AMS 691.02: Natural Language Processing – Fall 2024
Assignment 3 (90 Points)

Instructor: Jiawei (Joe) Zhou
Assigned: Friday, November 8, 2024
Due: 11:59 PM EST, Sunday, November 24, 2024

Submission Instructions

Submit your report (in pdf format) and code through Brightspace. You should submit exactly two files for the assignment for report and code (if there are multiple files for code, compress them into a single file). Your report should contain all of the required results and analysis. Your code should be well documented and directly executable.

Lateness Policy

If you turn in an assignment late, a penalty will be assessed. The penalty will be 2% (of the entire point total) per hour late. You will have 5 late days to use as you wish during the semester. Late days must be used in whole increments (i.e., if you turn in an assignment 6 hours late and want to use a late day to avoid penalty, it will cost an entire late day to do so). If you wish to use a late day for an assignment, indicate that in the comments along with your homework submission through Brightspace.

Collaboration Policy

You are welcome to discuss assignments with others in the course, but solutions and code must be written individually.

Provided Data

- `dbpedia_{train/dev/test}.jsonl`: the DBpedia classification dataset. Each line is a JSON object with the following fields:
  - `sentence`: the sentence to be classified.
  - `label`: an integer ranging from 0 to 13 denoting the label of the sentence.

In addition to data files, `hw3-code-skeleton.ipynb` is provided as a template for your code submission so that you don't need to worry about setting up the environment and the data processing issues. Feel free to use the provided code or not.

Implementation Tips

Problem 1 of this assignment aims to help you get familiar with GPU training with mini-batch, PyTorch and HuggingFace Transformers. I implemented everything on the Google Colab platform. You will need GPUs to speed up the training process for Problem 1

- To use GPU runtime on Colab, Select "Runtime" → "Change runtime type" → "T4 GPU".
- GPU hours on Colab are limited for each user. Make sure the code is runnable on CPUs before switching to GPUs.

1 Pretrained Language Models (60 points)

In this assignment, you will explore pre-trained language models, as well as finetuning them for downstream tasks. You will use the HuggingFace Transformers library, which provides a unified interface for many pre-trained language models. We will focus on the masked language model BERT (Devlin, 2018) first, and switch to the autoregressive language model GPT-2 (Radford et al., 2019) for extra credit.

1.1 Classification with BERT `[CLS]` Features (15 points)

Implement a ReLU-activated 2-layer perceptron classifier, with hidden-layer size 32. Your classifier should take the 768-dimensional `[CLS]` token (i.e., the first token of each sentence) representation as input and output a probability distribution over the 14 classes. Use the model `bert-base-cased` (see details in the given code skeleton). Optimize the cross-entropy loss using the Adam optimizer with a learning rate of 0.0005 (5e-4), and use a mini-batch size of 32. Train the model for 1 epoch (i.e., 1 pass over the training set).

You should only optimize the parameters of the classifier, and keep the BERT parameters fixed. The vectors extracted from BERT is also called *frozen BERT features*. If you use PyTorch and HuggingFace Transformers, you can achieve this by using `torch.no_grad()` and specifying the parameters in the declaration of the optimizer.

Run the above experiments 5 times with different random seeds, and **report the mean accuracy and the standard deviation on the development (dev) set, as well as the accuracy on the test set for the best-performing (in terms of development set accuracy) model**. It's preferable to specify random seeds (on both CPU and CUDA; see how to set random seeds in PyTorch here) to make the results reproducible; however, if you don't specify that, each run will be different due to the randomness in the initialization of the model parameters. Feel free to reuse some of your code from Assignment 2.

For sanity check, one of my runs gives an accuracy of 96.4.

1.2 Classification with Mean-Pooling and Max-Pooling (15 points)

BERT, among many other models, encodes a sentence to a sequence of fixed-dimensional token representations, where the sequence length is the number of tokens in the sentence. Let L denote the number of tokens in the sequence, and D denote the hidden-layer size of a BERT model (768 for `bert-base-cased`), the pooling technique converts the $L \times D$-shaped tensor to a D-dimenional vector by taking the max, average or more complex operations (e.g., attention-based pooling). Concretely, let $\mathbf{E} \in \mathbb{R}^{L \times D}$ denotes the representations of a sentence output by BERT (or any other sentence representation models), the ith dimension of the mean-pooling result $\mathbf{e}$ is calculated by

$$\mathbf{e}_i = \frac{1}{L}\sum_{\ell=1}^{L} \mathbf{E}_{\ell,i},$$

and the ith dimension of the max-pooling result $\mathbf{e}$ is calculated by

$$\mathbf{e}_i = \max_{\ell=1}^{L} \mathbf{E}_{\ell,i}.$$

Implement a classifier that takes the mean-pooled and max-pooled BERT representations of **all content tokens** as input,[1] and keep everything else the same as in 1.1. Any non-padding token is considered a content token, including the `[CLS]` (can be viewed as the start of sentence for BERT models) and `[SEP]` (can be viewed as the end of sentence for BERT models) tokens.

[1]Since we are using mini-batch for training, we need to pad the sentences in a mini-batch to the same length to take advantage of the fast GPU tensor computation, and such padding tokens should not be considered as content tokens.

Train the model for 1 epoch, run the same experiment for 5 times, and report your results in terms of accuracy and standard deviation.

Hints:

1. You probably wish to check out the `attention_mask` key of the tokenizer output, which provides information about which tokens are content tokens and which are not.
2. Use tensor operations as much as possible to speed up the computation. For sanity check, my code finishes within 1 minute on a Colab T4 GPU.

1.3 Comparison of Pooling Techniques (10 points)

Based on the above results, can you figure out which pooling mechanism (first-token, mean pooling, max pooling) is the best with frozen BERT feature?

1.4 Fine-tuning BERT with `[CLS]` Features (20 points)

In the above experiments, we only optimize the parameters of the classifier, and keep the BERT parameters fixed. Now you are allowed to modify the BERT parameters too. Use the same hyperparameters (e.g., hidden-layer size of classifiers, learning rate, and number of epochs etc.), you are now allowed to modify the last two layer (indexed by layers 10 and 11 in the model implementation). Report your result in terms of accuracy and standard deviation, and compare it with the results in 1.1 and 1.2.

Hints:

1. You probably wish to check out the `named_parameters()` method of the BERT model to get the parameters of the last two layers.
2. To pass whichever parameters to the optimizer, you can store them into a list and pass the list to the optimizer as the first argument.

1.5 Extra Credit: GPT-2 (10 points)

GPT-2 is an autoregressive language model, which means it predicts the next token given the previous tokens; however, we can also use it as a feature extractor for classification tasks. Implement GPT-2 as a feature extractor for the DBpedia classification task, and report the accuracy and standard deviation on the development set, as well as the accuracy on the test set for the best-performing (in terms of development set accuracy) model.

2 Final Exam Exercises (30 points)

Below are some example problems for the final exam. You can expect the final exam to be similar in format to the following problems. The final exam could cover any of the topics in the course.

2.1 Softmax (15 points)

The `softmax` operation converts a set of *logits* $s_i (i = 1, \ldots, n)$ to a well-formed probability distribution p, where

$$p_i = \frac{\exp(s_i)}{\sum_{j=1}^{n} \exp(s_j)}.$$

a. (2 points) Suppose $n = 3$ and $s_k = 2 \log k (k = 1, 2, 3)$. Calculate the value of p_k for each $k = 1, 2, 3$.
b. (4 points) Incremental softmax: suppose we now have the values of p_{n-1}, s_{n-1} and s_n, what is the value of p_n? In other words, derive p_n in terms of p_{n-1}, s_{n-1} and s_n.

c. (4 points) Temperature: in practice, we often use a *temperature* parameter $\tau (\tau > 0)$ to control the "sharpness" or "flatness" of the distribution. The adjusted softmax is defined as

$$p_i = \frac{\exp(s_i/\tau)}{\sum_{j=1}^{n} \exp(s_j/\tau)}.$$

$\tau = 1$ gives the original softmax. What will happen when $\tau \to 0$? Will the distribution become sharper or flatter compared to the one given by the original softmax? Please explain your answer.
d. (5 points) Backpropagation through softmax: show that $\frac{\partial p_i}{\partial s_i} = p_i(1 - p_i)$.

2.2 Perplexity (15 points)

The perplexity of a language model, as an evaluation metric, on some held-out data (i.e., a set of sentences) $\mathbf{X}$ is defined as

$$ppl = 2^{-\ell},$$

where $\ell = \frac{1}{\sum_{\mathbf{x} \in \mathbf{X}} |\mathbf{x}|} \sum_{\mathbf{x} \in \mathbf{X}} \log_2 p(\mathbf{x})$ is the average log probability of the tokens in $\mathbf{X}$. Here, $\mathbf{x}$ denotes a sentence, and $|\mathbf{x}|$ denotes the number of tokens in $\mathbf{x}$.
a. (9 points) Suppose there are two sentences in $\mathbf{X}$, each with 4 tokens and receives the probability of 0.25. What is the value of ℓ and the perplexity?
b. (6 points) Suppose we have a language model that predicts the next token by drawing from a uniform distribution, without looking at the previous tokens. What is the perplexity of this language model on $\mathbf{X}$? In addition to the variables mentioned above, these variables might be useful:

- V: the vocabulary used by the language model
- $|V|$: the size of the vocabulary

References

Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. [2]

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. [2]