

Homework 2: PyTorch and Self-Attention

INFO 159 / 259, Spring 2024


Due: February 13, 2024 (11:59 PM)

This homework is designed to help you:

1. familiarize yourself PyTorch and implement a feedforward neural network (FFNN)
2. understand the self-attention mechanism that is crucial to the Transformer architecture

This document contains the instructions for this homework, and the accompanying notebook can be found here:

<https://github.com/ucbnlp24/hws4nlp24/blob/main/HW2/HW2.ipynb>

To use it, like in HW1, hit the  **Open in Colab** button at the top, and be sure to save a copy in your Google Drive or Github before you start to work on the file.

Submit your work on Gradescope. For code questions (Q2, Q4–5), replace `raise NotImplementedError` with your solutions, and as in HW1, keep your code between the `#BEGIN SOLUTION` and `#END SOLUTION` flags. For short-answer questions (Q1 and Q3), submit all your answers as a `.pdf` file separately (see §4 for instructions on how to submit).

Note: Here is the [link](#) to a comprehensive PyTorch tutorial.

1 Deliverable 1: PyTorch and FFNN

The first deliverable can be seen as a tutorial of PyTorch, which is a standard library that researchers use to implement neural networks.

QUESTION 1 [10 pts] : **PyTorch and embeddings** (writeup only).

In the notebook, you can see that a PyTorch implementation of logistic regression is offered for your reference. Study this script carefully. Start with how we load and access the GloVe embeddings. The most critical ingredients of a neural net in PyTorch include a class that defines the architecture (pay attention to the `forward` method), an optimizer (`torch.optim.Adam`),

and a loss function, (`torch.nn.CrossEntropyLoss()`), which combines the softmax function `torch.nn.LogSoftmax()` and negative log-likelihood `torch.nn.NLLLoss()`¹.

A notable difference from HW1 here is that the input of this model is the average of GloVe embeddings for all words in a movie review². What is the difference between averaged embeddings and BoW? What are the advantages or disadvantages of each? Discuss in no more than 50 words.

QUESTION 2 [10 pts]: **Feedforward neural network (FFNN).**

For this question, we want to add a hidden layer to the logistic regression classifier you've seen in Q1. Implement Eqn. (7.13) in J&M SLP3 and let the non-linearity g be \tanh . Your implementation should be similar to the `LogisticRegressionClassifier` example in the notebook. Let's pick 20 for the size of the hidden layer—this is provided in the `__init__()` function (`hidden_dim=20`), so you don't need to worry about it. Note that in the J&M terminology, a “two-layer” network has one hidden layer, which is what you will be implementing. You should fill in the parts between “#BEGIN SOLUTION” and “#END SOLUTION.”

2 Deliverable 2: Self-attention basics

The self-attention mechanism is often thought of as one of the most transformative ideas in modern NLP. Its full form in Transformer, as introduced in “Attention is All You Need,”³ is rather involved. This deliverable aims to prepare you for it.

We will start with the simplest form of self-attention: scaled dot-product self-attention. The goal is to try to understand the roles that query, key, and value vectors play in attending to the input sequence: conceptually, what do they aim to achieve and improve on? How do you code this in Python? Instead of focusing on training models, here we really want to see vector transformations in action.

¹ See documentation: <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>

² For more on GloVe embeddings, see: <https://nlp.stanford.edu/projects/glove/>

³ Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, “Attention Is All You Need” (NIPS 2017).

QUESTION 3 [10 pts]: The Concept of Self-attention and Multi-head attention (writeup only).

3.1: To review the concept of self-attention and multi-head attention read section § 10.1 of [Chapter 10 in SLP 3](#). This will be pages 2-8 in the pdf linked. After reading these sections, answer the questions below. In Question 4 you will implement self-attention and in Question 5 you will implement multi-head attention as part of a model.

Why is the self-attention mechanism useful in NLP? What are the drawbacks of not using self-attention? Name one similarity and difference between self-attention and multi-head attention.

Respond in **no more than 100 words**. Points will be taken off for excessively long answers.

3.2: Fill in the blanks below based on your understanding of section "10.1.2 Multihead Attention".

Each head in a self-attention layer is provided with its own set of ____, ____, and ____ matrices: W^Q , W^K , and W^V . These matrices are used to project the inputs into separate key, value, and query embeddings separately for each head, with the rest of the ____ computation remaining unchanged.

QUESTION 4 [10 pts]: Scaled dot-product attention in NumPy.

In the notebook, we'll go through the basics of implementing the steps in attention outside of any model. We'll do that in NumPy to help you see the changes in dimensions over this process. Then, for this question, implement the attention function below in NumPy. Recall from lecture that attention given a query vector Q , key vector K and value vector V is given by the following equation:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

Where Q , K and V are the query, key and value matrices created by multiplying an input embedding matrix (for a sentence) by W^Q , W^K and W^V , respectively. You will calculate Q , K , V within the body of this function. The sole required argument to this function should be a 2-D matrix $\in \mathbb{R}^{n \times \text{input_embedding_size}}$ for any arbitrary n (that is, corresponding to a sentence of arbitrary

length). It should return a matrix of that same exact size that is the output of that attention process over the input, given the parameters specified below. `dk` here is the size of the key vector (`query key size=37`).

Hints:

- When taking softmax, remember to operate on the correct dimension. Work through the “Intuition of attention” section if you’re unsure.
- Use `a.shape` or `a.size()` to view the dimensions of a torch variable `a`, which you’ll need to get `dk`.
- For matrix multiplication between `A` and `B`, use `A @ B`.

QUESTION 5 [10 pts]: Scaled dot-product attention in PyTorch. (Multihead Attention)

Now it’s time to implement that as part of a model. Here we’re going to embed attention within a larger model. For an input document of, say, 20 words, each represented by a 100-dimensional embedding, the input to attention is a 20×100 matrix; the output from attention is also a 20×100 matrix. In this larger model, we’re going to average those output embeddings to generate a final document that’s a single 100-dimensional vector; pass through a fully-connected dense layer to make a prediction.

You will finish the implementation of the `MultiHeadAttention` class in the notebook. When used in a model (as distinct from the NumPy example above), it is important to note that W^Q , W^K , W^V , and W^O are all trainable (this is how the attention mechanism learns). You’ll see in the provided code that the output of your attention mechanism is averaged over all token embeddings and then fed into a learnable linear layer to make a categorical prediction—that’s all taken care of for you, so you only need to implement the multi-head attention module. Since W^Q , W^K , W^V , and W^O are matrices, initialize them as a linear layer (`nn.Linear`) without the learnable additive bias (i.e., `bias=False`). Do not forget to use `num_heads` to create multiple such matrices. You can also initialize the softmax layer in the `__init__()`. The `forward()` method should be a very straightforward implementation of the formula above and can be done within ten lines of code. For further help, check [SLP3 Chapter 10](#) pages 9 and 10.

3 Debugging guides

For this homework, you might find it helpful to use these tips to help debug your code:

- The PyTorch function `.shape` can be called on torch variables to view their dimensions. This might be helpful in ensuring the layer dimensions are set correctly.

- Python debugger is a valuable tool which you can use to step through your program's execution. The line of code `"import pdb; pdb.set_trace()"` will add a breakpoint to your code, where you can view dimensions of PyTorch layers and experiment with function calls.

4 How to submit

Submit your work to Gradescope. There are two different Gradescope submission links, one for the PDF writeup and one for code:

- Submit to: "Gradescope HW2 writeup"
 - Submit all your answers to Q1 and Q3 as one PDF file to Gradescope. As long as you submit a PDF, there's no additional requirements for formatting. There is no template for writeups.
- Submit to: "Gradescope HW2 code (.ipynb)"
 - Make sure you have replaced all `raise NotImplementedError` with your solutions, and that they are placed, as in HW1, between the `# BEGIN SOLUTION` and `#END SOLUTION` flags.
 - Download your Colab notebook as an `.ipynb` file (File → Download `.ipynb`)
 - Submit `HW_2.ipynb`. The file must be named in this way for the Gradescope autograder.