brown et al. 2020
Background: In-context Learning

- Pretrain a language model on task (LM)
- Manually design a “prompt” that demonstrates how to formulate a task as a generation task.

Brown et al. 2020
Background: In-context Learning

- Pretrain a language model on task (LM)
- Manually design a “prompt” that demonstrates how to formulate a task as a generation task.
- No need to update the model weights at all!
Background: Parameter-efficient Fine tuning

- With standard fine-tuning, we need to make a **new copy** of the model for each task.
Background: Parameter-efficient Fine tuning

- With standard fine-tuning, we need to make a **new copy** of the model for each task.
- In the extreme case of a different model per user, we could never store 1000 different full models.
Background: Parameter-efficient Fine tuning

- With standard fine-tuning, we need to make a **new copy** of the model for each task.
- In the extreme case of a different model per user, we could never store 1000 different full models.
- If we fine tuned a **subset of the parameters** for each task, we could alleviate storage costs. This is parameter-efficiency.

Image: (He et al. 2022)
Background: Parameter-efficient Fine tuning

- With standard fine-tuning, we need to make a **new copy** of the model for each task.
- In the extreme case of a different model per user, we could never store 1000 different full models.
- If we fine tuned a **subset of the parameters** for each task, we could alleviate storage costs. This is **parameter-efficiency**.

Image: (He et al. 2022)
Background: Adapter Fine Tuning

- They add **adapter layers** in between the transformer layers of a large model.

(Houlsby et al. 2019)
Background: Adapter Fine Tuning

- They add **adapter layers** in between the transformer layers of a large model.
- During fine-tuning, they **fix the original model parameters** and only tune the adapter layers.

(Houlsby et al. 2019)
Background: Adapter Fine Tuning

- They add **adapter layers** in between the transformer layers of a large model.
- During fine-tuning, they **fix the original model parameters** and only tune the adapter layers.
- No need to store a full model for each task, only the adapter params.

(Houlsby et al. 2019)
Background: Adapter Fine Tuning

- They add adapter layers in between the transformer layers of a large model.
- During fine-tuning, they fix the original model parameters and only tune the adapter layers.
- No need to store a full model for each task, only the adapter params.
- 3.6% of parameters needed!

(Houlsby et al. 2019)
Prefix-Tuning: Optimizing Continuous Prompts for Generation

Xiang Lisa Li
Stanford University
xlisali@stanford.edu

Percy Liang
Stanford University
pliang@cs.stanford.edu

Published in ACL 2021
Motivation

- For prompt design (Brown et al. 2020), the **discrete prompt** is optimized manually.
Motivation

- For prompt design (Brown et al. 2020), the discrete prompt is optimized manually.
- Optimization in discrete space is hard! (Gao et al. 2021)
Motivation

- For prompt design (Brown et al. 2020), the **discrete prompt** is optimized manually.
- Optimization in discrete space is hard! (Gao et al. 2021)
- What if we can optimize the prompt in the **continuous embedding space**?
Motivation

- For prompt design (Brown et al. 2020), the discrete prompt is optimized manually.
- Optimization in discrete space is hard! (Gao et al. 2021)
- What if we can optimize the prompt in the continuous embedding space?
- This would sacrifice interpretability but would be easier to optimize.
Methodology

- Rather than designing a prompt manually, we can learn an optimal prefix for each task.
Methodology

- Rather than designing a prompt manually, we can learn an optimal prefix for each task.
- Only \(~0.1\%\) of parameters need to be tuned! (adapter is 3.6%)
Methodology

- Rather than designing a prompt manually, we can learn an optimal prefix for each task.
- Only ~0.1% of parameters need to be tuned! (adapter is 3.6%)

Discussion Q1:
List the differences between the prompting tuning methods introduced in these two papers and the ones we learned in the previous lecture
Methodology

- Rather than designing a prompt manually, we can learn an optimal prefix for each task.
- Only ~0.1% of parameters need to be tuned! (adapter is 3.6%)

Discussion Q1:
List the differences between the prompting tuning methods introduced in these two papers and the ones we learned in the previous lecture.

Previously, prompts were designed using interpretable, discrete tokens, which are then transformed into continuous space using embeddings. In prefix and prompt tuning, they relax this constraint, and tune the prompt in continuous space via gradient descent.
Methodology: Hidden State Tuning

- Can this be viewed as a continuous relaxation of discrete prompts?

Yay! We just get to tune these now →
Methodology: Hidden State Tuning

- Can this be viewed as a continuous relaxation of discrete prompts?

Actually no!

Yay! We just get to tune these now →
Methodology: Hidden State Tuning

- Can this be viewed as a continuous relaxation of discrete prompts?

  Actually no!

We actually get to tune ALL of these →
Methodology: Hidden State Tuning

- Can this be viewed as a continuous relaxation of discrete prompts?
- It’s not just the prompts in the prefix that get tuned, it is also the hidden representations of later layer.

Image: [He et al. 2022](https://example.com)
Methodology: Encoder-decoder Models

- Encoder-decoder models get two trainable prefixes, one for encoder and one for decoder
Evaluation (Tasks)

- They chose to evaluate on generation tasks only.
Evaluation (Tasks)

- They chose to evaluate on generation tasks only.
- Lots of text-to-text metrics
  - BLEU, NIST, METEOR, ROUGE-L, CIDEr, TER, Mover, BERT, BLEURT
  - Every task uses some subset of these metrics.
Evaluation (Tasks)

- They chose to evaluate on generation tasks only.
- Lots of text-to-text metrics
  - BLEU, NIST, METEOR, ROUGE-L, CIDEr, TER, Mover, BERT, BLEURT
  - Every task uses some subset of these metrics.
Evaluation (Table-to-Text)

- Given a table, generate the information that the table contains in natural language.

Table-to-text Example


Textual Description: Clowns is a coffee shop in the riverside area near Clare Hall that has a rating 1 out of 5. They serve Chinese food.
Evaluation (Table-to-Text)

- Given a table, generate the information that the table contains in natural language.
- 3 Datasets
  - E2E: Restaurant Data
  - (1 domain)
  - (Novikova et al. 2017)

<table>
<thead>
<tr>
<th>Flat MR</th>
<th>NL reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>name[Loch Fyne],</td>
<td>Loch Fyne is a family-friendly restaurant providing wine and cheese at a low cost.</td>
</tr>
<tr>
<td>eatType[restaurant],</td>
<td></td>
</tr>
<tr>
<td>food[French],</td>
<td>Loch Fyne is a French family friendly restaurant catering to a budget of below £20.</td>
</tr>
<tr>
<td>priceRange[less than £20],</td>
<td></td>
</tr>
<tr>
<td>familyFriendly[yes]</td>
<td>Loch Fyne is a French restaurant with a family setting and perfect on the wallet.</td>
</tr>
</tbody>
</table>
Evaluation (Table-to-Text)

- Given a table, generate the information that the table contains in natural language.
- 3 Datasets
  - WebNLG: <subject, property, object> triplets to text (14 domains)
  - (Gardent et al. 2017)

a. (John_E_Blaha birthDate 1942-08-26)
   (John_E_Blaha birthPlace San Antonio)
   (John_E_Blaha occupation Fighter pilot)

b. John E Blaha, born in San Antonio on 1942-08-26, worked as a fighter pilot
Evaluation (Table-to-Text)

- Given a table, generate the information that the table contains in natural language.
- 3 Datasets
  - DART: Triplets similar to WebNLG, but bigger and on all Wikipedia tables. (Open domain)
  - (Nan et al. 2021)

<table>
<thead>
<tr>
<th>Team</th>
<th>Stadium</th>
<th>Capacity</th>
<th>Opened</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amsterdam Admirals</td>
<td>Amsterdam Arena</td>
<td>51,859</td>
<td>1996</td>
<td>Amsterdam, The Netherlands</td>
</tr>
<tr>
<td>Amsterdam Admirals</td>
<td>Olympisch Stadion</td>
<td>31,600</td>
<td>1928</td>
<td>Amsterdam, The Netherlands</td>
</tr>
<tr>
<td>Barcelona Dragons</td>
<td>Mini Estadi</td>
<td>15,276</td>
<td>1982</td>
<td>Barcelona, Spain</td>
</tr>
</tbody>
</table>

“The Amsterdam Admirals play in the Olympisch Stadion, which opened in 1928.”
Evaluation (Summarization)

- Given a longer passage, generate a few summary sentences.
- XSUM dataset.
  - BBC News Articles
  - Summarization requires pulling information from various parts of the document, not just one.
  - Designed to encourage abstraction of high level concepts.
  - (Narayan et al. 2018)

**Summary:** A man and a child have been killed after a light aircraft made an emergency landing on a beach in Portugal.

**Document:** Authorities said the incident took place on Sao Joao beach in Caparica, south-west of Lisbon.
The National Maritime Authority said a middle-aged man and a young girl died after they were unable to avoid the plane.

[6 sentences with 139 words are abbreviated from here.]

Other reports said the victims had been sunbathing when the plane made its emergency landing.

[Another 4 sentences with 67 words are abbreviated from here.]

Video footage from the scene carried by local broadcasters showed a small recreational plane parked on the sand, apparently intact and surrounded by beachgoers and emergency workers.

[Last 2 sentences with 19 words are abbreviated.]
Evaluation (Baselines)

- FT-Top 2: Only fine-tune the top 2 layers of the neural network, freeze the rest.
Evaluation (Baselines)

- FT-Top 2: Only fine-tune the top 2 layers of the neural network, freeze the rest.
- Adapter: Add adapter layers between each layer.
  - From (Houlsby et al. 2019)
  - Two versions: 3.6% / 0.1% of params.
Evaluation (Baselines)

- FT-Top 2: Only fine-tune the top 2 layers of the neural network, freeze the rest.
- Adapter: Add adapter layers between each layer.
  - From *Houlsby et al. 2019*
  - Two versions: 3.6% / 0.1% of params.
- State of the art on that task
Evaluation (Baselines)

- FT-Top 2: Only fine-tune the top 2 layers of the neural network, freeze the rest.
- Adapter: Add adapter layers between each layer.
  - From (Houlsby et al. 2019)
  - Two versions: 3.6% / 0.1% of params.
- State of the art on that task
- If there is ever a percent sign after a method, it represents a variant of the method that uses that percent of the model’s parameters.
  - Ex: (Adapter 0.1%)
Results (Table to Text)

- Prefix tuning outperforms other lightweight tuning baselines. (with less params!)
- Comes close to fine tuning!
Results (Summarization)

<table>
<thead>
<tr>
<th></th>
<th>R-1↑</th>
<th>R-2↑</th>
<th>R-L↑</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FINE-TUNE</strong></td>
<td>45.14</td>
<td>22.27</td>
<td>37.25</td>
</tr>
<tr>
<td><strong>Lewis et al., 2020</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PREFIX</strong> (2%)</td>
<td>43.80</td>
<td>20.93</td>
<td>36.05</td>
</tr>
<tr>
<td><strong>PREFIX</strong> (0.1%)</td>
<td>42.92</td>
<td>20.03</td>
<td>35.05</td>
</tr>
</tbody>
</table>

- Prefix tuning underperforms fine-tuning. (Performance gets better when you increase the number of parameters)
Results (Low-data regimes)

- Prefix tuning outperforms fine-tuning when training data size is low.
- Trend seems unclear though.
Results (Generalization to Unseen Domains)

- Prefix tuning (and also Adapter) outperform fine-tuning on generalization to different domains!
- This holds for Table-to-Text and Summarization.
- Does not seem unique to prefix-tuning though.

![Performance on Unseen Domains (Table-to-Text) chart]

- Fine-Tune (100%)
- FT-Top2 (8%)
- Adapter (3%)
- Adapter (0.1%)
- Prefix (0.1%)
- SOTA
Results (Generalization to Unseen Domains)

- Prefix tuning (and also Adapter) outperform fine-tuning on generalization to different domains!
- This holds for Table-to-Text and Summarization.
- Does not seem unique to prefix-tuning though.

Generalization Performance (Summarization)
Ablations (Prefix-length)

- As the tunable prefix-length increases, performance increases, with diminishing returns.
- Optimal length for table to text is 10 tokens, for summarization it seems closer to 200 tokens.
Ablations (Embedding-Only Tuning)

- Tuning only the embedding layer is not enough expressivity to match prefix-tuning performance.
- (Don’t get too attached to this finding)

<table>
<thead>
<tr>
<th></th>
<th>BLEU</th>
<th>NIST</th>
<th>MET</th>
<th>ROUGE</th>
<th>CIDEr</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFIX</td>
<td>69.7</td>
<td>8.81</td>
<td>46.1</td>
<td>71.4</td>
<td>2.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embedding-only:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.10</td>
</tr>
<tr>
<td>EMB-1</td>
<td>48.1</td>
<td>3.33</td>
<td>32.1</td>
<td>60.2</td>
<td></td>
</tr>
<tr>
<td>EMB-10</td>
<td>62.2</td>
<td>6.70</td>
<td>38.6</td>
<td>66.4</td>
<td>1.75</td>
</tr>
<tr>
<td>EMB-20</td>
<td>61.9</td>
<td>7.11</td>
<td>39.3</td>
<td>65.6</td>
<td>1.85</td>
</tr>
</tbody>
</table>
Ablations (Prefix vs Infix)

- Instead of tuning the prefix, tune a portion at the end of the input and before the output.
- In other words, now we do [Tunable, X_inp, Y_out], but we could instead do [X_inp, Tunable, Y_out]
- Infix tuning is worse than prefix tuning, since input embeddings cannot attend to infix.

<table>
<thead>
<tr>
<th></th>
<th>BLEU</th>
<th>NIST</th>
<th>MET</th>
<th>ROUGE</th>
<th>CIDEr</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PREFIX</strong></td>
<td>69.7</td>
<td>8.81</td>
<td>46.1</td>
<td>71.4</td>
<td>2.49</td>
</tr>
<tr>
<td><strong>INFIX-1</strong></td>
<td>67.9</td>
<td>8.63</td>
<td>45.8</td>
<td>69.4</td>
<td>2.42</td>
</tr>
<tr>
<td><strong>INFIX-10</strong></td>
<td>67.2</td>
<td>8.48</td>
<td>45.8</td>
<td>69.9</td>
<td>2.40</td>
</tr>
<tr>
<td><strong>INFIX-20</strong></td>
<td>66.7</td>
<td>8.47</td>
<td>45.8</td>
<td>70.0</td>
<td>2.42</td>
</tr>
</tbody>
</table>
Ablation (Initialization)

- Initializing randomly performs poorly and has high variance.
- It’s better to initialize with words in the LM’s vocabulary.
- It’s even better to initialize with task specific words (summarize / table-to-text)
Key Findings

- Prefix-Tuning can come close to the performance of full model fine-tuning with much less parameters.
- Tuning less parameters leads to better generalization performance.
The Power of Scale for Parameter-Efficient Prompt Tuning

Brian Lester*  Rami Al-Rfou  Noah Constant
Google Research
{brianlester,rmyeid,nconstant}@google.com

Published in EMNLP 2021
Motivation

- Why do we need **prompt-tuning** if we already have **prefix-tuning**?
  - Prefix-tuning learns a sequence of prefixes that are prepended at **every** transformer layer
  - Prompt-tuning uses a **single** prompt representation that is prepended to the embedded input
Motivation

- Why do we need **prompt-tuning** if we already have **prefix-tuning**?
  - Prefix-tuning learns a sequence of prefixes that are prepended at **every** transformer layer
  - Prompt-tuning uses a **single** prompt representation that is prepended to the **embedded input**

The “prefix-tuning” paper: *Tuning only the embedding layer is not expressive enough*

<table>
<thead>
<tr>
<th>Model scale?</th>
<th>Prompt length?</th>
<th>Prompt initialization?</th>
<th>Pre-training settings?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>BLEU</th>
<th>NIST</th>
<th>MET</th>
<th>ROUGE</th>
<th>CIDEr</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFIX</td>
<td>69.7</td>
<td>8.81</td>
<td>46.1</td>
<td>71.4</td>
<td>2.49</td>
</tr>
<tr>
<td>EMB-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>48.1</td>
<td>3.33</td>
<td>32.1</td>
<td>60.2</td>
<td>1.10</td>
</tr>
<tr>
<td>EMB-10</td>
<td>62.2</td>
<td>6.70</td>
<td>38.6</td>
<td>66.4</td>
<td>1.75</td>
</tr>
<tr>
<td>EMB-20</td>
<td>61.9</td>
<td>7.11</td>
<td>39.3</td>
<td>65.6</td>
<td>1.85</td>
</tr>
</tbody>
</table>

Embedding-only: EMB-{PrefixLength}
Methodology: Prompt-tuning

- Prepend **virtual tokens** to input
- Prompt and input representations flow through model like normal
- Learn embeddings of **only these special tokens** via backprop. Keep the rest **fixed**.

![Diagram](Image Credit: Brian Lester)
Methodology: Prompt-tuning

- Prepend **virtual tokens** to input
- Prompt and input representations flow through model like normal
- Learn embeddings of **only these special tokens** via backprop. Keep the rest **fixed**.

\[
\begin{align*}
\text{Pr}_\theta(Y|X) &\quad \text{Pr}_{\theta;\theta_P}(Y|[P; X]) \\
\quad &\quad \text{fixed} \quad \text{learnable, } <<\theta
\end{align*}
\]
Experiment Setup

- **Model backbone**: pre-trained T5 models (Small, Base, Large, XL, XXL)
- **Design decisions to explore**
  - Prompt initialization method
  - Prompt length
  - Pre-training method
  - LM adaptation steps (if use “LM adaptation” as the pre-training method)
- **Benchmark**: SuperGLUE (a collection of eight challenging English language understanding tasks)
  - Each prompt trains on a single task
  - Each dataset is translated into a text-to-text format
  - Task names prepended to inputs are omitted
Design Decision: Prompt Initialization

1. **Random initialization**: train the prompt representations from scratch
2. **Sampled vocabulary**: initialize each prompt token to an embedding drawn from the model’s vocabulary
3. **Class label**: initialize the prompt with embeddings that enumerate the output classes
Design Decision: Prompt Initialization

1. **Random initialization**: train the prompt representations from scratch
2. **Sampled vocabulary**: initialize each prompt token to an embedding drawn from the model’s vocabulary
3. **Class label**: initialize the prompt with embeddings that enumerate the output classes

\[
v(1) = \text{terrible} \quad v(2) = \text{bad} \quad v(3) = \text{okay} \quad v_1(0) = \text{Wrong} \quad v_1(1) = \text{Right} \quad v_1(2) = \text{Maybe} \\
v(4) = \text{good} \quad v(5) = \text{great} \quad v_2(0) = \text{No} \quad v_2(1) = \text{Yes} \quad v_2(2) = \text{Maybe}
\]

Yelp dataset  
(Zhang et al., 2015)

MNLI dataset  
(Williams et al., 2018)
Design Decision: Prompt Initialization

- **Class label** based initialization performs best
- The gaps between different initializations **disappear** when the model is **scaled to XXL size**
Design Decision: Prompt Length

- The shorter the prompt, the fewer new parameters must be tuned
Design Decision: Prompt Length

- The shorter the prompt, the fewer new parameters must be tuned

Observations

- **Increasing prompt length** is critical to achieve good performance
- The largest model still gives strong results with a **single-token** prompt
- Increasing beyond 20 tokens only yields marginal gains
Design Decision: Pre-training Method

- **Span Corruption**: The model is tasked with “reconstructing” masked spans in the input text, which are marked with unique sentinel tokens.

Pre-training

Input: Thank you <X> me to your party <Y> week
Output: <X> for inviting <Y> last <Z>
Design Decision: Pre-training Method

- **Span Corruption**: The model is tasked with “reconstructing” masked spans in the input text, which are marked with unique sentinel tokens.
- **Span Corruption + Sentinel**: prepend all downstream targets with a sentinel

**Example**

Input: Thank you <X> me to your party <Y> week
Output: <X> for inviting <Y> last <Z>
Design Decision: Pre-training Method

- **Span Corruption**: The model is tasked with “reconstructing” masked spans in the input text, which are marked with unique sentinel tokens.
- **Span Corruption + Sentinel**: prepend all downstream targets with a sentinel
- **“LM Adaptation”**: given a natural text prefix as input, the model must produce the natural text continuation as output

---

Raffel et al. (2020)
Design Decision: Pre-training Method

- “Span corruption” objective is **not well-suited** for pre-training frozen models
- Adding a sentinel to the downstream targets has **little benefit**
- “LM Adaptation” performs best for all sizes of models
- The **largest** model is the **most forgiving** one
Design Decision: LM Adaptation Steps

- Longer adaptation provides additional gains, up to 100K steps
- At the largest model size, the gains from adaptation are quite modest
Closing the Gap

- **Design configuration**
  - Prompt initialization: class label
  - Prompt length: 100
  - Pre-training method: LM adaptation
  - LM adaptation steps: 100K
Closing the Gap

- **Design configuration**
  - Prompt initialization: class label
  - Prompt length: 100
  - Pre-training method: LM adaptation
  - LM adaptation steps: 100K

- **Prompt tuning becomes more competitive** with model tuning as scale increases

- **Prompt tuning beats GPT-3 prompt design by a large margin**
Q2. Comparing "model tuning", "prompt tuning" and "prompt design" in Lester et al., 2021, how do their task performances scale with model sizes? Why do you think it is the case?
Closing the Gap

Q2. Comparing "model tuning", "prompt tuning" and "prompt design" in Lester et al., 2021, how do their task performances scale with model sizes? Why do you think it is the case?

● Scalability
  ○ Model tuning: fairly good → poor
  ○ Prompt tuning: very good
  ○ Prompt design: poor

● Reason
  ○ Model tuning: possibly over-parameterized when model size is large
  ○ Prompt tuning: reserving general understanding and including tunable parameters
  ○ Prompt design: not fitting downstream tasks well
Q2. Comparing "model tuning", "prompt tuning" and "prompt design" in Lester et al., 2021, how do their task performances scale with model sizes? Why do you think it is the case?

When the model size is relatively small, “model tuning” outperforms “prompt tuning” by a small margin. However, “prompt tuning” becomes more competitive with “model tuning” as scale increases. “Prompt tuning” beats “prompt design” by a large margin with all sizes of models.

In “prompt design”, prompts are chosen manually and there is no tunable task-specific parameters. Thus, the model can’t fit the downstream task well. When the model size is large, “model tuning” may be over-parameterized and more prone to overfit the training task. “Prompt tuning” prevents the model from modifying its general understanding of language and also includes tunable task-specific prompt representations.
Comparison to Similar Approaches

- Compared with prefix-tuning
  - Tuning only the embedding layer
  - No need for prefix reparameterization

- Compared with WARP
  - Less extra parameters
  - WARP can only work on classification
Resilience to Domain Shift

- Tasks: question answering (QA) and paraphrase detection
- Question answering
  - MRQA 2019 (Fisch et al., 2019) shared task on generalization
  - It collects extractive QA datasets in a unified format
  - Train on “in-domain” datasets and evaluate on “out-of-domain” datasets
Resilience to Domain Shift

- Tasks: question answering (QA) and paraphrase detection
- Question answering
  - MRQA 2019 ([Fisch et al., 2019](#)) shared task on generalization
  - It collects extractive QA datasets in a unified format
  - Train on “in-domain” datasets and evaluate on “out-of-domain” datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Domain</th>
<th>Model</th>
<th>Prompt</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD</td>
<td>Wiki</td>
<td>94.9 ±0.2</td>
<td>94.8 ±0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>in-domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TextbookQA</td>
<td>Book</td>
<td>54.3 ±3.7</td>
<td>66.8 ±2.9</td>
<td>+12.5</td>
</tr>
<tr>
<td>BioASQ</td>
<td>Bio</td>
<td>77.9 ±0.4</td>
<td>79.1 ±0.3</td>
<td>+1.2</td>
</tr>
<tr>
<td>RACE</td>
<td>Exam</td>
<td>59.8 ±0.6</td>
<td>60.7 ±0.5</td>
<td>+0.9</td>
</tr>
<tr>
<td>RE</td>
<td>Wiki</td>
<td>88.4 ±0.1</td>
<td>88.8 ±0.2</td>
<td>+0.4</td>
</tr>
<tr>
<td>out-of-domain</td>
<td>Movie</td>
<td>68.9 ±0.7</td>
<td>67.7 ±1.1</td>
<td>-1.2</td>
</tr>
<tr>
<td>DuoRC</td>
<td>Wiki</td>
<td>68.9 ±1.7</td>
<td>67.1 ±1.9</td>
<td>-1.8</td>
</tr>
</tbody>
</table>
Resilience to Domain Shift

- Tasks: question answering (QA) and paraphrase detection
- Question answering
- Paraphrase detection
  - Transfer between two paraphrase tasks from GLUE (Wang et al., 2019b)
  - QQP (Iyer et al., 2017): ask if two questions from Quora are “duplicates”.
  - MRPC (Dolan and Brockett, 2005): asks if two sentences drawn from news articles are paraphrases
Resilience to Domain Shift

- Tasks: question answering (QA) and paraphrase detection
- Question answering
- Paraphrase detection
  - Transfer between two paraphrase tasks from GLUE (Wang et al., 2019b)
  - QQP (Iyer et al., 2017): ask if two questions from Quora are “duplicates”.
  - MRPC (Dolan and Brockett, 2005): asks if two sentences drawn from news articles are paraphrases

<table>
<thead>
<tr>
<th>Train</th>
<th>Eval</th>
<th>Tuning</th>
<th>Accuracy</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>QQP</td>
<td>MRPC</td>
<td>Model</td>
<td>73.1 ±0.9</td>
<td>81.2 ±2.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prompt</td>
<td><strong>76.3 ±0.1</strong></td>
<td><strong>84.3 ±0.3</strong></td>
</tr>
<tr>
<td>MRPC</td>
<td>QQP</td>
<td>Model</td>
<td>74.9 ±1.3</td>
<td>70.9 ±1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prompt</td>
<td><strong>75.4 ±0.8</strong></td>
<td>69.7 ±0.3</td>
</tr>
</tbody>
</table>
Resilience to Domain Shift

- Tasks: question answering (QA) and paraphrase detection
- Question answering
- Paraphrase detection
- Conclusions
  - Prompt tuning prevents the model from modifying its **general understanding** of language
  - Model tuning may be **over-parameterized** and more prone to overfit the training task, to the detriment of similar tasks in different domains
Prompt Ensembling

- Train **N prompts** on the same task → create **N separate “models”**
- One example is **replicated** across the batch and is processed varying the prompt
Prompt Ensembling

- Train N prompts on the same task → create N separate “models”
- One example is replicated across the batch and is processed varying the prompt

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>Average</th>
<th>Best</th>
<th>Ensemble</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoolQ</td>
<td>acc.</td>
<td>91.1</td>
<td>91.3</td>
<td>91.7</td>
</tr>
<tr>
<td>CB</td>
<td>acc./F1</td>
<td>99.3 / 99.0</td>
<td>100.00 / 100.00</td>
<td>100.0 / 100.0</td>
</tr>
<tr>
<td>COPA</td>
<td>acc.</td>
<td>98.8</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>MultiRC</td>
<td>EM/F1&lt;sub&gt;a&lt;/sub&gt;</td>
<td>65.7 / 88.7</td>
<td>66.3 / 89.0</td>
<td>67.1 / 89.4</td>
</tr>
<tr>
<td>ReCoRD</td>
<td>EM/F1</td>
<td>92.7 / 93.4</td>
<td>92.9 / 93.5</td>
<td>93.2 / 93.9</td>
</tr>
<tr>
<td>RTE</td>
<td>acc.</td>
<td>92.6</td>
<td>93.5</td>
<td>93.5</td>
</tr>
<tr>
<td>WiC</td>
<td>acc.</td>
<td>76.2</td>
<td>76.6</td>
<td>77.4</td>
</tr>
<tr>
<td>WSC</td>
<td>acc.</td>
<td>95.8</td>
<td>96.2</td>
<td>96.2</td>
</tr>
<tr>
<td>SuperGLUE (dev)</td>
<td></td>
<td>90.5</td>
<td>91.0</td>
<td>91.3</td>
</tr>
</tbody>
</table>
Interpretability

- discrete token → continuous embedding: difficult to interpret
- Find the nearest neighbors to the learned prompts from the vocabulary
  - The top-5 nearest neighbors form tight semantic clusters
Interpretability

- discrete token → continuous embedding: **difficult to interpret**
- Find the nearest neighbors to the learned prompts from the vocabulary
  - The top-5 nearest neighbors form **tight** semantic clusters: **learning “word-like” representations**

\{Technology / technology / Technologies / technological / technologies\}
\{entirely / completely / totally / altogether / 100% \}
Interpretability

- discrete token → continuous embedding: difficult to interpret
- Find the nearest neighbors to the learned prompts from the vocabulary
  - The top-5 nearest neighbors form tight semantic clusters: learning “word-like” representations
  - The class labels persist through training: learning to store the expected output

Initialized with “technology”

\{Technology / technology / Technologies / technological / technologies\}
\{entirely / completely / totally / altogether / 100\%\}

Initialized with “completely”
Interpretability

- discrete token → continuous embedding: **difficult to interpret**
- Find the nearest neighbors to the learned prompts from the vocabulary
  - The top-5 nearest neighbors form **tight** semantic clusters: learning **“word-like”** representations
  - The class labels **persist** through training: learning to **store** the expected output
  - Several prompt tokens have the **same** nearest neighbors if the prompt is **long**: difficult to localize information to a specific position
Interpretability

- discrete token $\rightarrow$ continuous embedding: **difficult to interpret**
- Find the nearest neighbors to the learned prompts from the vocabulary
  - The top-5 nearest neighbors form **tight** semantic clusters: learning **“word-like”** representations
  - The class labels **persist** through training: learning to **store** the expected output
  - Several prompt tokens have the **same** nearest neighbors if the prompt is **long**: difficult to localize information to a specific position

Don’t get too attached to the findings here! A “**wayward**” behavior is observed:
Key Findings

- **Prompt tuning** is a competitive technique due to
  - Better performance **scalability** as model size increases
  - **More forgiving** for bad design decisions (when model is large)
  - Improved **generalization** ability
  - Enabling efficient high-performing **prompt ensembling**
Discussion

Q3: You have collected a dataset for a downstream task of your interest and you are asked to design a prompting tuning method to solve it, which design choices do you need to consider? Can you think of other parameter-efficient methods besides what we have seen in these two papers?