EECS 595: Homework 3 – Transformers and BERT

Due: See Canvas

1 Introduction

Modern NLP is frequently based on large language models that have been pretrained on massive amounts of text to learn and internalize world knowledge and utilize it to perform various downstream tasks. Underlying most of these models is the transformer neural network which consists of self-attention and a feed-forward network. In Homework 3, you'll see how these work in practice by implementing the BERT Devlin et al. [2018] architecture and pretraining it on a small amount of data. This homework helps you understand how these types of models learn and what are the computational bottlenecks and challenges in training.

Much like Homework 2, this homework has three conceptual parts. The first will have you build a Transformer and a BERT model and pretraining them for "language modeling". This is the most time-consuming part of development and training (similar to building word2vec). The second part helps you explore the pretrained model and visualize its internals. Finally, in the third part, you will fine-tune the BERT model for classification. To maintain conceptual simplicity, we will utilize a significant portion of the data and retain the same sentiment analysis task from the previous assignment, Homework 2.

This homework has the following learning goals:

- Develop your PyTorch programming skills by working more with the library
- Learn how to use the `tokenizers` package to turn text into sequences of tokens
- Learn how multi-headed attention works
- Learn how to build neural networks using other networks as components
- Learn how to pretrain vs. fine-tune neural language models
- Learn how to do model selection and evaluation during training

2 Notes

This homework requires that you use a GPU for parts of the pretraining. You can use your own or you can use the GPUs on Great Lakes for free. We have provided you with a course account for Great Lakes, `eecs595w24`, which will give you access for many hours. Your Great Lakes jobs are limited to 4 hours of training, which is more than enough to train your BERT model for this assignment.

Here is a guide on how to access Great Lakes, set up your environment, and submit your jobs: Great Lakes Tutorial. We strongly encourage learning to use Great Lakes for this assignment as (1) you have free access to substantial computational resources and (2) you would want to use it for the next homework and the final project.

Finally, this is the first release of the transformer language model homework. While we have gone to great lengths to test and document all the pieces in the homework, some parts may be unclear. Your feedback is greatly appreciated and specific feedback is rewarded in Extra Credit (see Section 7) so that we can improve this assignment for future classes. Given the importance of large language models, we think this type of assignment is essential for helping NLP students understand how these models are built and how to use them so we want to make sure it's approachable!

3 Data

For data, we use the same data previously used in Homework 2. It is a sample of cleaned Amazon book reviews that have been downsized for efficient processing.

If you are very ambitious, we have included one huge file you can use to train your model. Feel free to see how the model works and whether you can get through a single epoch! We have provided several files to use in both the transformer/BERT part and the downstream classification part:

1. reviews-word2vec.med.txt – This is the debug data for testing.
2. reviews-word2vec.large.txt – You can train on this on your laptop to get a semi-decent model.
3. reviews-word2vec.large.txt.gz – Use this file for training on Great Lakes to get your final pretrained model. *This is a new file.*
4. reviews-word2vec.huge.txt.gz – If you're willing to train for a while, use this dataset.
5. sentiment.train.csv – This is the training data for your fine-tuning of BERT. You do not need this data for pretraining (nor should you use it).
6. sentiment.dev.csv – Labeled data for evaluating the fine-tuned classifier in Part 5 and performing model selection.
7. sentiment.test.csv – Unlabeled test data for evaluating the fine-tuned BERT classifier in Part 5. You will upload your predictions for this test to Kaggle.

Note that most of these files are the same so they can be downloaded from the Homework 2 folder on Canvas again or re-used if you already have them. Only the "larger" file is new and is included in the directory. Also, note that the files still say word2vec. We've left this name the same so that you don't have to have two of these files sitting around on your disk—but you're still using these for BERT (not word2vec).

4 Parts 1-3: Implement and pretrain BERT

The majority of your effort will be spent in Parts 1-3 which involve writing most of the code. We have put more detailed notes on how to do the implementations in the notebook, rather than in the PDF. There are many guides for how to think about how each of the components works (e.g., this one you saw in class). You are welcome to look them up to help build intuition. Lectures 13 and 14 also provide some guidance on the algorithms and network structure.

4.1 Part 1: Tokenizing

To start working with textual content consisting of sequences of words, we need to decide what those sequences are made of. We have seen two extremes so far: character-based language models and token-based language models. Character-based models have very small vocabularies but very long sequences. Token-based models have much larger vocabularies (100K+) but shorter sequences. Neural language models use a middle ground. Typically, the text is tokenized into shorter sequences that usually occur in the data. These might be whole words at times but also be prefixes or suffixes. Crucially, these are learned from patterns in the data.

For this homework, we will use the existing BERT tokenizer from the `tokenizers` library. This means the tokenization strategy has already been chosen for you and you don't need to learn anything. We also have provided the vocabulary to the tokenizer on Canvas. You need to use this vocabulary in this assignment.

■ **Problem 1.** (5 points; code and PDF) Initialized a BERT workpiece tokenizer with the provided vocabulary. Tokenize 5 sentences of your choosing and report these in the PDF as well as what you see in terms of how the text is broken up into subwords. (You are encouraged to try diverse sentences.) What advantages or disadvantages do you think this tokenization strategy has compared with character-based or whole-word-based approaches to tokenization?

4.2 Part 2: Transformer and BERT Implementation

Part 2 has you building the Transformer network and then using that network to build a BERT model. Homework 2 had you building separate neural networks for each task. In Homework 3, we use a slightly different approach where we define multiple types of neural networks for different parts of the BERT/Transformer model and then combine them. This allows us to compose the whole network from a series of smaller, more conceptually simple networks. There are seven of these networks described in the starter code and we've provided some examples so you can build simple pieces and verify each piece runs as you go.

■ **Problem 2.** (30 points; code) Complete Parts 2.1 to 2.7 in the code. These will fully implement the Transformer (2.1 to 2.4) and BERT (2.5 to 2.7).

4.3 Part 3: Pretraining

Part 3 will have you pretrain the BERT model on book review text. You will have seen some parts of this code before from writing the Dataset and core training loops in Homeworks 1 and 2. However, new in Homework 3 is having to deal with creating batches. You saw a bit of this in Homework 2 where for word2vec training, we had to ensure that all the training examples had the same total number of context words, and add a few extra negative examples for target words on the edge. Here, we'll need to deal with getting our token sequences to be the same length in a batch.

■ **Problem 3.** (10 points; code) Generate the Dataset, collator function, and core training loop for masked language modeling

Once you have your basic model implemented, it's time to start pretraining. We recommend starting with the *large* dataset. This should be enough data that one or two epochs will pretrain the model enough to test that it's working in the exploratory pieces of Part 4. If you're feeling ambitious or your CPU is slower than you'd like, feel free to do some of this initial training on Great Lakes (skip to below). For getting a sense of whether the model is working, we strongly recommend testing the masked language modeling piece in Task 4.4 on your cpu-trained model to see whether it can correctly fill in common words.

In your implementation, we recommend starting with these default parameter values:

- batch size = 8
- loss = cross-entropy
- hidden layer size = 256
- $\eta = 4e - 5$ (learning rate)
- epochs $= 5$
- optimizer = AdamW

You are welcome to try changing these parameters on your machine.

Once you have an implementation that you're satisfied with, it's time to pretrain it on more data and more epochs. Doing this on a CPU is going to take too long, so you'll need to use a GPU. We've provided access to Great Lakes (see Section 2 Notes). In Part 3.5, you'll convert your notebook to a script and submit it to Great Lakes.

Important Change from CPU training: When running the final pretraining, you will need to change the code in the script as follows:

1. Train using the ”larger” file (not ”large”)
2. Train for at least 5 epochs. Your code should be set up to save the model after every epoch, so feel free to keep the job running the full 4 hours to see how many epochs you can make it through
3. Set the batch size to 128
4. Use the spgpu partition. This has the ”fast” GPUs that should be ideal for training

With these changes, you should still be able to monitor progress on Weights & Biases while your model is pretraining to verify that it's working as expected.

■ **Problem 4.** (10 points; code) pretrain model on larger data using a GPU (on Great Lakes) and save your model every epoch.

If you are *really* unsure it's going to work or you want to try out some alternative configurations, we recommend requesting an interactive notebook and verifying that the training works (e.g., that the GPU doesn't run out of memory, or that you have all the necessary packages installed). However, we don't recommend doing the actual pretraining in interactive mode—in part, because you won't know when you'll actually get access to the notebook and that might come at a very inconvenient time (e.g., Saturday at 3am). If needed, please just use the interactive mode to do some verification that your hyperparameters work and the model trains, then convert the notebook to a script and submit that directly.

5 Part 4: Qualitative Evaluation of Attention and MLM

Once you've finished getting a pretrained model, let's see what it can do. You are also encouraged to work with the Part 4 code to explore your CPU-trained models and see whether the code seems like it's working.

Part 4 has you do a few exploratory analyses of your BERT model. There are no wrong answers here, per se, but, ideally, your pretrained model should have learned enough that the results of the exploration look meaningful, rather than random.

In 4.2, we have provided code for looking at the nearest neighbors for words using the BERT model's initial non-contextual embeddings. These embeddings aren't trained the same as word2vec, but knowing what you saw in Homework 2, some of the neighbors should look thematically or semantically similar. Also, still keep in mind that the training data is book product reviews, so choose words that are likely to be used with higher frequency in that genre.

■ **Problem 5.** (5 points; PDF) Evaluate the nearest neighbors. Choose 5 words and report the nearest neighbors. At least one of your words should have some reasonable neighbors, while at least two should not look reasonable. In a few sentences describe what you see and why you think you see the behavior you do.

In 4.3, we have provided code that lets you explicitly mask some tokens and see what the model fills in. This is the task the model is trained to do, so for preliminary analyses, we encourage you to test with masked language modeling, e.g., can the model correctly fill in high frequency words (e.g., "the") when masked? We strongly encourage you to play around with this function to get a sense of what the model has learned

■ **Problem 6.** (5 points; PDF) Evaluate masked language modeling. Write at least six sentences with at least one [Mask] token in each and write a variety of sentences in terms of content, number of masks, and which kinds of words are masked (and show what the model produces). In a few sentences describe what you see in terms of how well the model is performing at filling in the blanks (are the words reasonable) and what kind of systematic behavior you see. You're welcome to include more than six sentences if it helps show your point.

For the next problem, you'll need to extend the code we gave you to generate the most probable words for a mask (the code we gave just fills in one word).

■ **Problem 7.** (10 points; code and PDF) Examine the most probable words for a variety of sentences. Implement the get_word_probabilities function for a masked word. Choose at least six sentences with one masked token (some of these could be from the previous problem) and report the 10 most probable words for each. In a few sentences, describe what you see in terms of how the model is behaving. How many of the probable tokens are good substitutes? Do some contexts have only a few good substitutes while others have many or none?

In Part 4.4, you now will look at what BERT is actually looking at by assessing its attention heads. We've provided code to generate plots showing the strength of attention. Your model is still relatively small, so the attention may not look like much but with some contexts, you can start to see the model attend to specific words. We strongly recommend trying to examine attention when context matters: e.g., for compound phrases, multiword expressions, names, masked tokens, and pronouns.

■ **Problem 8.** (5 points; PDF) Evaluate the attention heads. Write 5 sentences that show a variety of different types of text. At least one of these sentences should have a masked token. Examine the two layers' attention heads for each and include these in the PDF. In a few sentences describe what you see in the model's attention behavior. For example, are the heads looking at similar or different things? Does the attention for some types of words follow systematic behavior? Are the layers learning different things?

6 Part 5: Fine-tune your BERT model

Part 5 will have you fine-tuning your BERT model for a specific task: sentiment analysis. This is the same task as in Homework 2, except this time we're using a more sophisticated architecture. Part 5 can be done entirely on a CPU and we have included some benchmark times for the reference implementation.

The basic steps for implementation are (1) defining a Dataset for the sentiment data, (2) writing a data collator function for creating batches and (3) writing the core training loop. Along the way, we'll use Weights & Biases to do more advanced reporting of the model performance. When training large models, it's common to keep training but do "model selection" where you use performance on the development data to choose which version of a model (i.e., parameters) to keep as the best model. You will periodically evaluate every 1000 steps and add these to your Weights & Biases logs. This will show not only the loss, but also model performance on the prediction task.

In your implementation, we recommend starting with these default parameter values:

- batch size = 8
- loss = cross-entropy
- $\eta = 2e - 5$ (learning rate)
- epochs $= 2$ (we do recommend having this run for more though if you can!)

- optimizer = AdamW

You are welcome to try changing these parameters on your machine. In particular, we encourage you to try running for more epochs to see when the model will start to overfit on the data.

■ **Problem 9.** (5 points; code) Generate the Dataset, collator function, and core training loop for sentiment analysis. Implement the evaluation function and report evaluation metrics to Weights & Biases

Once the model is done training, let's look at the Weights & Biases plots to see what the model learned. In particular, we'll want to see how loss relates to performance. Deep learning models can overfit when enough training, so these plots are very helpful for deciding when to stop training and how to choose a model.

■ **Problem 10.** (5 points; PDF) Examine the relationship between loss and model performance (all of your metrics). In a few sentences, describe what you see. Are the results correlated? Describe how you pick your "best" model based on what you see in the plot.

Once you have decided on your best model, implement the evaluation code and generate predictions for the test set. We'll use Kaggle again so you can get a sense of how well the model performs. Since this is the same data as with word2vec, you can also compare how well your attention classifier did compared with your new BERT model! Submissions close to the average performance or above will get full credit.

■ **Problem 11.** (10 points; PDF) Predict on the test data and submit your predictions to Kaggle. Be sure to report your username in the PDF.

7 Optional Extra Credit Part 6: Homework Feedback (up to 10 points)

Homework 3 is experimental and designed to demystify how transformer models are trained. However, it is a big homework with many moving pieces. We would greatly appreciate your feedback to help improve it. For 2 points each, please submit some of the following types of feedback:

1. Detailed descriptions of where you got stuck in the assignment, e.g., what you were confused about (code, concepts, etc.). This will help us figure out which pieces need more details. You can submit multiple examples of this.
2. Suggestions for where the assignment could be extended, e.g., what more would you like to learn. You can submit multiple examples of this.
3. Any pain points on what made Great Lakes hard to use (with the exception of waiting in the queue). We're looking for specific details on gaps in their documentation or confusion around how to submit jobs.

4. Suggested edits to the instructions or code TODOs to improve clarity. These should be substantial suggestions (not typos) and of the form that would help a future student do the assignment more easily.

If you would like to provide some other type of feedback for credit, please reach out to us first via a Piazza post and we'll consider it.

8 Submission

Please upload the following to Canvas by the deadline:

1. Your Jupyter notebooks and python script code for all the implementations (as code)
2. A separate PDF document that contains these answers, labeled by problem number. Do not submit a PDF copy of your Jupyter notebook, as it is slower to grade and receives a penalty. Be sure to include your username on Kaggle. All written/plotting problems should be numbered so we can search for them during grading. We reserve the right to score answers as zero if we can't easily find them during grading.

We reserve the right to run any code you submit; **code that does not run or produces substantially different outputs will receive a zero**.

9 Academic Honesty

Unless otherwise specified in an assignment all submitted work must be your own, original work. Any excerpts, statements, or phrases from the work of others must be identified as a quotation, and a proper citation provided. Any violation of the University's policies on Academic and Professional Integrity may result in serious penalties, which might range from failing an assignment to failing a course, to being expelled from the program. Violations of academic and professional integrity will be reported to Student Affairs. Consequences impacting assignments or course grades are determined by the faculty instructor; additional sanctions may be imposed.

Copying code from any existing BERT or attention implementation is considered grounds for violation of Academic Integrity and will receive a zero. Code generated through automated or AI-assisted tools, such as Co-Pilot will also receive a zero.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.