Ling 282/482 hw7

Due 11pm on November 13, 2024

In this assignment, you will

- Develop understanding of transformers, especially as encoders
- Explain how pre-training and fine-tuning work
- Implement a sentiment analysis model leveraging a pre-trained transformer model

Submission Instructions

This assignment contains both written and programming portions. The answers to written questions must be submitted in a *.txt or *.pdf file to Blackboard. You will receive an invitation link to complete the programming portion via Github Classroom. This will open a Github repository with starter code and missing portions for you to complete. When you are finished with implementation, simply commit and **push** your changes to the online repository that was created for you. Unless you request otherwise, I will grade your work **as of the most recent commit** in your repository, subject to any applicable late penalties.

1 Transformers and Pre-training [28 pts]

Q1: Parallelizability [10 pts] One major cause for the development and subsequent adoption of transformers is that they are very parallelizable.

- In your own words, why are recurrent neural networks hard/impossible to make parallel? [2-3 sentences, 4 pts]
- How does the transformer architecture overcome this limitation and thus enable parallelizability? [2-3 sentences, 4 pts]
- How is information about sequential order represented in the transformer? [2 pts]

Q2: Parameters [4 pts] Let d_e be the embedding/model dimension (i.e. d_{mdoel}) of a transformer model. (These can in principle be different, but are in practice made to be the same.)

- In a self-attention layer with a single head of attention, how many parameters are there? (Note: you can ignore W^O for this.) [2 pts]
- In the position-wise feed-forward network in one block, how many parameters are there? You may write d_f for the size of the single hidden layer of this sub-network. [2 pts]

Q3: Pre-training [14 pts] Transformers have also helped jump-start the pre-training + fine-tuning approach to NLP tasks.

- Provide at least two reasons as to why variants of language modeling are good pre-training tasks. [4 pts]
- What are two differences between BERT (and its variants) and GPT (and its variants)? [4 pts]
- Describe one method (e.g. diagnostic classifiers, adversarial data, ...) for analyzing a pre-trained model and one example result from that method. What do we learn from that result? [4 pts]
- In your own words, describe one risk in the current approach to pre-training ever-larger language models on ever-larger datasets. [2 pts]

2 Implementing a Transformer-based Sentiment Classifier [10 pts]

In the coding portion of this assignment, you will implement a model for sentiment analysis on the SST dataset, using a pretrained transformer as a text encoder. In particular, the following paper trains and makes available several "mini-BERTs", i.e. transformer encoders trained on masked language modeling, but of significantly smaller sizes than BERT: Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. By default, we will use their smallest model, which has 2 layers and a hidden dimension of 128.

Q1: Implement PretrainedClassifier In `model.py`, you will find the skeleton of a classifier that uses the representation of the special token '[CLS]' from a pretrained model (`self.encoder`) to make classification decisions. You must:

- In `__init__`, initialize `self.output` to be a linear layer of the right shape. The comment there provides more information. [2 pts]
- Implement `.forward`, which: extracts the top-layer '[CLS]' representation and then passes that through a linear layer to produce logits over the sentiment classes. Please read the comments (and the docs referenced therein) closely. [8 pts]

3 Running the Classifier [8 pts]

`run.py` contains a basic training loop for SST classification, using the final layer's representation of '[CLS]' of a pre-trained transformer encoder. It will record the training and dev loss at each epoch, and save the best model according to dev loss. At the end, it samples 10 random dev data points and prints the review, the gold label, and the model's prediction.

Q1: Default parameters Execute `run.py` with its default arguments. Please report: the best dev loss, the epoch at which the best dev loss was achieved, and the best model's dev accuracy. Moreover, please include: the 10 random dev examples, with gold labels and model predictions here. In 2-3 sentences, describe what you see and observe any trends in what the model gets right and what (and/or how) it gets things wrong. [4 pts]

Q3: Comparison to RNN Classifier In 2-3 sentences, please explain what differences you see in the classification decisions by this model using a pretrained transformer and the RNN classifier that you trained in HW6. What do you think may be causing these effects (or lack thereof)? [4 pts]

4 Testing your code

In the dropbox folder for this assignment, you will find a file `test_hw7.py` with a few very simple unit tests for the methods that you need to implement. You can verify that your code passes the tests by running `pytest` from your code's directory, with the course's conda environment activated.