

# HOMWORK 4: DEEP LEARNING

10-701 Machine Learning (Spring 2026)

Carnegie Mellon University

<https://piazza.com/cmu/spring2026/10701>

OUT: Friday, March 13th, 2026\*

DUE: Tuesday, March 24th, 2026 11:59 PM

## START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., “Jane explained to me what is asked in Question 2.1”). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section in our [Course Syllabus](#) for more information:
- **Late Submission Policy:** See the late homework policy in our [Course Syllabus](#).
- **Submitting your work:** There will be two submission slots for this homework on Gradescope, the Written and the Programming:
  - **Written:** Submit your homework to the Homework 4: Written Gradescope submission slot. Please use the provided template. The best way to format your homework is by using the Latex template released in the handout and writing your solutions in Latex. However submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Each derivation/proof should be completed in the boxes provided below the question, **you should not move or change the sizes of these boxes** as Gradescope is expecting your solved homework PDF to match the template on Gradescope. If you find you need more space than the box provides you should consider cutting your solution down to its relevant parts, if you see no way to do this, please add an additional page at the end of the homework and guide us there with a ‘See page xx for the rest of the solution’.
  - **Programming:** You are also required to upload your code, which you wrote to solve the final questions of this homework, to the Programming submission slot. Your code may be run by TAs so please make sure it is in a workable state.
- Regrade requests can be made after the homework grades are released, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted.

For multiple choice or select all that apply questions, shade in the box or circle in the template document corresponding to the correct answer(s) for each of the questions. For  $\text{\LaTeX}$  users, use ■ and ● for shaded boxes and circles, and don’t change anything else. If an answer box is included for showing work, **you must show your work!**

---

\*Compiled on Saturday 14<sup>th</sup> March, 2026 at 05:03

## 1 Convolutional Neural Networks (6 Points)

1. [2pts] **Numerical answer:** Suppose the first layer of a CNN uses a  $5 \times 5$  convolutional filter and outputs 3 channels. It takes as input images of size  $100 \times 100$ . It also uses a padding of 3 and a stride 2 in both input dimensions. What are the dimensions of the output of this layer?

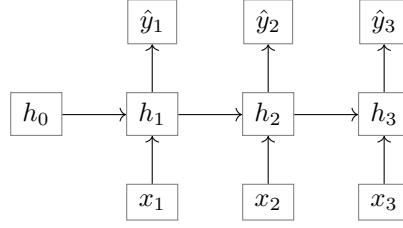
2. [2pts] **Numerical answer:** Suppose you have a CNN that takes inputs of shape  $3 \times 100 \times 100$ . Suppose the first convolutional layer uses  $3 \times 5 \times 5$  filters and outputs 8 channels. How many parameters does this layer have (including a scalar bias for each output channel)?

3. [2 pts] **Select all that Apply:** Which of the following statements about pointwise (sometimes called  $1 \times 1$ ) convolutions is/are true?

- ☐ Pointwise convolutions introduce non-linearity between two CNN layers
- ☐ A pointwise convolution can be formulated as a fully connected layer that maps each stack of pixels (across all channels) in the input to a pixel in the output
- ☐ Pointwise convolutions can reduce the overall number of parameters in a model by reducing the total size of the following layer
- ☐ Pointwise convolutions preserve the number of channels but may impact other dimensions
- ☐ None of the above

## 2 Recurrent Neural Networks (10 Points)

1. Consider the following simple RNN architecture:



where the dimensions of the layers and their corresponding weights are given below:

$$\begin{aligned}
 \mathbf{x}_t &\in \mathbb{R}^3 & \mathbf{W}_{hx} &\in \mathbb{R}^{4 \times 3} \\
 \mathbf{h}_t &\in \mathbb{R}^4 & \mathbf{W}_{hy} &\in \mathbb{R}^{2 \times 4} \\
 \mathbf{y}_t, \hat{\mathbf{y}}_t &\in \mathbb{R}^2 & \mathbf{W}_{hh} &\in \mathbb{R}^{4 \times 4}
 \end{aligned}$$

The loss and update equations are:

$$\begin{aligned}
 J &= \sum_{t=1}^3 J_t \\
 J_t &= - \sum_{i=1}^2 y_{t,i} \log(\hat{y}_{t,i}) \\
 \hat{\mathbf{y}}_t &= \sigma(\mathbf{o}_t) \\
 \mathbf{o}_t &= \mathbf{W}_{hy} \mathbf{h}_t \\
 \mathbf{h}_t &= \psi(\mathbf{z}_t) \\
 \mathbf{z}_t &= \mathbf{W}_{hh} \mathbf{h}_{t-1} + \mathbf{W}_{hx} \mathbf{x}_t \\
 \mathbf{h}_0 &= \mathbf{0}
 \end{aligned}$$

Here,  $\sigma$  is the **softmax** activation and  $\psi$  is the **identity** activation (i.e., no activation).  $J$  is the cross entropy loss and  $t$  indicates timesteps. Note that we have no intercept terms (biases) in this architecture.

In this question, you will derive the steps of the backpropagation through time algorithm that lead to the computation of  $\frac{dJ}{d\mathbf{W}_{hh}}$ . For all parts of this question, please write your answer in terms of  $\mathbf{W}_{hh}$ ,  $\mathbf{W}_{hy}$ ,  $\mathbf{y}_t$ ,  $\hat{\mathbf{y}}_t$ ,  $\mathbf{h}_t$ , and any additional terms specified in the question.

- (a) [2 pts] What is  $\mathbf{G}_{\mathbf{o}_t} = \frac{\partial J}{\partial \mathbf{o}_t}$ ? Write your solution in the first box, and show your work in the second. Write your answer in terms of  $\hat{\mathbf{y}}_t$  and  $\mathbf{y}_t$ .

$$\frac{\partial J}{\partial \mathbf{o}_t}$$

Work

- (b) [4 pts] What is  $\mathbf{G}_{\mathbf{h}_i} = \frac{\partial J}{\partial \mathbf{h}_i}$  for an arbitrary  $i \in [1, 3]$ ? Write your solution in terms of  $\mathbf{G}_{\mathbf{o}_t}$ ,  $\mathbf{W}_{hh}$ ,  $\mathbf{W}_{hy}$  in the first box, and show your work in the second.

$$\frac{\partial J}{\partial \mathbf{h}_i}$$

Work

- (c) [4 pts] What is  $\mathbf{G}_{\mathbf{w}_{hh}} = \frac{\partial J}{\partial \mathbf{w}_{hh}}$ ? Write your solution in terms of  $\mathbf{G}_{\mathbf{h}_i}$  and  $\mathbf{h}$  in the first box, and show your work in the second.

$$\frac{\partial J}{\partial \mathbf{w}_{hh}}$$

Work

### 3 Attention & Transformers (4 Points)

1. [2pts] **Short Answer:** One limiting factor in a Transformer architecture for processing longer sequence lengths is its fixed context window. Does this limitation also apply to RNNs? Why or why not?

2. [2 pts] **Select One:** Given an input sequence of length  $n$ , how does the time complexity of the self-attention mechanism scale with  $n$ ?

- ☐  $O(n)$   
☐  $O(n \log n)$   
☐  $O(n^2)$   
☐  $O(n^3)$

### 4 Algorithmic Fairness (5 Points)

1. [5pts] In class we identified some criticisms of the COMPAS classifier for recidivism risk. In this question you will discuss changes that COMPAS could have made to the process of learning and using the classifier.
- (a) [2.5 pts] Identify one change that COMPAS could have made to the process of training the classifier, with the goal of addressing a criticism we discussed in class. That is, keeping the same training and validation data they used, how could COMPAS have delivered a different function that maps defendants to risk scores? Describe your change and give a brief explanation of what its effects might be. Be sure to discuss both positive and negative consequences of your changes.

- (b) **[2.5 pts]** Identify two changes that COMPAS could have made outside the process of training the classifier — e.g., in how they set up the problem, how they collected data, or how the delivered classifier was used. Again, describe your changes and give brief explanations of what each one's effects might be, and cover both positive and negative consequences.

## 5 Programming: RNN and Transformer in PyTorch [45 pts]

In this programming task, you will implement an RNN and an encoder-only Transformer in PyTorch. The task we will be working on is a binary classification task on text data; your neural networks will predict whether each text sequence exhibits positive sentiment or negative sentiment. Out of the 53 points for this question, 27 points are from the written, 13 points are from passing the RNN autograder, and 13 points are from passing the Transformer autograder.

**Important Note on PyTorch:** In this programming task you are REQUIRED to use the automatic differentiation in PyTorch. You are allowed to use most of the built-in neural network modules in PyTorch (such as `nn.Linear`, `nn.ReLU`, `nn.Softmax`, etc.) unless otherwise specified.

**PyTorch modules that are NOT allowed in this task:**

1. All recurrent layers: `nn.RNNBase`, `nn.RNN`, `nn.RNNCell`, `nn.LSTM`, `nn.LSTMCell`, `nn.GRU`, `nn.GRUCell`
2. All attention and Transformer layers: `nn.MultiHeadAttention`, `nn.Transformer`, `nn.TransformerEncoder`, `nn.TransformerEncoderLayer`

### 5.1 The Yelp Dataset

The dataset used in this assignment is the Yelp dataset, the same dataset used in the programming section of Homework 2. This dataset uses polarized labels, where reviews with 1 and 2 stars are labeled negative (“0” label) and reviews with 4 and 5 stars are labeled positive (“1” label). This specific dataset can be found [at this link](#) (though you don’t need to download it from there, see our dataloader below) and is a subset of the full Yelp reviews dataset. This polarized version contains 598K text reviews (560k for training, 38k for testing) of various types of establishments, and we have further subsampled it to 15% of the original size, such that the training set and testing set have 84k and 5.7k samples respectively.

The Yelp dataset will be loaded from the Hugging Face file system via Pandas. Each entry of the dataset contains 2 columns, the review text and its polarized label.

#### 5.1.1 Provided Dataloader

For efficiency, we will only take the first  $L = 50$  words in each review for training and testing. We provide a dataset class and function to initialize the dataset and the dataloader for you in `reference_data.py`. **Please carefully read and make sure you understand all the code in this file.**

The `reference_data.py` class preprocesses the text reviews by translating each word into an index representing that word. The word-to-index mapping is stored in the `self.word_dict` variable. Note that we include an additional padding token “PAD” that has index 0. We append the padding token to reviews that have lengths less than  $L$ , so that all reviews have the same length and can be batched together.

The dataloader will return a tuple containing the text tensor and the label tensor. If  $B$  is the batch size, then the text tensor has shape  $(B, L)$ , where each element represents the dictionary index of the word at that specific position. The label is a one-dimensional tensor with length  $B$ .

### 5.2 Implementing an RNN

#### 5.2.1 Word Embedding

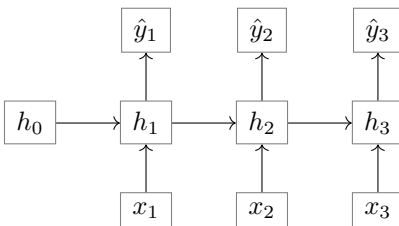
To convert the word indices into a word embedding, create an embedding layer using `torch.nn.Embedding`. The embedding layer takes in batched, word-index tensors with shape  $(B, L)$  and outputs an embedding  $\mathbf{X}$  with shape  $(B, L, D_e)$ , where the hyperparameter  $D_e$  is the embedding dimension. The embedding layer maintains the embedding table tensor of shape  $(N, D_e)$  that will be optimized during training.  $N$  is the number of words in the dictionary. The  $k$ -th row in the embedding table will be the output vector corresponding to the input word at index  $k$ . Note that for the padding token “PAD” with index 0, we want to enforce its embedding to be a zero vector.



### 5.2.2 LayerNorm

We covered layer normalization in class. It computes the mean and variance of a tensor across the feature dimension (as opposed to the batch dimension in batch normalization) and normalizes the tensor with respect to the computed mean and variance. It then learns a new mean and variance for the output and scales the normalized tensor according to those learned parameters. You should use `nn.LayerNorm` to perform layer normalization in your networks. Please see the equation below in Section 5.2.3 to see where LayerNorm should be applied in the RNN layers.

### 5.2.3 RNN



Given a working dataloader and embedding layer, we will now implement a simple RNN with one hidden layer. (Note: this is *not* the same as the RNN you worked with in the written section above; e.g., that one had no LayerNorm.) The structure of the network is shown in the above diagram.  $\mathbf{x}_i$ ,  $\mathbf{h}_i$ ,  $\hat{\mathbf{y}}_i$  represents the input word embedding, hidden state, and output at the  $i^{\text{th}}$  recurrent step respectively. Your network should contain three linear layers: input-to-hidden, hidden-to-hidden, and hidden-to-output layer, with weights represented by  $\mathbf{W}_{xh}$ ,  $\mathbf{W}_{hh}$ , and  $\mathbf{W}_{hy}$  respectively. You should initialize  $\mathbf{h}_0$  as a zero vector. Given the previous hidden state  $\mathbf{h}_{i-1}$  and the  $i$ -th word embedding  $\mathbf{x}_i$ , you can compute  $\mathbf{h}_i$  and  $\hat{\mathbf{y}}_i$  as

$$\mathbf{h}_i = \tanh(\text{LayerNorm}(\mathbf{W}_{hh}\mathbf{h}_{i-1} + \mathbf{W}_{xh}\mathbf{x}_i)),$$

$$\hat{\mathbf{y}}_i = \text{softmax}(\mathbf{W}_{hy}\mathbf{h}_i).$$

### 5.2.4 Loss and Prediction

Our RNN computes an output at every recurrent step. **Important:** You need to compute the loss for all of the outputs and sum over the recurrent steps to compute your total loss for each iteration. Use `nn.CrossEntropyLoss` as the loss criterion. During training, the same label should be used to calculate the loss for each output in the entire sequence, which is then summed over. However, for prediction, we only take the output at the very last recurrent step  $\hat{\mathbf{y}}_L$ , and predict the label by taking the argmax of the last output.

### 5.2.5 Train and Test your RNN

After finishing your RNN implementation, initialize your model, optimizer, and loss function. For this assignment, please use the Adam optimizer.

You are also asked to write training and testing loops over the training dataloader. For each batch of inputs, run the forward and backward passes, then update the model using the optimizer. Train your model using the hyperparameters specified in the starter code. For testing your model, loop over the test dataloader and run a forward pass with `torch.no_grad()`. For this step, **we highly recommend reviewing supplementary material in Piazza post @79 on implementing training and testing loops in PyTorch.**

Please log the average loss and prediction accuracy at each epoch in both your training and testing loop, as you will be asked to report and plot these values for your written responses in this section.

## 5.3 Implementing a Transformer

### 5.3.1 Transformer Embedding

For the Transformer, we will use the same method as described above to convert word indices into embeddings. However, on top of that, we also need to incorporate positional encodings that provide the model information on the relative location of each token in the sequence. We add the positional encoding to the word embedding as the input

to the model. The class `TransformerEmbedding` does this and is provided for you in the starter code. This positional encoding follows the implementation described in “Attention Is All You Need” (Vaswani et. al., 2017).

### 5.3.2 Multi-head Attention Layer

Implement the `AttentionLayer` class and then the `MultiHeadAttention` class. For single-head attention, given input  $\mathbf{X}$  with shape  $(B, L, D_e)$ .

$$\mathbf{Q} = \mathbf{W}_Q \mathbf{X}, \mathbf{K} = \mathbf{W}_K \mathbf{X}, \mathbf{V} = \mathbf{W}_V \mathbf{X}$$

where adjacent tensors denotes a batched matrix multiplication (or equivalently, tensor contraction) operation,  $\mathbf{Q}$  has shape  $(B, L, D_Q)$  and  $D_Q$  is the dimension of  $\mathbf{Q}$ ;  $\mathbf{K}$  and  $\mathbf{V}$  behave similarly. Note that the dimensions of  $\mathbf{Q}$  and  $\mathbf{K}$  must be the same.

Now we can compute the attention score and the final output of the attention layer as

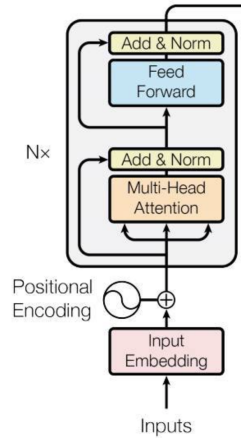
$$\text{score} = \text{softmax} \left( \frac{\mathbf{Q} \mathbf{K}^\top}{\sqrt{D_Q}} \right), \mathbf{z} = \text{score} \mathbf{V}$$

where  $\text{score}$  has shape  $(B, L, L)$ ,  $\mathbf{z}$  has shape  $(B, L, D_V)$ , and  $D_V$   $D_Q$  are hyperparameters (additionally, note that the transpose does not operate over the batch dimension).

Multi-head attention with  $N$  heads should contain  $N$  attention layers. It computes outputs for each attention head and concatenates the outputs  $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N$ . The concatenated output has shape  $(B, L, N \times D_V)$ .

Lastly, we pass this output through a linear layer to reduce the dimension from  $N \times D_V$  to  $D_{\text{model}}$ , giving a final output of shape  $(B, L, D_{\text{model}})$ .

### 5.3.3 Transformer



The structure of a Transformer encoder layer is shown in the above figure. (We will use an encoder-only architecture.)

Given input  $\mathbf{X}$ , it computes the intermediate tensor  $\mathbf{z}$  and output  $\hat{\mathbf{y}}$  as

$$\mathbf{z} = \text{LayerNorm}(\mathbf{X} + \text{MultAttn}(\mathbf{X})), \hat{\mathbf{y}} = \text{LayerNorm}(\mathbf{z} + FF(\mathbf{z}))$$

where `MultAttn` represents multi-head attention and `FF` represents a feed-forward network. The `FF` network should consist of two linear layers and one ReLU activation after the first linear layer. Note the residual connections (add back in  $\mathbf{X}$  when computing  $\mathbf{z}$ , add back in  $\mathbf{z}$  when computing  $\hat{\mathbf{y}}$ ). Also note that we perform layer normalization *after* each residual connection. In your implementations,  $\mathbf{z}$  and  $\hat{\mathbf{y}}$  should have the same shape as input  $\mathbf{X}$ . Implement the `FeedForward`, `Residual`, and the `EncoderLayer` classes. The Transformer contains multiple `EncoderLayer` modules and passes the input embedding through all the layers sequentially. The starter code gives the hyperparameters you should use (e.g., width of the FF network and number of Transformer layers).

### 5.3.4 Loss and Prediction

For loss and prediction, we will use the output of the *first* word token “SOS”, a dummy “start of sentence” token we add to each text review. We pass this output through one more linear layer to reduce the dimension to 2, the number of classes in our task. Please use `nn.CrossEntropyLoss` for the loss and `argmax` to translate the model output to labels.

### 5.3.5 Train and Test your Transformer

We encourage you to reuse the same code you wrote for training and testing your RNNs to train and test your Transformers.

## 5.4 Empirical Questions

1. **[4 pts]** Train both the RNN and the Transformer encoder for 10 epochs using the hyperparameters specified in the starter code. Report the final accuracy of both models on both the test dataset and the training dataset. Report values up to the 4th decimal place.

RNN Training Accuracy:

RNN Test Accuracy:

Transformer Training Accuracy:

Transformer Test Accuracy:

2. **[5 pts]** Describe the relationship between the values you reported in the previous question: does one model significantly outperform the other? Provide an explanation for the relative performances of the two models using what you know of the task and how we preprocessed the dataset.

3. [10 pts] Now we want to give you a chance to experiment with Transformers by exploring ways to enhance performance. You will need to implement one of the non-trivial enhancements listed below and write a report on your findings. In other words, your report should **not** just be tuning hyperparameters. Your report should include:

- (a) A plot that shows the train and test loss for your newly enhanced Transformer.
- (b) Model size (number of parameters).
- (c) A description of the improvement(s) you made.
- (d) The set of hyperparameters you used (if it differs from the ones given). E.g., you could consider changing embedding dimension (`d_model`) or number of transformer layers (`n_layers`).
- (e) The time required to train the model.
- (f) An analysis your results in terms of improvements you made and the hyperparameters you used.

Please implement one of the following major changes to your encoder-only transformer and report the effect on performance:

- **SwiGLU:** Replace the standard feed-forward network with SwiGLU, a gated linear unit that combines a linear transformation with a sigmoid-activated gate. This can enhance the model's expressiveness and has been shown to improve performance in large language models. <sup>1</sup>
- **Switch from Post to Pre Normalization:** Transition to pre-normalization, where layer normalization is applied before the attention and feed-forward computations, as used in models like Llama. This can improve training stability, especially for deeper architectures. <sup>2</sup>
- **Relative Positional Encodings:** Integrate relative positional encodings into the attention mechanism to dynamically capture the relationships between tokens. Unlike fixed absolute encodings, relative encodings model the distance between tokens, enabling the model to generalize better. <sup>3</sup>
- **RoPE (Rotary Positional Embeddings):** Implement RoPE, which encodes positional information using rotation matrices. Note, this can be challenging to implement from scratch. It is most useful for handling longer sequences and is used in advanced models. <sup>4</sup>
- **Other Changes:** Consider additional modifications such as using different activation functions like GELU <sup>5</sup> or using RMSNorm instead of LayerNorm for potentially more efficient normalization. <sup>6</sup>

#### Important Requirements:

- Your model size should **not** be more than 3M parameters.
- You **must** include at least one of the listed non-trivial enhancements to your Transformer. Simply tuning hyperparameters will not result in full points.

In addition to your report below, please submit your new implementation to the autograder in the separate Gradescope assignment provided for this question. You will be tested on a held-out dataset, and should get full points if your test accuracy is above 82%.

<sup>1</sup><https://arxiv.org/abs/2002.05202>

<sup>2</sup>On Layer Normalization in the Transformer Architecture (<https://arxiv.org/abs/2002.04745>)

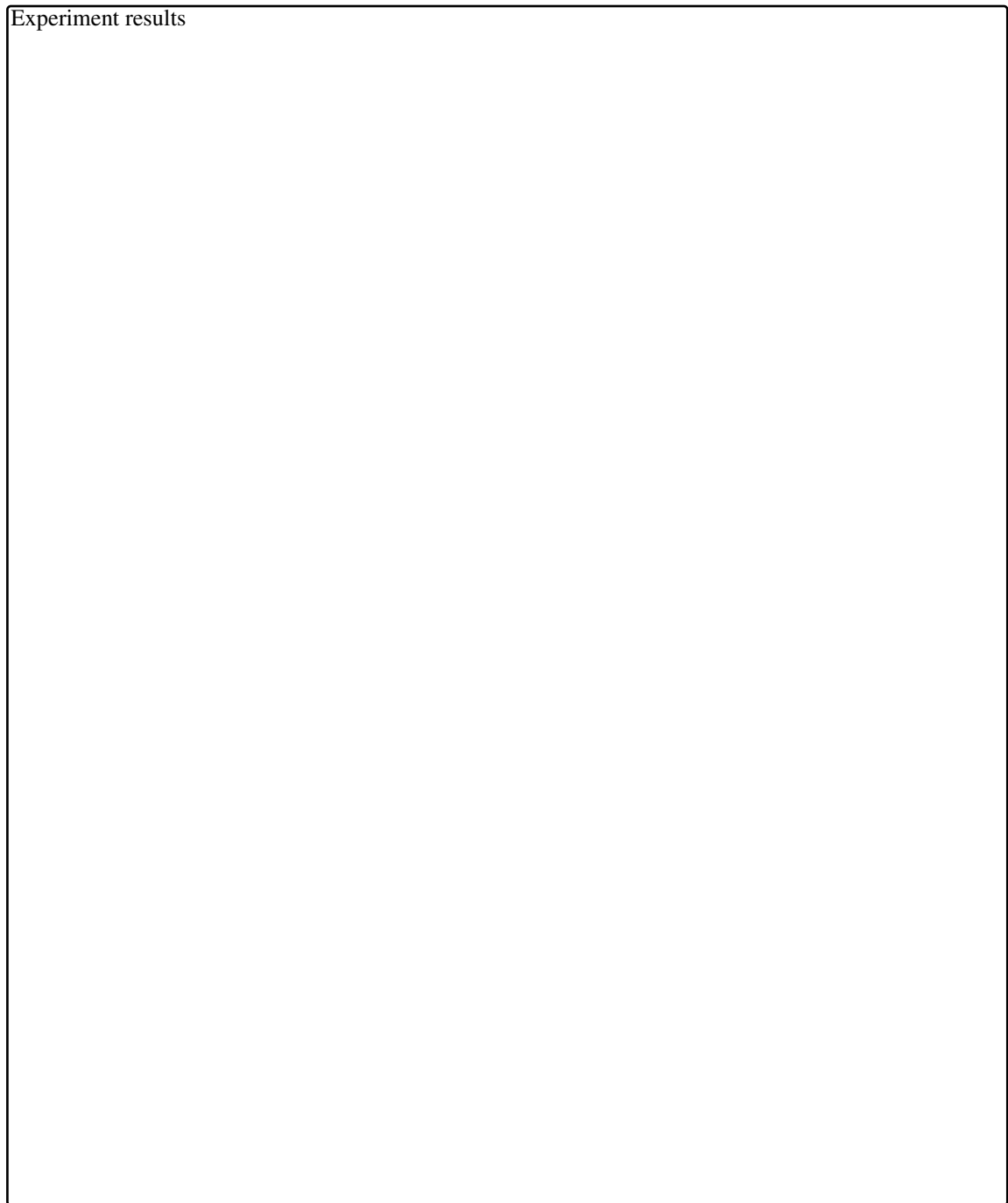
<sup>3</sup>Self-Attention with Relative Position Representations (<https://arxiv.org/abs/1803.02155>)

<sup>4</sup>RoFormer: Enhanced Transformer with Rotary Position Embedding (<https://arxiv.org/abs/2104.09864>)

<sup>5</sup>Gaussian Error Linear Units (<https://arxiv.org/abs/1606.08415>)

<sup>6</sup>Root Mean Square Layer Normalization (<https://arxiv.org/abs/1910.07467>)

Experiment results



Experiment results contin. (if needed)

## 6 Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment?  
(b) If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4")

2. (a) Did you give any help whatsoever to anyone in solving this assignment?  
(b) If you answered 'yes', give full details (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")

3. (a) Did you find or come across code that implements any part of this assignment?  
(b) If you answered 'yes', give full details (book & page, URL & location within the page, etc.).