Machine Learning Methods

Exercise 5

Last Update: 29.2.2024

1 General Description

In this final exercise, we'll explore the world of transformers and auto-regressive generative models. We'll delve into various aspects, including implementing self-attention and layer-normalization from scratch. We'll analyze the impact of self-attention on sentiment analysis tasks, examining how this mechanism influences model performance. Additionally, we'll use a generative model, specifically GPT2, to generate coherent and contextually relevant text. Through these tasks, you'll gain a deeper understanding of transformers and their applications in natural language processing.

You are expected to hand in a report, no longer than 6 pages, describing your experiments and answers to the questions in the exercise. The submission guidelines are described in Sec. 9, please read them carefully.

Note 1: You must explain your results in each question! Unless specified otherwise, an answer without explanation will not be awarded marks.
Note 2: When you are asked to plot something, we expect that plot to appear in the report.

2 Seeding

You should seed your code as in the previous exercises. Specifically, we will need to see both numpy and pytorch as follows:
(i) `np.random.seed(42)`
(ii) `torch.manual_seed(42)`
NOTE: It doesn't matter which seed you just that your results are reproducible!

3 Resources

In this exercise you are required to train models using a GPU (Graphics Processing Unit). This requires a configurable GPU for working with PyTorch. Implement your exercise in Google Colab, which is enough. Google Colab works like a Jupyter Notebook. Therefore, you may submit a Jupyter Notebook as your code. Remember to choose the GPU runtime under the "Runtime" → "Change runtime type" tab. **YOU DO NOT NEED ANYTHING MORE THAN THE FREE VERSION FOR THIS EXERCISE!**. Our experiments are very lightweight. If you are having trouble with slow runtimes (hours) your implementation is probably very inefficient.

4 Prerequisites

Before delving into the implementation of Self-Attention and LayerNorm modules, there are a few prerequisites to ensure smooth execution of the provided exercise. We provide skeleton code in two files: `models.py` and `sentiment_analysis.py`. To use these files effectively, you'll first need to set up your virtual environment, similar to previous exercises, and activate it. Next, you'll need to install two new libraries that may not have been previously used: `spacy`, an NLP library for tokenization, and `torchtext`, a library that combines PyTorch and NLP functionalities. You can install these libraries using the following commands:

```
pip install tqdm
pip install torchtext==0.6.0
pip install spacy
python -m spacy download en_core_web_sm
```

Once these installations are complete, you should be ready to run the provided skeleton code smoothly.

Before proceeding further, we recommend running the `load_imdb_data()` function from the `sentiment_analysis.py` file. This function loads the IMDB dataset and creates training, validation, and test sets. It may take approximately 15-20 minutes to run for the first time, as it downloads GloVe embeddings, the IMDB dataset, and extracts vocabulary. Afterwards, the runs will be significantly faster. By completing these steps, you'll be well-prepared to start the upcoming sections of the exercise.

5 Implementing Self-Attention and LayerNorm

In the first part of the exercise, we'll focus on implementing two fundamental components of transformer architectures: Self-Attention and Layer Normalization.

Self-Attention is a mechanism that allows the model to weigh the importance of different words in a sequence when processing each word. This is achieved

by computing attention scores between each pair of tokens (e.g. words) and using those scores to create a weighted sum of the corresponding word embeddings. Formally, given inputs x over a minibatch of size m, denoted as $B = \{x_1, x_2, ..., x_m\}$, where each sample x_i represents a sentence of size (T, d), with T denoting the number of tokens and d as the embedding size. Self-Attention maps x_i into query, key, and value feature spaces through a linear operation:

$$Q = x_i \cdot W_q^T$$
$$K = x_i \cdot W_k^T$$
$$V = x_i \cdot W_v^T$$

where W_q, W_k, and W_v are learnable parameters of size $d \times d$, consistent for all x_i, which can be easily implemented with the nn.Linear module. Consequently, the resulting Q, K, and V matrices are of size $T \times d$. Next, the Self-Attention mechanism is expressed as:

$$\text{Self-Attention}(x_i) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{1}$$

Here, Q, K, and V represent the query, key, and value matrices of the mapped tokens, respectively, with d denoting the embedding size. The softmax function normalizes attention scores across the entire sequence. For further details, please consult the lecture notes and presentation slides.

Layer Normalization is a technique used to normalize the activations of each layer in the network. It helps stabilize the training process by ensuring that the inputs to each layer have a consistent mean and variance. Formally, given inputs x over a minibatch of size m, denoted as $B = \{x_1, x_2, ..., x_m\}$, where each sample x_i represents a sentence of size (T, d), with T being the number of tokens and d as the embedding size. Each sample x_i can be viewed as containing $K = T \cdot d$ elements. To compute the mean μ_i and variance σ_i^2 for each sample x_i, we sum the elements and then normalize as follows:

$$\mu_i = \frac{1}{K}\sum_{k=1}^{K} x_{i,k}$$
$$\sigma_i^2 = \frac{1}{K}\sum_{k=1}^{K}(x_{i,k} - \mu_i)^2$$

We can now normalize each sample x_i:

$$LayerNorm(x_i) = \gamma \cdot \left(\frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}\right) + \beta \tag{2}$$

Following this, each sample is normalized to have a zero mean and unit variance, with an added term ϵ for numerical stability. γ and β serve as learnable parameters of size K for scaling and shifting each of the K dimensions (multiplication

and addition are element-wise), respectively. We will use $\epsilon = 10^{-8}$. For more information, see[1].

Task: Your assignment is to implement these two modules, `MySelfAttention` and `MyLayerNorm`, in the provided `models.py` file. Both `MySelfAttention` and `MyLayerNorm`, must be implemented without for-loops. Once you have completed your implementation, we will integrate it into the Transformer framework.

Hints: (i) You can create your own tester function using for-loops. By this we mean that you can implement this part of the exercise using for-loops as a sanity checker, and then use its output as a tester function for the implementation without using for-loops. (ii) You may find `torch.bmm` extremely useful. It provides the ability to perform tensor and matrix multiplications without requiring a for-loop. (iii) You might find it easier to complete the implementations and testing on your own computer, and then migrate it to Google Colab when it's ready.

6 Sentiment Analysis

We will now move on to the sentiment analysis task using the IMDB review dataset, which consists of movie reviews classified as positive (1) or negative (0) sentiment. In typical NLP tasks, we map text inputs to token embeddings. Here, we'll use the `spacy` library for tokenization and GloVe embeddings for vector representations. Your tasks in this section are:

1. Use your own implementations of `MyLayerNorm` and `MySelfAttention` to create your Transformer. We provide a Transformer code, `MyTransformer`, assuming your implementations of LayerNorm and Self-Attention. This Transformer will serve as the backbone for the sentiment analysis task, with IMDB reviews as input.

2. Implement the binary classification task in the `sentiment_analysis.py` file, where we've provided some initial code. The file contains missing parts and the task itself, which needs to be completed.

Now, let's try our model. **Task:**

1. Train the Transformer with a single transformer block for 5 epochs using 0.0001 learning rate. Plot the training and validation loss for each epoch. Additionally, report the test accuracy at the end of training. **Hint:** Test accuracy should be around $80 - 85\%$.

6.1 Bonus - 5 points

Find a configuration that yields a test accuracy greater than 90%. Report it. You may perform more epochs, play with different learning rates and transformer blocks, etc.

[1] `https://leimao.github.io/blog/Layer-Normalization/`

7 Text Generation - Bonus 5 points

In the final part of this exercise, you'll gain a valuable skill in practical machine learning: leveraging GitHub repositories to implement or utilize desired models. Machine learning often involves re-using existing code for various tasks, and here, we'll explore text generation using a pre-trained GPT2 model. To help you with this process, we'll provide you with a popular GitHub repository that simplifies GPT2 usage. While working with such repositories can be complex, this one offers clear instructions and examples for easy implementation. Your task is to clone the repository from the following link: `https://github.com/karpathy/minGPT`.

Once you've cloned the repository, your objective is to create your own Python file that uses the provided code to generate English sentences. The repository contains numerous code examples demonstrating how to achieve this task effectively. For simplicity, we recommend using the `gpt2` model type, as larger models may not fit within Google Colab constraints.

Your final task involves selecting 10 two-word prefixes (e.g. "My name...", "Machine learning...") and instructing the model to generate 5 sentences for each prefix. After generating the sentences, you should document your results in the provided PDF, showcasing the chosen prefixes along with the corresponding generated sentences. By completing this task, you'll gain hands-on experience with text generation using GPT2 but also develop a deeper understanding of how to use external repositories for machine learning tasks. *It is not necessary to submit the entire repository's code. You only need to submit your new file.*

8 Ethics

We will not tolerate any form of cheating or code sharing. You may only use the `numpy`, `pandas`, `mathplotlib`, `xgboost`, `torch`, `torchtext`, `spacy`, `tqdm` and `sklearn` libraries.

9 Submission Guidelines

You should submit your report, code and README for the exercise as $ex5\_\{YOUR\_ID\}.zip$ file. **Other formats (e.g. *.rar*) will not be accepted!**.

The README file should include your name, cse username and ID.

Reports should be in PDF format and be no longer than 6 pages in length. They should include any analysis and questions that were raised during the exercise. Please answer the questions in Sec. 6 and Sec. 7 in sequential order. **You should submit your code (including your modified helpers file), Report PDF and README files alone without any additional files.**

9.1 Submission

Please submit a zip file named $ex5\_\{YOURID\}.zip$ that includes the following:

1. A README file with your name, cse username and ID.
2. The *.py* files with your filled code.
3. A PDF report.