Homework 5

This is the fifth (out of five) assignment for DS-GA 1008 Deep Learning. Out: November 14, 2020. Due: December 4, 2020; 11:55 pm. Per the course syllabus, the solutions must be submitted on or before the deadline, and they must be typeset; handwritten answers will not be accepted. You will need to submit a zip folder which contains the following to NYU classes: (1) the typeset pdf containing solutions to problems 1 through 5 (note that you need to write up parts of the implementation questions in the pdf as well); (2) the two downloaded notebooks corresponding to problems 4 and 5. Please name your zip folder `lastname-firstname-netid-hw5`.

Problem 1. (33/100 points.)

Hopefully now, we are all better prepared to read deep learning related papers. BERT is one of the most popular (also somehow controversial) papers in recent years. The ideas are extremely simple (and the corresponding paper is easy to read), but the impact is huge. There is no doubt that a wide variety of industry products already integrated BERT, including Google search (https://blog.google/products/search/search-language-understanding-bert/).

Read the BERT paper (https://arxiv.org/pdf/1810.04805.pdf), and answer the following questions. The materials of Lab 11[1] will also help.

(a) What does BERT stand for (i.e., what does B, E, R, T stand for, respectively)?

(b) What is the dataset and what is the size of the dataset (by the number of words) which BERT is pretrained on?

(c) What is the backbone to BERT-large? Give the model, together with its number of layers, the number of self-attention heads, and the hidden size. No need to write down equations.

(d) Recall that we once used transformer to do language modeling. We needed position embeddings to help encode the input sentence. BERT also requires position embeddings. Explain in a few sentences: what are position embeddings and why do we need them?

(e) What are the two BERT pretraining objectives? Describe each objective in one or two sentences.

(f) Given a sentence $\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$ where each $y_t \in \mathcal{V}$, $t \in \{1, 2, \ldots, T\}$, is a word where $\mathcal{V}$ is the vocabulary, suppose we want to train a language model p_θ using the BERT-version of the masked language modeling (MLM) objective. Suppose that y_{t_1} and y_{t_2} are two masked words. Based on your understanding of MLM from the paper, write down the corresponding MLM

[1] Lab on November 16, 2020: transfer learning in natural language processing.

negative log-likelihood which you would use to train p_θ. *Hint: this approach is different from the Lab 4 left-to-right language modeling.*

(g) How does MLM compare to left-to-right language modeling, in terms of performance on downstream tasks? Answer this question using at least Table 5 and Figure 5.

(h) How long did the authors pretrain BERT-large, on what machine?

(i) Recall that Lab 10[2] proposed two ways of finetuning a pretrained model: (1) only finetune the classification head; (2) finetune the ResNet backbone as well as the classification head. Which way is BERT paper using (to finetune on GLUE tasks)? What are the benefits, compared to the other way?

(j) What is a `[CLS]` token? How would you use it for classification tasks?

(k) Suppose we trained an BERT model for two months non-stop on GPUs. But oops, we forgot to add in the `[CLS]` token. Now, we want to finetune on MNLI (which we discussed in Lab 3[3]). How would you leverage BERT without the `[CLS]` token? Explain how you would fine-tune on MNLI, in detail, using ~5 sentences. Feel free to use equations if you find them helpful. (Suppose that retraining or continuing training with `[CLS]` is not an option.)

(l) Explain the learning rate scheduler used for BERT pretraining. This scheduler is a common choice used by most transformer models.

Problem 2. (7/100 points.)

Recall that in the lectures as well as Lab 4[4], we discussed left-to-right autoregressive language modeling. Given a sentence $\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$ where each $y_t \in \mathcal{V}$, $t \in \{1, 2, \ldots, T\}$, is a word where $\mathcal{V}$ is the vocabulary, we have the autoregressive factorization $p_\theta(\boldsymbol{y}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{<t})$.

At training time, the model is optimized according to the maximum likelihood objective: $\arg\max_\theta \sum_{\mathbf{y} \in \mathcal{D}} \log p_\theta(\mathbf{y})$ where $\mathcal{D}$ is the training dataset (i.e., a set of sentences).

Now, we extend this language modeling concept to machine translation. We want to use an encoder-decoder model to do (supervised) machine translation from English to Burmese. Specifically, the model contains 6 layers of transformer encoder and 6 layers of transformer decoder. Given a source English sentence $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$ where each $x_t \in \mathcal{V}_{\text{English}}$ is a word in the English vocabulary, we have the autoregressive factorization $p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T'} p_\theta(y_t \mid \boldsymbol{y}_{<t}, \boldsymbol{x})$, where $\boldsymbol{y} = (y_1, y_2, \ldots, y_{T'})$ where each $y_t \in \mathcal{V}_{\text{Burmese}}$ is a word in the Burmese vocabulary. We use a loss similar to the loss we would use to train language models.

[2] Lab on November 9, 2020: transfer learning in computer vision.
[3] Lab on September 21, 2020: sequence classification using LSTM.
[4] Lab on September 28, 2020: language modeling using recurrent networks and transformer.

However, the issue is that we do not have lots of English-Burmese parallel sentence pairs. One way to address this issue is to use a pretrained English BERT to help the machine translation model.

How should we integrate BERT into the aforementioned machine translation model, using transfer learning? No need to be very detailed. But explain your design and intuition in a few sentences (for example, 5-10 sentences). Full credits as long as the answers are reasonable. Reviewing transformer materials[5] may help.

Problem 3. (15/100 points.) Transfer learning in NLP. As you have seen from the BERT paper, a very powerful strategy for transfer learning in NLP is to train a transformer-based language model on large amounts of text, and then fine-tune it on some downstream task of interest.

While this strategy has given us considerable gains on a variety of tasks, it still faces issues when fine-tuning on a downstream task for which there is limited data available. In this question, we will set up the background for Problems 4 and 5. We will explore how we can put together all the tools we have learnt so far to push performance on tasks even when we do not have a lot of data.

We will stick to natural language understanding (NLU) tasks and use the tasks from the GLUE benchmark as our running example. We would strongly encourage you to check out https://gluebenchmark.com/ and https://huggingface.co/nlp/viewer to find out more about the tasks and what the datasets look like.

There are several ways to leverage data from other tasks in a way that is beneficial to the task you are trying to solve. Intermediate task training (https://www.aclweb.org/anthology/2020.acl-main.467.pdf) and multi-task training (https://ruder.io/multi-task/index.html) are two such approaches.

The first approach consists of first fine-tuning your pre-trained model on an "intermediate" task that has valuable information which could be useful for the downstream task. Then, as usual, the model is fine-tuned on the downstream task.

Multi-task training is another approach whereby the model is fine-tuned jointly on multiple tasks. Both of these approaches have significant drawbacks which were discussed in the lab on transfer learning for NLP.

A third more recent option, has been to use adapters. "Adapter" refers to a set of newly introduced weights, typically within the layers of a transformer model. Adapters provide an alternative to fully fine-tuning the model for each downstream task, while maintaining performance. Only the weights inside the adapter are trained during fine-tuning while leaving the rest of the models weights frozen. To learn more, read this paper: https://arxiv.org/pdf/2007.07779.pdf.

[5] Materials that may help (Problem 2): Lecture 4, Lab 4, as well as the following guides: http://jalammar.github.io/illustrated-transformer and https://nlp.seas.harvard.edu/2018/04/03/attention.html.

(a) Write a few words about catastrophic forgetting and why it is generally a problem, and specifically when trying to fine-tune on multiple tasks sequentially.

(b) Write about some benefits of using multi-task learning.

(c) Write a few words about difficulties in multi-task training.

By using adapters, it is possible to combine multiple tasks without the problems that we discuss above in part (a) and (c). You will implement this in Problem 5.

Problem 4. (30/100 points.) Implementation: BERT fine-tuning. Please see `hw5_problem4_transfer_learning_classification.ipynb`. **Note:** While running this experiment you may observe high variance in the results that you obtain. Please run at least 3 times (with different seeds) and report the mean and standard deviation across your runs.

Important note: For this problem, answer the following questions in your writeup pdf (together with all previous problems), and complete the notebook instructions as well.

(a) Our target task for this problem is RTE. Use the dataset viewer https://huggingface.co/nlp/viewer, and answer the following question. How is the RTE task different from the MNLI task, in Lab 3? *Hint: if you can't find RTE in the data viewer, it might be in GLUE.*

(b) In the notebook, you will first fine-tune a BERT model from scratch on the RTE dataset. Note that in this case, the BERT model is not pretrained. You will need to read the Huggingface documentation to understand how the model is set up, what the model class returns and complete the SequenceClassificationBERT class. Your objective will be to extract the hidden state representation of the CLS token from the final layer and apply a pooling layer (this will be a linear layer followed by a tanh non-linearity). This representation will be used as input to a linear classifier to predict the correct class. In the pdf, report results in terms of best validation accuracy (its mean and standard deviation across runs). Please keep the logs in the notebook as well.

(c) Use the same class that you have written for part (a), but this time you will be loading the pre-trained weights of the BERT model. In the pdf, report results in terms of validation accuracy. Please keep the logs in the notebook as well.

(d) Compute and report the number of trainable parameters in this model which you uses to fine-tune on RTE.

(e) Write a few words about why training this model from scratch on the RTE dataset gives inferior results compared to fine-tuning the model after loading pre-trained weights. What is the limiting factor here? How is it overcome?

Problem 5. (15/100 points.) Implementation: transfer learning with adapters. While running this experiment you may observe high variance in the results that you obtain. Please run at least 3 times (with different seeds) and report the mean and standard deviation across your runs.

Important note: For this problem, answer the following questions in your writeup pdf (together with all previous problems), and complete the notebook instructions as well.

(a) (0 pt) In this question you will learn how to train adapters. The objective of this question is to get familiar with the concept of adapters, the library which provides you with easy-to-use functionality for training your own adapters and then using them for inference. You will learn how to train an adapter for RTE. See the `hw5_adapter_quickstart_training.ipynb` notebook. You don't need to submit the `hw5_adapter_quickstart_training.ipynb` notebook.

(b) (15 pt) See `hw5_problem5_adapter_fusion.ipynb`. As we discussed before, it is often beneficial to leverage multiple tasks in order to extract useful information that can give you better performance on your task of interest. In this question, you will learn how to load pre-trained adapters, combine them using AdapterFusion (https://arxiv.org/pdf/2005.00247.pdf) to obtain better performance on the RTE task. You can check out all available task adapters on https://adapterhub.ml/explore/ and try using some combination of them. Report results on using 3 other task adapters (other than RTE) and report which ones you used, and why.

*When submitting notebooks, you will need to submit the downloaded version of the notebooks (*not *a URL). Please read the first paragraph of this assignment for submission instructions. If you need help, don't forget about the six weekly office hours. Best of luck with the assignment, and thanks for your hard work!*