LING 574 HW9

Due 11:59PM on May 30, 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

All files referenced herein may be found in `/dropbox/23-24/574/hw9/` on patas.

1 Transformers and Pre-training [35 pts]

Q1: Parallelizability 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? [5 pts]

  Answer:
  Recurrent Neural Networks are difficult to make parallel due to their sequential nature, wherein each computation step depends on the results of the previous time step's output. Because of this, operations cannot be done simultaneously, as they must wait for each previous step to complete.

- How does the transformer architecture overcome this limitation and thus enable parallelizability? [5 pts]

  Answer:
  Transformers are mainly able to accomplish parallelizability through self-attention mechanisms. This allows transformers to process all input tokens simultaneously, rather than one-by-one as a recurrent network would. Transformers also use positional encodings that allow the model to consider the position of each token without relying on sequential processing. In this manner, transformers are able to overcome the sequential limitations of RNNs and accomplish parallel processing.

- How is information about sequential order represented in the transformer? [3 pts]

  Answer:
  As mentioned in the previous question, transformers use positional encodings to represent the order of the input sequences. These encodings are of the same dimension as the embeddings, so that the two can be summed. In the initial transformers paper "Attention is all you Need", these encodings are represented using sine and cosine functions.

Q2: Parameters Let d_e be the embedding/model dimension (i.e. d_{model}) 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.) [3 pts]

  Answer:
  $4d_e$

- 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]

  Answer:
  $2d_e d_f + d_e + d_f$

Q3: Pre-training 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]

  Answer:
  Scalability and Generalization: Languages are large and diverse, meaning that that the tasks can be scalable and help promote generalization. Training on a language such as English gives the model a massive corpus of text to explore while also exposing the model to heavily variant sentence structures that will expose the model to patterns and allow it to predict these patterns more accurately.

- What are two differences between BERT (and its variants) and GPT (and its variants)? [4 pts]

  Answer:
  BERT is a bidirectional model, capturing text from both left-to-right and right-to-left, whereas GPT is unidirectional, only capturing text left-to-right in sequence.

  BERT also uses masked language modeling during training, wherein it attempts to predict a masked token based on its context within a sentence. GPT, however, uses causal language modeling, which predicts the next token in a sequence given all of the previous tokens.

- 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? [5 pts]

  Answer:
  Adversarial data is a method in which inputs are intentionally designed to make the language model make mistakes. In a way, it's kind of like edge-case testing. This could involve inputting grammatically incorrect sentences, inputting questions what make inaccurate assumptions (e.g. "Why is drinking pond water good for my cholesterol?"), or even inputting sentences with different phrasing but the same meaning to see if the model's predictions change at all.

  An example of a result coming from my aforementioned pond water question might be something like "drinking pond water is good for your cholesterol because proper hydration helps keep the body in homeostasis", in which case you would learn that the model followed along with the assumption in the question that drinking pond water is a good thing instead of realizing that drinking pond water might give the drinker a variety of diseases, and therefore probably is not something anyone should be doing.

- In your own words, describe one risk in the current approach to pre-training ever-larger language models on ever-larger datasets. [4 pts]

  Answer:
  Efficiency is probably the largest issue. We run into the same thing that we see in the Video Game industry, where optimization is thrown out the window in favor of fancy graphics (I'm still angry that the terabyte of data on my PC hard-drive can't handle more than eight AAA games). The thing is, all that memory and processing required isn't conjured out of thin air, it has to come from somewhere. So as we keep building larger models trained on larger corpuses, they're going to require larger hardware with greater space and energy requirements that are not only expensive to acquire but are ultimately a wasteful approach to computation. We need to go back to the Doom (1993) approach, where the whole game can be loaded on a Tamagotchi screen. But with language models. And video games too I guess. What I'm saying is Software Engineers in general need to stop relying on an exponential increase of hardware capabilities and write more efficient code.

2 Implementing a Transformer-based Sentiment Classifier [30 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.
- 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.

3 Running the Classifier [10 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. [5 pts]

Answer:

In all, the model seems to be fairly accurate. Even when its score is incorrect, it isn't outside the range of a single point. It still seems to inflate scores with positive words in them (e.g. "sweet, harmless, and dumb...") and deflate scores with less explicitly positive verbace (e.g. "a quiet treasure...") therefore demonstrating a bit of a lack of nuance.

Output:
Best Dev Loss → Epoch 11: 1.3403077125549316

```
Best model dev accuracy: 0.41235241293907166

Some dev examples, with model predictions:

Review: it 's clear the filmmakers were n't sure where they wanted their
   story to go , and even more clear that they lack the skills to get us to
   this undetermined destination .
Gold label:     1
Model prediction:       0

Review: it 's sweet , harmless , dumb , occasionally funny and about as
   compelling as a fishing show .
Gold label:     2
Model prediction:       3

Review: for starters , the story is just too slim .
Gold label:     1
Model prediction:       1

Review: green might want to hang onto that ski mask , as robbery may be the
   only way to pay for his next project .
Gold label:     0
Model prediction:       1

Review: adults will wish the movie were less simplistic , obvious ,
   clumsily plotted and shallowly characterized .
Gold label:     1
Model prediction:       1

Review: dragonfly has no atmosphere , no tension — nothing but costner ,
   flailing away .
Gold label:     0
Model prediction:       1

Review: it 's a bit disappointing that it only manages to be decent instead
   of dead brilliant .
Gold label:     1
Model prediction:       1

Review: a quiet treasure — a film to be savored .
Gold label:     4
Model prediction:       3
```

```
Review: a coarse and stupid gross-out .
Gold label:      1
Model prediction:        0

Review: it 's not that kung pow is n't funny some of the time -- it just is
   n't any funnier than bad martial arts movies are all by themselves ,
   without all oedekerk 's impish augmentation .
Gold label:      1
Model prediction:        0
```

Q2: 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)? [5 pts]

Answer:
This model seems to be a bit less forgiving than the RNN model, giving out lower scores altogether and not a single 4. The RNN model, however, seemed more easily duped by the presence of positive words in a negative review. It seems that the pretrained model is better at putting together the collective meaning of a sentence, perhaps due it having trained over a broader corpus.

4 Testing your code

In the dropbox folder for this assignment, you will find a file `test_all.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.

Submission Instructions

In your submission, include the following:

- readme.(txt|pdf) that includes your answers to §1 and §3.
- `hw9.tar.gz` containing:
  - run_hw9.sh. This should contain the code for activating the conda environment and your run commands for §3 above. You can use run_hw2.sh from the previous assignment as a template.
  - model.py