CS 601.471/671 NLP: Self-supervised Models

Homework 6: Training Transformer Language Models

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! :)

Note: You can do this homework as a team of 2 people. Basically, you can write a single solution for both people and upload it as a single PDF as a group.

How to hand in your written work: Via Gradescope as before.

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: *TBD*

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: *TBD*

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.

[*] https://self-supervised.cs.jhu.edu/sp2023/

Answer: *TBD*

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: *TBD*

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: *TBD*

2 Getting Your Attention with Self-Attention

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: *TBD*

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: *TBD*

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

[*]Question credit: John Hewitt

would result in the output $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_t$ being approximately equal to $\mathbf{v}_j$ for some $j \in [0 \dots n]$.

Hint: Show that there can exist $j \in [0 \dots n]$ 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 $\{\mathbf{k}_0 \dots \mathbf{k}_n\}$.
Answer: *TBD*

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[‡] $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m\}$, while $\mathbf{v}_b$ lies in a subspace B formed by the p basis vectors $\{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_p\}$. 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 $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m\}$, 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 $\mathbf{v}_a$, $\mathbf{v}_b$, $M\bar{\mathbf{v}} = v_a$. Show that $M\bar{\mathbf{v}} = \mathbf{v}_a$ holds for your M.
   Answer: *TBD*

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 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: *TBD*

3 Programming

Unlike the prior homework assignments, there is no Jupyter notebook anymore. Instead, here you will write your own code in your IDE.[§] **What are the deliverables in this homework?** For each problem subsection, we ask for a brief explanation or plot. Include those in your homework answers when you return them. Additionally, create a public Github repository to upload your code. In most subsections we will ask you to include a link to your code.

In this programming assignment you will get your hands dirtier with building self-supervise models. You will build a classifiers using Huggingface Transformer library and PyTorch. This classifier is expected to solve the BoolQ dataset [?]. 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.

Let's solve this problem

1. Throughout the homework, we will use "accuracy" (ratio of the instances on which our model produces the right label) as a measure of our models' performance. What is the random chance performance for this dataset? Answer: *TBD*

[†] https://en.wikipedia.org/wiki/Linear_subspace
[‡] https://en.wikipedia.org/wiki/Basis_(linear_algebra)
[§] Don't have an IDE? Try PyCharm: https://www.jetbrains.com/pycharm/.

2. You are given a starter code for this programming assignment. This starter code has a bunch of basic functionalities implemented, but it is missing details. Your job is to complete the missing pieces and extend beyond what is there. You should feel free to change this starter code in any way that you'd like in order to address the requested results below.

   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` [?].

   In particular, here we want to make sure that the implementation is correct by training on 10 training instances upon training for a few epochs (say, a total of 20 epochs). Note, 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.

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

   Then **plot of training accuracy as a function of training epochs.**
   Answer: *Include the figure here:*

   Link to Github Code

   URL-HERE

3. Now let's start using Rockfish GPUs. The first step is to go to `https://coldfront.rockfish.jhu.edu/`. Your JHED are already added to the system however, we have not given you a password. To set a password, click on "reset password" and enter your email (`JHEDid@jhu.edu`) and reset your password. If you have any difficulties here, please let us know ASAP! You certainly don't want to do this a day before the homework deadline as this might take some time.

   After this step, use your terminal to login to Rockfish's login node:

   ```
   > ssh JHEDid@login.rockfish.jhu.edu
   ```

   You will be prompted to enter your password and then you should be able to see your home directory on Rockfish login node. **Here include a screenshot of your terminal as soon as your login.**

   Answer: *Include the figure here:*

4. To submit jobs to Rockfish we will use Slurm job management system. Slurm is an open-source job management system used to manage and schedule jobs on high-performance computing (HPC) clusters. Slurm stands for "Simple Linux Utility for Resource Management" and is widely used in academic, research, and government institutions. Slurm allows users to submit, monitor, and manage jobs across multiple nodes in a cluster. You can learn about Slurm here or here.

   Let's try one Slurm command:

   ```
   > squeue
   ```

   This command should show your all the jobs that are in the queue to be executed or the ones that are already being executed. If this command runs successfully, **include a screenshot of it.** You can personalize it, if you type `squeue -t JHEDid`.

   Another useful command is `sinfo` which shows the list of compute nodes on Rockfish:

   ```
   > sinfo -o "%50N  %10c  %20m  %30G "
   ```

   If this command runs successfully, **include a screenshot of it.**
   Answer: *Include the figure here:*

5. Now let's repeat step 2 above on Rockfish. Move your code to the login node on Rockfish. Note, `launchpad.sh` includes the Slurm specifications on how to run your code on Rockfish. You can lunch your experiment on Rockfish, by simply running "`sbatch launchpad.sh`". You can track this job on in the queue via "`squeue -u JHEDid`". Similar to step 2, include a figure showing the accuracy of your model on the training data, as a function of epoch.

Link to Github Code

`URL-HERE`

6. Now, rather than running training on a subset of the data, try training all the data. 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. Train for 30 epochs and **plot the accuracy on both train and dev sets** as a function of epoch number. 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.
Answer: *The largest batch size: ?*
Output figure:

Link to Github Code

`URL-HERE`

7. Let's implement hyper-parameter selection. Basically, train models for various choices of learning rates (`1e-4, 5e-4, 1e-3`) and training epochs (`5, 7, 9`). Select the model with the highest accuracy on the dev set and report its accuracy on the test set.
Answer:
Best accuracy of distilBERT-base-uncased on the dev set: ?
Best accuracy of distilBERT-base-uncased on the test set: ?

Link to Github Code

`URL-HERE`

8. Now repeat the previous experiment with other models. In particular, create a bar plot that shows test/dev accuracies 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.
Answer: *Output figure:*

Link to Github Code

`URL-HERE`

9. **Extra Credit:** Repeat step 8 for T5 models of various size (small, base, medium, ...). Note, you will need to adjust the implementation of your classifier since T5's architecture is a bit different than the BERT-family.
Answer: *Output figure:*

Link to Github Code

`URL-HERE`

Optional Feedback

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