CS 601.471/671 NLP: Self-supervised Models

Homework 6: Adapting Transformers Language Models for Your Problems

For homework deadline and collaboration policy, check the calendar on the course website[*]

Name: ______________________

Collaborators, if any: ____

Sources used for your homework, if any: _________________________________

This assignment will return to basics of Transformers and dig deeper into the fundamental concepts. Additionally, we will get our hands dirty by fine-tuning Transformer language models and comparing their performance. We will also be working more closely with GPU processors on our cluster and using them with Slurm queuing system. Isn't that awesome? We tend to think it is! :)

Homework goals: After completing this homework, you should be comfortable with:

- Training a pre-trained Language Model for your tasks
- You will explore different variations of model training
- You will compare the above results with few-shot prompting of models.

How to hand in your written work: via Gradescope.

Collaboration: You can do this homework as a team of 4 people. Basically, you can write a single solution for both people and upload it as a single PDF and a single `.zip` file as a group.

Typesetting: We strongly recommend typesetting your homework, especially if you have sloppy handwriting! :) We will provide a LaTeX template for homework solutions.

[*] https://self-supervised.cs.jhu.edu/sp2024/

Homework 6, oh how you test my skill,
Fine-tuning Transformers, it's a daunting thrill,
BERT, T5, and GPT, they all await,
To see if I can prompt them to create.

First, I must understand their inner workings,
And how to train them to do my biddings,
I tweak and tune their parameters with care,
Hoping my models will be beyond compare.

With each iteration, my models learn and grow,
Their predictions becoming more accurate, I know,
But there's still work to be done, and prompts to write,
To make sure they generate what's right.

I craft my prompts with precision and care,
To guide my models and make them aware,
Of the context and task at hand,
Hoping they'll produce output that's grand.

Homework 6, you pushed me to my limit,
But in the end, I'm glad I did it,
For now I understand these Transformers so well,
And know how to prompt them to excel.
–ChatGPT March 3 2023

1 Concepts, intuitions and big picture

1. How does the BERT model expect a pair of sentences to be processed?

□ `Tokens-of-sentence-1 [SEP] Tokens-of-sentence-2`

□ `[CLS] Tokens-of-sentence-1 Tokens-of-sentence-2`

□ `[CLS] Tokens-of-sentence-1 [SEP] Tokens-of-sentence-2`

Answer: *c*

2. When should you train a new tokenizer?

□ When your dataset is similar to that used by an existing pretrained model, and you want to pretrain a new model

□ When your dataset is similar to that used by an existing pretrained model, and you want to fine-tune a new model using this pretrained model

□ When your dataset is different from the one used by an existing pretrained model, and you want to pretrain a new model

Answer: *c*

3. Select the sentences that apply to the BPE model of tokenization.

□ BPE is a subword tokenization algorithm that starts with a small vocabulary and learns merge rules.

□ BPE is a subword tokenization algorithm that starts with a big vocabulary and progressively removes tokens from it.

□ BPE tokenizers learn merge rules by merging the pair of tokens that is the most frequent.

□ A BPE tokenizer learns a merge rule by merging the pair of tokens that maximizes a score that privileges frequent pairs with less frequent individual parts.

□ BPE tokenizes words into subwords by splitting them into characters and then applying the merge rules.

□ BPE tokenizes words into subwords by finding the longest subword starting from the beginning that is in the vocabulary, then repeating the process for the rest of the text.

Answer: *a, c, e*

4. What are the labels in a masked language modeling problem?

□ Some of the tokens in the input sentence are randomly masked and the labels are the original input tokens.

□ Some of the tokens in the input sentence are randomly masked and the labels are the original input tokens, shifted to the left.

□ Some of the tokens in the input sentence are randomly masked, and the label is whether the sentence is positive or negative.

□ Some of the tokens in the two input sentences are randomly masked, and the label is whether the two sentences are similar or not.

Answer: *a*

5. When should you pretrain a new model?

□ When there is no pretrained model available for your specific language

□ When you have lots of data available, even if there is a pretrained model that could work on it

□ When you have concerns about the bias of the pretrained model you are using

Answer: *a, c*

6. Is there something wrong with the following code?

```python
from transformers import AutoTokenizer, AutoModel

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
model = AutoModel.from_pretrained("gpt2")

encoded = tokenizer("Hey!", return_tensors="pt")
result = model(**encoded)
```

□ No, it seems correct.

□ The tokenizer and model should always be from the same checkpoint.

□ It's good practice to pad and truncate with the tokenizer as every input is a batch.

Answer: *b*

2 Getting Your Attention with Self-Attention

Recall that the transformer architecture uses scaled dot-product attention to compute *attention weights*:

$$\boldsymbol{\alpha}^{(t)} = \text{softmax}\left(\frac{\mathbf{q}_t \mathbf{K}^\top}{\sqrt{h}}\right) \in [0,1]^n$$

The resulting embedding in the output of attention at position t are:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V})_t = \sum_{t'=1}^{n} \alpha_{t'}^{(t)} \mathbf{v}_{t'} \in \mathbb{R}^{1\times h},$$

where $\boldsymbol{\alpha}^{(t)} = [\alpha_0^{(t)}, \ldots, \alpha_n^{(t)}]$. The same idea can be stated in a matrix form,

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{h}}\right)\mathbf{V} \in \mathbb{R}^{n\times h}.$$

In the above equations:

- h is the hidden state dimension and n is the input sequence length;
- $\mathbf{X} \in \mathbb{R}^{n\times h}$ is the input to the attention;
- $\mathbf{x}_t \in \mathbb{R}^{1\times h}$ is the slice of $\mathbf{X}$ at position t, i.e. vector representation (embedding) of the input token at position t;
- $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{h\times h}$ are the projection matrices to build query, key and value representations;
- $\mathbf{Q} = \mathbf{X}\mathbf{W}_q \in \mathbb{R}^{n\times h}, \mathbf{K} = \mathbf{X}\mathbf{W}_k \in \mathbb{R}^{n\times h}, \mathbf{V} = \mathbf{X}\mathbf{W}_v \in \mathbb{R}^{n\times h}$ are the query, key and value representations;
- $\mathbf{q}_t = \mathbf{x}_t\mathbf{W}_q \in \mathbb{R}^{1\times h}$ is the slice of $\mathbf{Q}$ at position t. Similarly, $\mathbf{k}_t = \mathbf{x}_t\mathbf{W}_k \in \mathbb{R}^{1\times h}$ and $\mathbf{v}_t = \mathbf{x}_t\mathbf{W}_v \in \mathbb{R}^{1\times h}$.

2.1 Forget 'Softmax', 'Argmax' is the New Boss in Town!

Recall from lectures that we can think about attention as being queryable softmax pooling. In this problem, we ask you to consider a hypothetical 'argmax' version of attention where it returns exactly the value corresponding to the key that is most similar to the query, where similarity is measured using the traditional inner-product. In other words, here use a version of Attention that instead of using softmax we use argmax to generate outputs from the attention layer. For example, `softmax([1, 3, 2])` becomes `argmax([1, 3, 2]) = [0, 1, 0]`.

1. Perform argmax attention with the following keys and values:

$$\text{Keys} = \left\{ \begin{bmatrix}0\\1\\2\end{bmatrix}, \begin{bmatrix}1\\1\\2\end{bmatrix}, \begin{bmatrix}-1\\1\\-2\end{bmatrix}, \begin{bmatrix}0\\-1\\-2\end{bmatrix} \right\}, \text{Values} = \left\{ \begin{bmatrix}3\\2\\-4\end{bmatrix}, \begin{bmatrix}3\\-2\\4\end{bmatrix}, \begin{bmatrix}3\\2\\0\end{bmatrix}, \begin{bmatrix}3\\2\\4\end{bmatrix} \right\}$$

using the following query:

$$\mathbf{q} = \begin{bmatrix}-2\\1\\1\end{bmatrix}$$

What would be the output of the attention layer for this query? Remember, to simplify calculations, use an argmax instead of softmax. Note about notation: each vector in 'Keys = $\{.\}$' and 'Values$\{.\}$' are key/value vectors corresponding to four, non-interchangeable positions (i.e., ordering of these vectors matter).

Answer:

$$\mathbf{K}^\top = \begin{bmatrix}0 & 1 & -1 & 0\\1 & 1 & 1 & -1\\2 & 2 & -2 & -2\end{bmatrix}$$

$$
\begin{aligned}
\mathbf{a} &= argmax(\frac{\mathbf{q}\mathbf{K}^\top}{\sqrt{h}}) \\
&= argmax(\frac{\mathbf{q}\mathbf{K}^\top}{\sqrt{3}}) \\
&= argmax(\frac{\begin{bmatrix} -2 & 1 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 & -1 & 0 \\ 1 & 1 & 1 & -1 \\ 2 & 2 & -2 & -2 \end{bmatrix}}{\sqrt{3}}) \\
&= argmax(\frac{\begin{bmatrix} 3 & 1 & 1 & -3 \end{bmatrix}}{\sqrt{3}}) \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
Attention &= \sum_{i=1}^{4} \mathbf{a}^{(i)}\mathbf{v}_i \\
&= 1\begin{bmatrix} 3 \\ 2 \\ -4 \end{bmatrix} + 0 + 0 + 0 \\
&= \begin{bmatrix} 3 \\ 2 \\ -4 \end{bmatrix}
\end{aligned}
$$

2. How does this design choice affect our ability to usefully train models involving attention? (**Hint:** think about how the gradients flow through the network in the backward pass. Can we learn to improve our queries or keys during the training process?)
   Answer: *The argmax function is not differentiable. Thus, during backward pass, the gradient is unable to flow through the argmax operation since it does not have defined local gradient.*

2.2 Superposition of Information in Self-Attention

We will show that the attention mechanism is able to copy the information from its input, whenever needed.[*]

1. **'Copying' mechanism in self-attention:** We'll first show that it is particularly simple for attention to "copy" a value vector to the output. Describe (in 1-2 sentences) what properties of the inputs to the attention operation would result in the output $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_t$ being approximately equal to $\mathbf{v}_j$ for some $j \in [0 \ldots n]$.

   Hint: Show that there can exist $j \in [0 \ldots n]$ such that $\alpha_j^{(t)} \gg \sum_{i \neq j} \alpha_k^{(t)}$, if certain property holds for the query $\mathbf{q}_t$, and the keys $\{\mathbf{k}_0 \ldots \mathbf{k}_n\}$.
   Answer: *For this to hold for any $j \in [0, ..., n]$, the query $\mathbf{q}_t$ needs to be much more similar to the key $\mathbf{k}_j$ than any other keys $\mathbf{k}_i$ where $i \neq j$. This means that $\mathbf{q}_t \cdot \mathbf{k}_j \gg \mathbf{q}_t \cdot \mathbf{k}_i$ where $i \neq j$, so that the softmax operation will output a value approximately 1 at the j-th entry and 0 elsewhere,*

$$
\begin{aligned}
&\mathbf{q}_t \cdot \mathbf{k}_j / \sqrt{h} \gg \mathbf{q}_t \cdot \mathbf{k}_i / \sqrt{h} \; \forall i \neq j \\
\implies &\mathbf{q}_t \cdot \mathbf{k}_j / \sqrt{h} \gg \sum_{i \neq j} \mathbf{q}_t \cdot \mathbf{k}_i / \sqrt{h} \\
\implies &\frac{e^{\mathbf{q}_t \cdot \mathbf{k}_j / \sqrt{h}}}{e^{\mathbf{q}_t \cdot \mathbf{k}_j / \sqrt{h}} + \sum_{i \neq j} e^{\mathbf{q}_t \cdot \mathbf{k}_i / \sqrt{h}}} \approx 1 \text{ and } \frac{e^{\mathbf{q}_t \cdot \mathbf{k}_i / \sqrt{h}}}{e^{\mathbf{q}_t \cdot \mathbf{k}_j / \sqrt{h}} + \sum_{m \neq j} e^{\mathbf{q}_t \cdot \mathbf{k}_m / \sqrt{h}}} \approx 0 \\
\implies &\boldsymbol{\alpha}_j^{(t)} \gg \sum_{i \neq j} \boldsymbol{\alpha}_i^{(t)} \\
\implies &Attention(Q, K, V)_t = \boldsymbol{\alpha}_j^{(t)} \mathbf{v}_j + \sum_{i \neq j} \boldsymbol{\alpha}_i^{(t)} \mathbf{v}_i \approx \mathbf{v}_j
\end{aligned}
$$

[*] Question credit: John Hewitt

2. **Extracting signals after averaging them:** Instead of focusing on just one vector $\mathbf{v}_j$, attention mechanism might want to incorporate information from multiple source vectors. Consider the case where we instead want to incorporate information from two vectors $\mathbf{v}_a$ and $\mathbf{v}_b$, with corresponding key vectors $\mathbf{k}_a$ and $\mathbf{k}_b$. How should we combine information from two value vectors $\mathbf{v}_a$ and $\mathbf{v}_b$? A common way to combine values vectors is to average them: $\bar{\mathbf{v}} = \frac{1}{2}(\mathbf{v}_a + \mathbf{v}_b)$. However, after such averaging it is not quite clear how to tease apart the original information in value vectors $\mathbf{v}_a$ and $\mathbf{v}_b$. Unless ... some special properties hold!

   Suppose that although we don't know $\mathbf{v}_a$ or $\mathbf{v}_b$, we do know that $\mathbf{v}_a$ lies in a subspace[†] A formed by the m basis vectors[‡] $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m\}$, while $\mathbf{v}_b$ lies in a subspace B formed by the p basis vectors $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_p\}$. This means that any $\mathbf{v}_a$ can be expressed as a linear combination of its basis vectors, as can $\mathbf{v}_b$. All basis vectors have norm 1 and are orthogonal to each other. Additionally, suppose that the two subspaces are orthogonal; i.e. $\mathbf{a}_j^\top \mathbf{b}_k = 0$ for all j, k. Using the basis vectors $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m\}$, construct a matrix M such that for arbitrary vectors $\mathbf{v}_a \in A$ and $\mathbf{v}_b \in B$, we can use M to extract $\mathbf{v}_a$ from the average vector $\bar{\mathbf{v}}$. In other words, we want to construct M such that for any $\mathbf{v}_a$, $\mathbf{v}_b$, $M\bar{\mathbf{v}} = \mathbf{v}_a$. Show that $M\bar{\mathbf{v}} = \mathbf{v}_a$ holds for your M.

   Answer: *For any $\mathbf{v}_a \in A$ can be expressed as $\mathbf{v}_a = c_1\mathbf{a}_1 + c_2\mathbf{a}_2 + \ldots + c_m\mathbf{a}_m = \sum_{j=1}^{m} c_j\mathbf{a}_j$ where $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_m\}$ are the basis vectors of the subspace A. Similarly, for any $\mathbf{v}_b \in B$, $\mathbf{v}_b = d_1\mathbf{b}_1 + \ldots d_p\mathbf{b}_p = \sum_{k=1}^{p} d_k\mathbf{b}_k$. $\bar{\mathbf{v}} = \frac{1}{2}(\mathbf{v}_a + \mathbf{v}_b) = \frac{1}{2}(\sum_{j=1}^{m} c_j\mathbf{a}_j + \sum_{k=1}^{p} d_k\mathbf{b}_k)$*

$$\text{Let } M = 2\begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \ldots & \mathbf{a}_m\end{bmatrix}\begin{bmatrix}\mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top\end{bmatrix}$$

$$M\bar{\mathbf{v}} = 2\begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \ldots & \mathbf{a}_m\end{bmatrix}\begin{bmatrix}\mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top\end{bmatrix}\frac{1}{2}(\sum_{j=1}^{m} c_j\mathbf{a}_j + \sum_{k=1}^{p} d_k\mathbf{b}_k)$$

$$= \begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \ldots & \mathbf{a}_m\end{bmatrix}\left(\begin{bmatrix}\sum_{j=1}^{m} c_j\mathbf{a}_1^\top\mathbf{a}_j \\ \sum_{j=1}^{m} c_j\mathbf{a}_2^\top\mathbf{a}_j \\ \vdots \\ \sum_{j=1}^{m} c_j\mathbf{a}_m^\top\mathbf{a}_j\end{bmatrix} + \begin{bmatrix}\sum_{k=1}^{p} d_k\mathbf{a}_1^\top\mathbf{b}_k \\ \sum_{k=1}^{p} d_k\mathbf{a}_2^\top\mathbf{b}_k \\ \vdots \\ \sum_{k=1}^{p} d_k\mathbf{a}_m^\top\mathbf{b}_k\end{bmatrix}\right)$$

since $\mathbf{a}_j^\top\mathbf{b}_k = 0\ \forall j, k$ and $\mathbf{a}_i^\top\mathbf{a}_i = 1$ and $\mathbf{a}_i^\top\mathbf{a}_u = 0\ \forall u \neq i$,

$$= \begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \ldots & \mathbf{a}_m\end{bmatrix}\left(\begin{bmatrix}c_1\mathbf{a}_1^\top\mathbf{a}_1 \\ c_2\mathbf{a}_2^\top\mathbf{a}_2 \\ \vdots \\ c_m\mathbf{a}_m^\top\mathbf{a}_m\end{bmatrix} + \mathbf{0}\right)$$

$$= \begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \ldots & \mathbf{a}_m\end{bmatrix}\begin{bmatrix}c_1 \\ c_2 \\ \vdots \\ c_m\end{bmatrix}$$

$$= \sum_{j=1}^{m} c_j\mathbf{a}_j$$

$$= \mathbf{v}_a$$

3. **Averaging pairs of representations:** As before, let $\mathbf{v}_a$ and $\mathbf{v}_b$ be two value vectors corresponding to key vectors $\mathbf{k}_a$ and $\mathbf{k}_b$, respectively. Assume that (1) all key vectors are orthogonal, so $\mathbf{k}_i^\top\mathbf{k}_j = 0$ for all $i \neq j$; and (2) all

[†] https://en.wikipedia.org/wiki/Linear_subspace

[‡] https://en.wikipedia.org/wiki/Basis_(linear_algebra)

key vectors have norm 1. Find an expression for a query vector $\mathbf{q}_t$ such that $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_t = \frac{1}{2}(\mathbf{v}_a + \mathbf{v}_b)$ and justify your answer.

Answer: *Let* $p \to \infty$*. Then let* $\mathbf{q}_t = (\mathbf{k}_a + \mathbf{k}_b - p\sum_{i \neq a, i \neq b} \mathbf{k}_i)^\top$*. Since all key vectors are orthogonal, We have*

$$\mathbf{q}_t \mathbf{k}_a = (\mathbf{k}_a + \mathbf{k}_b - p \sum_{i \neq a, i \neq b} \mathbf{k}_i)\mathbf{k}_a = \mathbf{k}_a^\top \mathbf{k}_a + \mathbf{k}_b^\top \mathbf{k}_a - p \sum_{i \neq a, i \neq b} \mathbf{k}_i^\top \mathbf{k}_a = \mathbf{k}_a^\top \mathbf{k}_a = 1$$

$$\mathbf{q}_t \mathbf{k}_b = (\mathbf{k}_a + \mathbf{k}_b - p \sum_{i \neq a, i \neq b} \mathbf{k}_i)\mathbf{k}_b = \mathbf{k}_a^\top \mathbf{k}_b + \mathbf{k}_b^\top \mathbf{k}_b - p \sum_{i \neq a, i \neq b} \mathbf{k}_i^\top \mathbf{k}_b = \mathbf{k}_b^\top \mathbf{k}_b = 1$$

$$\forall j \neq a, j \neq b,\ \mathbf{q}_t \mathbf{k}_j = (\mathbf{k}_a + \mathbf{k}_b - p \sum_{i \neq a, i \neq b} \mathbf{k}_i)\mathbf{k}_j = \mathbf{k}_a^\top \mathbf{k}_j + \mathbf{k}_b^\top \mathbf{k}_j - p \sum_{i \neq a, i \neq b} \mathbf{k}_i^\top \mathbf{k}_j = -p\mathbf{k}_j^\top \mathbf{k}_j = -p \to -\infty$$

Hence, $\boldsymbol{\alpha}_a^{(t)} = \boldsymbol{\alpha}_b^{(t)} = \frac{e^{1\sqrt{h}}}{e^{1\sqrt{h}} + e^{1\sqrt{h}} + \sum_{i \neq a, i \neq b} e^{-p/\sqrt{h}}} \approx \frac{e^{1\sqrt{h}}}{e^{1\sqrt{h}} + e^{1\sqrt{h}} + 0} = \frac{1}{2}$.

For $j \neq a, j \neq b$, $\boldsymbol{\alpha}_j^{(t)} = \frac{e^{-p/\sqrt{h}}}{e^{1\sqrt{h}} + e^{1\sqrt{h}} + \sum_{i \neq a, i \neq b} e^{-p/\sqrt{h}}} \approx \frac{0}{e^{1\sqrt{h}} + e^{1\sqrt{h}} + 0} = 0$.

$Attention(Q, K, V)_t = \sum_{i=1}^n \boldsymbol{\alpha}_i^{(t)} \mathbf{v}_i = \boldsymbol{\alpha}_a^{(t)} \mathbf{v}_a + \boldsymbol{\alpha}_b^{(t)} \mathbf{v}_b + \sum_{i \neq a, i \neq b} \boldsymbol{\alpha}_i^{(t)} \mathbf{v}_i = \frac{1}{2}\mathbf{v}_a + \frac{1}{2}\mathbf{v}_b + 0 = \frac{1}{2}(\mathbf{v}_a + \mathbf{v}_b)$

2.3 Extra Credit: Importance of 'Scaling' in Self-Attention

In practice, we scale each dot product $\mathbf{QK}^\top$ by a factor of $\sqrt{h}$. This is called *scaled dot product attention*. In this part, we will prove why we perform this scaling. Suppose we are performing a dot product between a key $\mathbf{k}$ and query $\mathbf{q}$, where $\mathbf{k}, \mathbf{q} \in \mathbb{R}^{1 \times h}$ and $\mathbf{k}, \mathbf{q} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$. You can assume that $\mathbf{k}, \mathbf{q}$ are sampled independently.

1. Compute $\mathbf{E}[\mathbf{q}\mathbf{k}^\top]$ in terms of $\boldsymbol{\mu}, h, \sigma$.

   Answer: *Let* $\boldsymbol{\mu}$ *be of shape* $1 \times h$*. By independence*

$$\begin{aligned} \mathbf{E}[\mathbf{q}\mathbf{k}^\top] &= \mathbf{E}[\mathbf{q}]\mathbf{E}[\mathbf{k}^\top] \\ &= \boldsymbol{\mu}\boldsymbol{\mu}^\top \\ &= \sum_{i=1}^h \mu_i^2 \end{aligned}$$

2. Compute $\mathbf{Var}[\mathbf{q}\mathbf{k}^\top]$ in terms of $\boldsymbol{\mu}, h, \sigma$.

   Answer:

$$\begin{aligned} \boldsymbol{Var}[\boldsymbol{qk}^\top] &= \boldsymbol{Var}[\sum_{i=1}^h \mathbf{q}_i \mathbf{k}_i] \\ &= \sum_{i=1}^h \boldsymbol{Var}[\mathbf{q}_i \mathbf{k}_i] \text{ by independence} \\ &= \sum_{i=1}^h \left(\mathbf{E}[\mathbf{q}_i^2 \mathbf{k}_i^2] - \mathbf{E}^2[\mathbf{q}_i \mathbf{k}_i] \right) \\ &= \sum_{i=1}^h \left(\mathbf{E}[\mathbf{q}_i^2]\mathbf{E}[\mathbf{k}_i^2] - (\mathbf{E}[\mathbf{q}_i]\mathbf{E}[\mathbf{k}_i])^2 \right) \text{ by independence} \\ &= \sum_{i=1}^h \left(\mathbf{E}[\mathbf{q}_i^2]\mathbf{E}[\mathbf{k}_i^2] - (\mu_i \mu_i)^2 \right) \\ &= \sum_{i=1}^h \left((\mathbf{E}^2[\mathbf{q}_i] + Var(\mathbf{q}_i))(\mathbf{E}^2[\mathbf{k}_i] + Var(\mathbf{k}_i) - \mu_i^4 \right) \quad \text{since } \mathbf{E}[X^2] = \boldsymbol{Var}(X) + \mathbf{E}^2[X] \\ &= \sum_{i=1}^h \left((\mu_i^2 + \sigma^2)(\mu_i^2 + \sigma^2) - \mu_i^4 \right) \\ &= \sum_{i=1}^h \left(\sigma^4 + 2\mu_i^2 \sigma^2 \right) \end{aligned}$$

3. Based on the variance computed above and assuming that $\boldsymbol{\mu} = 0$ and $\sigma = 1$, explain why we need to scale the dot product by $\sqrt{h}$. Explain why this scaling is important for numerical stability.
Answer: *In this case,* $\mathbf{Var}[\boldsymbol{q}\boldsymbol{k}^\top] = \sum_{i=1}^{h} \left(\sigma^4 + 2\mu_i^2\sigma^2\right) = \sum_{i=1}^{h}(1+0) = h.$
This scaling would give $\mathbf{Var}[\boldsymbol{q}\boldsymbol{k}^\top/\sqrt{h}] = \mathbf{Var}[\boldsymbol{q}\boldsymbol{k}^\top]/h = \sum_{i=1}^{h} \left(\sigma^4 + 2\mu_i^2\sigma^2\right)/h = \sum_{i=1}^{h}(1+0)/h = 1.$
Such scaling helps ensure unit variance before passing it to the softmax. When h is large, making the original variance also large, this scaling can help scale down the variance and prevent exploding gradients due to large values in the unnormalized distribution. Additionally, making sure the variance of the inputs to softmax is unit can prevent the softmax function from assigning a large score at a specific position due to large differences in the inputs, so this rescaling helps the model incorporate information from multiple value vectors.

3 Programming

In this programming homework, we will finetune encoder LMs for a classification task with yes/no questions. In particular, we will

- Implement full-parameter finetuning.
- Explore various parameter-efficient finetuning strategies.

Additionally, we will also experiment with in-context learning using GPT-3.5 models via OpenAI APIs.

Skeleton Code and Structure: The code base for this homework can be found at this GitHub repo under the `hw6` directory. Your task is to fill in the missing parts in the skeleton code, following the requirements, guidance, and tips provided in this pdf and the comments in the corresponding .py files. The code base has the following structure:

- `base_classification.py`: implements the basic full finetuning
- `classification.py`: implements different parameter-efficient finetuning

TODOs — Your tasks include 1) generate plots and/or write short answers based on the results of running the code; 2) fill in the blanks in the skeleton to complete the code. We will explicitly mark these plotting, written answer, and filling-in-the-blank tasks as **TODOs** in the following descriptions, as well as a `# TODO` at the corresponding blank in the code.

Submission: Your submission should contain two parts: 1) plots and short answers under the corresponding questions below; and 2) your completion of the skeleton code base, in a `.zip` file, which includes the two provided skeleton files and a `openai_gpt-boolq.py` script you created (details in subsection 3.3), and additionally a `classification_lora.py` if you implement extra credit question. (details in item 6)

Overview and Dataset In this programming assignment, you will get your hands dirtier with building self-supervised models. You will build a classifier using the Huggingface Transformer library and PyTorch. This classifier is expected to solve the BoolQ dataset [Clark et al., 2019]. You can find this dataset on Huggingface's dataset hub: `https://huggingface.co/datasets/boolq`. In this task, each input consists of a question statement and a passage that may contain the answer to this question. The output is the answer to the question which may be one of 'Yes' or 'No' labels. Here is an example:

```
{
    "passage": "\"All biomass goes through at least some of these steps: it needs to be grown,
                                                     collected, dried, fermented, distilled, and
                                                     burned...",
    "question": "does ethanol take more energy make that produces"
    "answer": false,
 }
```

The data comes with training and validation subsets. We will select a subset of the training data as 'test' set. So for the main experiments we will use 8k instances for training and the rest for testing.

sbatch Scripts You can create your own sbatch scripts for the following **TODOs** whenever you need to run GPU jobs on the Rockfish. You can use the `sbatch.sh` in homework 5 as a reference.

Python Scripts Arguments Many of the **TODOs** below involves setting different training configurations and hyper-parameters, this could be achieved through the arguments provided in `main` of both `base_classification.py` and `classification.py` - they should be self-explanatory.

General Hint Follow the specifications and requirements in the code comments to complete missing lines.

3.1 Finetuning: Basics

1. Let's start with the basic full finetuning, where all the parameters will be updated during the training. Specifically, we will use AutoModelForSequenceClassification which implements a classifier that maps the representation of encoder LMs (BERT, RoBERTa, etc.) into a fixed set of labels. Use this class to build a binary classifier using the representations of distilbert-base-uncased [Sanh et al., 2020]. The starter code for full finetuning in base_classification.py has a bunch of basic functionalities implemented, but it is missing details.

   TODOs: Read and complete the missing lines in evaluate_model and train functions in base_classification.py to complete the functionalities of full finetuning.

   To make sure your implementation is correct, we can train the model on 10 training instances for a few epochs (say, a total of 20 epochs). Note that over-fitting a few training examples is a good first test to make sure your implementation can successfully learn the patterns of your data. You should be able to complete the starter code and run it on a few instances in your local machine. You should be able to see that your classifier successfully over-fits its few training examples.

   TODOs Run base_classification.py with the following command on your local machine, paste the generated plot of training accuracy vs. epochs plots below.

```
> python base_classification.py --small_subset --device cuda --model "distilbert-base-uncased"
                                                    --batch_size "64" --lr 1e-4 --num_epochs 20
```

   your plot

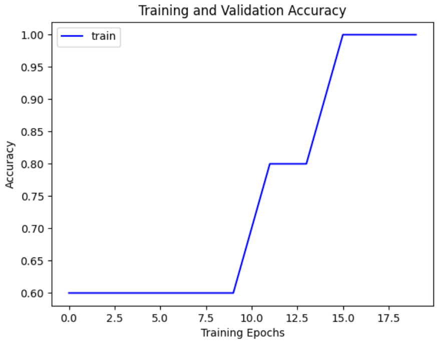


Figure 1: Q 3.1.1

2. Now, rather than running training on a subset of the data, try training with all the data on Rockfish. Make sure to find the largest batch size that can fit in the Rockfish GPUs. You can start with a reasonable number, say 32. If it fits, double it to 64. If it does not fit, halve the value to 16.

   TODOs Train for 30 epochs and paste the generated plot of train and dev accuracies vs. epochs below. Also report the largest batch size.
   Hint: You should see that training accuracy goes high, while the dev accuracy will go high and then turn downhill as the model overfits the data.
   your plot

   Answer: *Largest batch size is 64.*

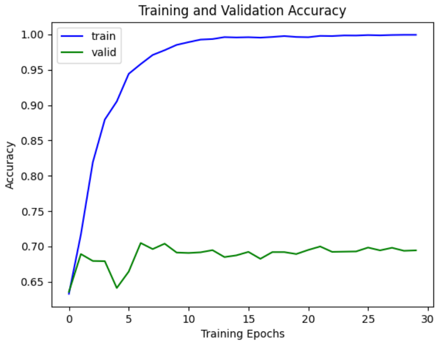


Figure 2: Q 3.1.2

3. Let's implement hyper-parameter selection by exploring the combination of different hyper-parameter choices.

   TODOs: Train models for various choices of learning rates (`1e-4, 5e-4, 1e-3`) and training epochs (`7, 9`). Select the model with the highest accuracy on the dev set and report its accuracy on the test set. Also report the corresponding choice of hyperparameters
   Answer:
   Best accuracy of distilBERT-base-uncased on the dev set: 0.7116207951070337
   Best accuracy of distilBERT-base-uncased on the test set: 0.708479327259986

   Learning rate:1e-4
   Training epochs:7

4. **TODOs** Now repeat the previous experiment with other models. In particular, create a bar plot that shows test/dev accuracies for two of your favorite of the following models: `distilBERT-base-uncased, BERT-base-uncased, BERT-large-uncased, BERT-base-cased, BERT-large-cased, RoBERTa-base, RoBERTa-large`. Note, it is possible that some of these models wouldn't fit in Rockfish GPUs for any choice of batch size, in which case report 0 for their performance.
   your plot

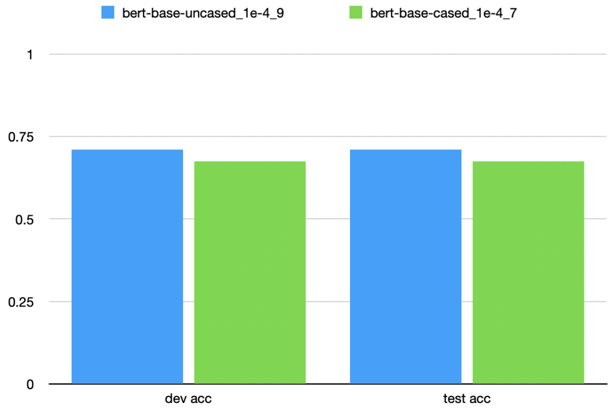


Figure 3: Q 3.1.4

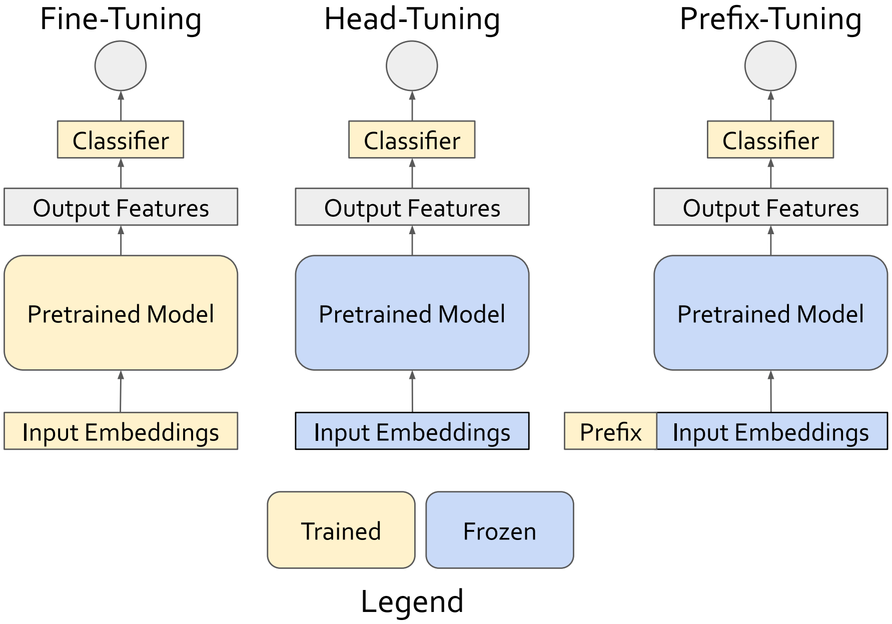


Figure 4: Overview of fine-tuning methods. Blue indicates parts that are frozen (no gradient update) during the fine-tuning process. Yellow indicates parts that are trained (with gradient updates).

3.2 Parameter-Efficient Fine-Tuning

In this section, we will implement two more efficient ways of tuning language models: **Head-Tuning**, which freezes all parameters but the classification head, and **Prefix-Tuning**, which also freezes all parameters but the classification head, and additionally appends a fixed length ℓ of trainable embeddings to the input embeddings. See figure 4 for an illustration of which parameters are getting updated in each fine-tuning method we will explore in this homework.

Note that "output features" tensor for a single sequence of input should be of shape $(\text{length}, \text{hidden size})$, or equivalently $\mathbb{R}^{N\times d}$. However, the classifier expects a tensor of shape $\mathbb{R}^{1\times d}$. There are multiple ways to transform the output features to $\mathbb{R}^{1\times d}$, but in this work, we will take the **averaged** token representation.

Instead of using `AutoModelForSequenceClassification`, we now use the base `AutoModel` class to tailor the model to our needs. But don't worry, a skeleton code is provided to you in `classification.py`, you would only need to modify the details.

1. For comparison, let's redo the full finetuning first. Change the base model to `RoBERTa-base`. Implement full fine-tuning by adding a binary classifier on the averaged sentence representation.

   TODOs: Read and complete the missing lines

   (a) in `evaluate_model` function

   (b) after `for index, batch in tqdm(enumerate(train_dataloader)):` in `train` function

   in `classification.py` to complete the functionalities of full finetuning.
   Hint: Your implementation should be the same as that of `base_classification.py` for `evaluate_model` and almost the same except for how we get the logtis for `train` (until loss backward).

   TODOs: Read and complete the missing lines after `if self.type == 'full'` in the `forward` function of the `CustomModelforSequenceClassification` class in `classification.py`. Run the code with `type="full"` to train the model with full finetuning. Report the final accuracy you obtained along with the hyper-parameters you used (number of epochs and learning rate).
   Answer:
   dev accuracy: 0.713
   test accuracy: 0.696

epochs: 5
learning rate: 5e-5

2. **TODOs**: Write down the number of tuned parameters for head-tuning for BoolQ, given that the model hidden size is d.
   Answer: *The number of tuned parameters for head-tuning is the number of parameters in the classifier. Therefore,*

$$\#parameters = input\_size * output\_size + \#bias = d * 2 + 2$$

Now let's implement head-tuning by adding a binary classifier on the averaged sentence representation.

TODOs: Read and complete the missing lines after `if self.type == ‘head’` in the `forward` function of the `CustomModelforSequenceClassification` class in `classification.py`. You would also need to modify the optimizer by adding code after `mymodel.type == ‘‘head":` in the `train` function in `classification.py`. Run the code with `type="head"` to train the model with head-tuning. Report the final accuracy on full head-tuning along with the hyper-parameters you used[§].
Answer:
dev accuracy: 0.6449541284403669
test accuracy: 0.6412053258584443
epochs: 12
learning rate: 1e-2

3. **TODOs**: Write down the number of tuned parameters for our prefix-tuning on BoolQ that appends trainable prefix embeddings of length 128, given the model hidden size is d.
   Answer: *The number of parameters for prefix-tuning the is number of parameters in the classifier plus the number of parameters in the prefix. Therefore,*

$$\#parameters = \#classifier + prefix\_length * hidden\_size = d * 2 + 2 + 128 * d = 130 * d + 2$$

It's time to implement prefix tunning!

TODOs: Read and complete the missing lines after `if self.type == ‘prefix’` in the `forward` function of the `CustomModelforSequenceClassification` class in `classification.py`. You would also need to modify the optimizer by adding code after `mymodel.type == ‘‘prefix":` in the `train` function in `classification.py`.Run the code with `type="prefix"` to train the model with prefix-tuning. Report the final accuracy of prefix-tuning along with the hyper-parameters you used [¶].
Answer:
dev accuracy:0.6507645259938838
test accuracy:0.6334968465311843
epochs:9
learning rate:5e-3

4. **Extra Credit:** Here is a simple tutorial on how to use torch.profile to record the memory usage of forward and backward operations.

   TODOs Generate a bar plot of the forward/backward memory usage of full-finetuning, head-tuning, and prefix-tuning. Note that you don't need to upload any code for this.
   your plot

[§] You would be only graded by the correctness of your implementation, but if you want good performance, Try using a larger learning rate and longer training epochs.

[¶] Again, you would be only graded by the correctness of your implementation, but if you want good performance, Try using a larger learning rate and longer training epochs.

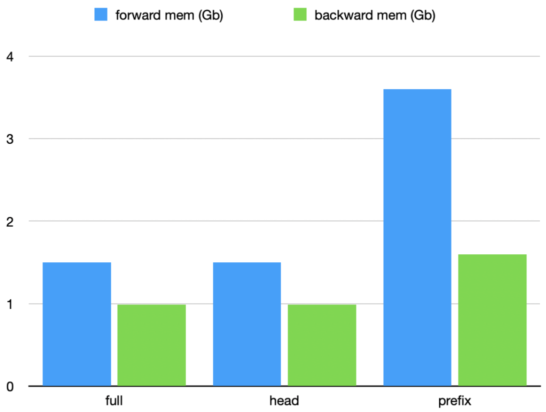


Figure 5: Q3.2.4

5. **LoRA** fine-tuning [Hu et al., 2022] decomposes the parameter updates of a large dense matrix to two low-rank matrices. For a matrix M of size $\mathbb{R}^{d \times d}$, the full updates require storing the gradients and optimizer states of $d \times d$ parameters.

   TODOs: However, if we decompose the gradient update on the full matrix $\Delta M \approx AB$ where $A \in \mathbb{R}^{d \times n}$ and $B \in \mathbb{R}^{n \times d}$, write down the number of parameters whose gradients and optimizer states we need to store:
   Answer:

   $$d \times n + d \times n = 2dn$$

   .

6. **Extra Credit:** Implement **LoRA** fine-tuning.

   TODOs: Create `classification_lora.py` and copy `classification_base.py` to it and implement the LoRA fine-tuning method using the peft library. This should be just changing a few lines of code. Report the trainable parameters and final accuracy of your LoRA fine-tuning, as well as your hyper-parameters.
   Answer:
   dev accuracy: 0.7379204892966361
   test accuracy: 0.744218640504555
   epochs: 9
   learning rate: 1e-4

3.3 In-Context Learning

1. Here we will prompt language models.[||] You should sign up for the OpenAI API, which lets you use GPT-3 a large, neural language model like the ones that we learned about in lecture.

   The OpenAI API is a paid service. OpenAI will give you a few dollars in credit when you first create your account (See the usage page). For this assignment, the cost should be less than that. For the first part of the assignment, we'll get warmed up by playing with the OpenAI API via its interactive Playground website. Later we'll see how to integrate it directly into our code.

   OpenAI OpenAI products offer a wide spectrum of capabilities and modes of interaction with their models. For this homework, **we use the** `Completion` **mode for both playground and API calls**. In the future, you can explore other modes that might be more up-to-date and powerful, e.g. `Chat`(Playground, APIs), `Assistants` (Playground, APIs), and many more!
   Note: we suggest you to prioritize your credit usage on finishing this homework question with `Completion`

[||] Question credit: Chris Callison-Burch's AI course at UPenn.

before exploring other modes, as querying powerful models might be more expensive.

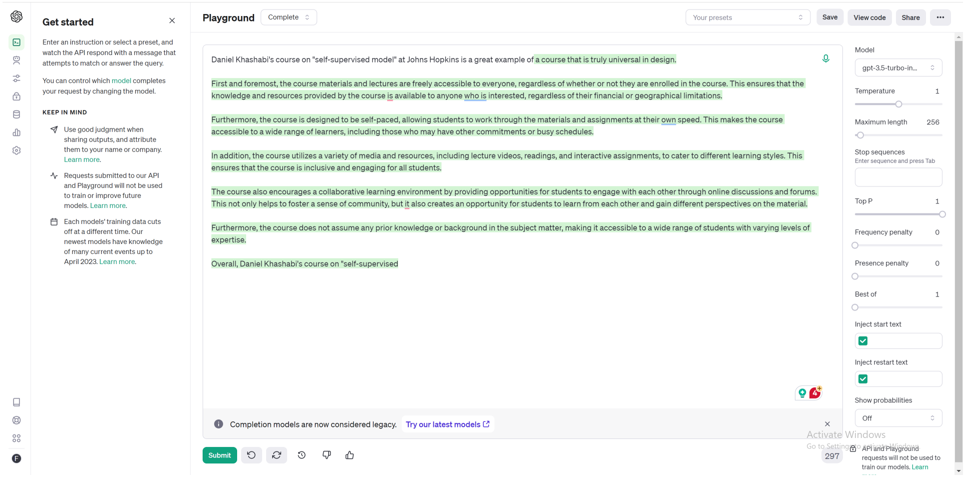
First, let's learn some basic terminology:

- Prompt: the input to the model
- Completion: what the model outputs

Let's try it out.
TODOs: Paste this prompt into the playground, press the "Generate" button, and see what it says. Paste a screenshot of the completion below:

```
Daniel Khashabi's course on "self-supervised model" at Johns Hopkins is a great example of
```

your plot



Enter an instruction or select a preset, and watch the API respond with a message that attempts to match or answer the query.

You can control which model completes your request by changing the model.

KEEP IN MIND

- Use good judgment when sharing outputs, and attribute them to your name or company. Learn more.
- Requests submitted to our API and Playground will not be used to train or improve future models. Learn more.
- Each models' training data cuts off at a different time. Our newest models have knowledge of many current events up to April 2023. Learn more.

Daniel Khashabi's course on "self-supervised model" at Johns Hopkins is a great example of a course that is truly universal in design.

First and foremost, the course materials and lectures are freely accessible to everyone, regardless of whether or not they are enrolled in the course. This ensures that the knowledge and resources provided by the course is available to anyone who is interested, regardless of their financial or geographical limitations.

Furthermore, the course is designed to be self-paced, allowing students to work through the materials and assignments at their own speed. This makes the course accessible to a wide range of learners, including those who may have other commitments or busy schedules.

In addition, the course utilizes a variety of media and resources, including lecture videos, readings, and interactive assignments, to cater to different learning styles. This ensures that the course is inclusive and engaging for all students.

The course also encourages a collaborative learning environment by providing opportunities for students to engage with each other through online discussions and forums. This not only helps to foster a sense of community, but it also creates an opportunity for students to learn from each other and gain different perspectives on the material.

Furthermore, the course does not assume any prior knowledge or background in the subject matter, making it accessible to a wide range of students with varying levels of expertise.

Overall, Daniel Khashabi's course on "self-supervised

Completion models are now considered legacy. Try our latest models

297

Model: gpt-3.5-turbo-in...
Temperature: 1
Maximum length: 256
Stop sequences: Enter sequence and press Tab
Top P: 1
Frequency penalty: 0
Presence penalty: 0
Best of: 1
Inject start text: ☑
Inject restart text: ☑
Show probabilities: Off

API and Playground requests will not be used to train our models. Learn

Activate Windows
Go to Settings to activate Windows.

Now save its output for the end of the semester for your course reviews ... just kidding!!

2. There are several controls on the right hand side of the playground. These are:

   - **Model:** GPT models come in several different sizes and capabilities. You can read more documentation on these models here. For our homework, we use the `gpt-3.5-turbo-instruct` model.
   - **Temperature and Top P sampling:** control how the model samples tokens from its distribution. Setting Temperature to 0 will cause the model to produce the highest probability output. Setting it closer to 1 will increase its propensity to create more diverse output.
   - **Maximum length:** what's the maximum length (in tokens) that the model will output?
   - **Stop sequences:** you can specify what tokens should cause the model to stop generating. You can use the newline character or any special sequence that you designate.
   - **Frequency Penalty and Presence Penalty:** two parameters that help to limit how much repetition there is in the model's output.
   - **Best of:** Best of n indicates the model generate n completion candidates on the server side, and display the top-1 on the client (user) side.

   In-context learning: In addition to writing awesome reviews of your professors, you can design prompts to get GPT-3 to do all sorts of surprising things. For instance, GPT-3 can perform few-shot learning. Given a few examples of a task, it can learn a pattern very quickly and then be used for classification tasks. It often helps to tell the model what you want it to do.

   Here's an example from the paper that introduced GPT-3. It shows a few-show learning example for correcting grammatically incorrect English sentences.

```
Poor English input:  I eated the purple berries.
Good English output:  I ate the purple berries.

Poor English input:  Thank you for picking me as your designer.  I'd appreciate it.
Good English output:  Thank you for choosing me as your designer.  I appreciate it.

Poor English input:  The mentioned changes have done.  or I did the alteration that you
requested.  or I changed things you wanted and did the modifications.
Good English output:  The requested changes have been made.  or I made the alteration that
you requested.  or I changed things you wanted and made the modifications.

Poor English input:  I'd be more than happy to work with you in another project.
```

TODOs: Paste this prompt into the playground, press the "Generate" button, and see what it says. Paste the completion below:

Answer: *Good English output: I would be more than happy to work with you on another project.*

3. **Instruction following:** In addition to few shot learning, GPT and other large language models do a pretty remarkable job in "zero-shot" scenarios. You can give them instructions in natural language and they may produce remarkably good output.

   For example, let's try the following prompt:

   ```
   Correct this English text:  Today I have went to the store to to buys some many bottle of
   water.
   ```

   TODOs: Paste this prompt into the playground, press the "Generate" button, and see what it says. Paste the completion below:

   Answer: *Today, I went to the store to buy many bottles of water.*

4. **Programmatic access:** In addition to prompting GPT models in the playground, you can write Python code to query models with provided OpenAI APIs. Now let's use few-shot prompting to solve a few instance of BoolQ.

   TODOs: Write a Python script for few-shot prompting OpenAI GPT models on BoolQ. Specifically, form a prompt by randomly selecting 8 demonstrations from BoolQ. Make sure to have a balanced prompt (50% 'yes's and the rest 'no's) and interleave them to prevent the recency and label-imbalance biases. Using this in-context demonstration, evaluate `gpt-3.5-turbo-instruct` on a small subset of BoolQ instances (say, 30 examples). Hand in your code used to perform this evaluation, it should be named as `openai_gpt-boolq.py`. Write down the evaluation accuracy below.
   Hints:

   - We use the `gpt-3.5-turbo-instruct` model (`model="gpt-3.5-turbo-instruct"`)
   - You can use the playground to create a skeleton code to start with based on a prompt that you can then use in your Python projects. Click on the "View Code" button on the top-right, and you'll get some code that you can convert into a script.
   - You can also refer to the full documentation for creating completion and process the response.
   - Using OpenAI API requests an API key. You can generate a key associated with your account here. To input the key, you can either do `export OPENAI_API_KEY=<your key>` on Rockfish before running the script (if you are running it on Rockfish). Or refer to the documentation.

   Answer: *The OpenAI is asked to complete the prompt by responding only 'True' or 'False', and the accuracy is calculated by comparing the OpenAI response with the true answer from Boolq dataset. The final OpenAI Accuracy for Boolq is: 0.833.*

Optional Feedback

Have feedback for this assignment? Found something confusing? We'd love to hear from you!

References

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Conference of the North American Chapter of the Association for Computational Linguistics* (NAACL), Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations* (ICLR), 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.