ECE 661: Homework #3
Understand and Implement Sequence Models

Hai "Helen" Li

ECE Department, Duke University — Octobor 2, 2023

Objectives

Homework #3 covers the contents of Lectures 09∼10. This assignment includes basic knowledge about sequence models (i.e., RNNs and Transformers), detailed instructions on how to implement the core of the sequence models and the training pipeline for training sentiment analysis on the IMDB dataset. In this assignment, you will gain hands-on experience in training an LSTM and a GRU on the IMDB dataset, while also learning techniques for improving the performance of your models.

Some parts of this assignment require advanced features in the latest PyTorch version, which are unfortunately not available on our JupyterLab server. Thus **using Google CoLab is strongly encouraged** for finishing the lab questions. A brief introduction on setting up and using CoLab is provided at the beginning of Lab 1. When conducting the lab projects, actively referring to the **NumPy/PyTorch tutorial** slides on Sakai for the environment setup and instructions for NumPy/PyTorch utilities can be very helpful.

Warning: You are asked to complete the assignment independently.

This lab has 100 points plus 15 bonus points, yet your final score cannot exceed 100 points. The submission deadline will be **11:59pm, Wednesday, October 18**. For each lab, we provide one or multiple notebooks for you to start with. Your task is to fill in the missing modules in these notebooks. No new notebook or scripts need to be created. You will need to submit two independent files including:

1. A self-contained PDF report, which provides answers to all the conceptual questions and clearly demonstrates all your lab results and observations. Remember, **do NOT generate PDF from your Jupiter notebook to serve as the report**, which can increase the TA's burden of grading. Besides, you pdf file should also contain the plotted figures (instead of submit separately). You can upload a saved figure or a screenshot of the figure to the pdf report.

2. `LabRNN.ipynb`, a single Jupiter notebook that includes your work on Lab 1 and Lab 2.

Note that 20 percent of the grade will be deducted if the submissions doesn't follow the above guidance.

1 True/False Questions (30 pts)

For each question, please provide a short explanation to support your judgment. Note that for questions regarding BERT and GPT, the answers are not fully contained in the lecture. You may need to go thorugh the original paper to find the solutions.

Problem 1.1 (3 pts) In the self-attention layer of Transformer models, we compute three core variables—key, value and query.

Problem 1.2 (3 pts) In the self-attention layer of Transformer models, the attention is denoted by the cosine similarity between the key and query.

Problem 1.3 (3 pts) In the self-attention layer of Transformer models, after obtaining the attention matrix, we need to further apply a normalization on it (e.g., layer normalization or batch normalization).

Problem 1.4 (3 pts) The encoder of Transformer learns auto-regressively.

Problem 1.5 (3 pts) Both BERT's and GPT's architecture are Transformer decoder.

Problem 1.6 (3 pts) BERT's pre-training objectives include (a) masked token prediction (masked language modeling) and (b) next sentence prediction.

Problem 1.7 (3 pts) Both GPT and BERT are zero-shot learner (can be transferred to unseen domain and tasks).

Problem 1.8 (3 pts) Gradient clipping can be used to alleviate gradient vanishing problem.

Problem 1.9 (3 pts) Word embeddings can contain positive or negative values.

Problem 1.10 (3 pts) The memory cell of an LSTM is computed by a weighted average of previous memory state and current memory state where the sum of weights is 1.

2 Lab 1: Recurrent Neural Network for Sentiment Analysis (45 pts)

Google colab is a useful tool for modeling training. In order to run `ipynb` document, you need to upload your jupyter notebook document to google drive. Then double click the file in the google drive, which will lead you to google colab. Moreover, it'd be more efficient to use GPU to train your LSTM and Transformer models in this assignment. To use GPU, please check *Runtime > Change run-time type*, and select *GPU* as the hardware accelerator. You may click on side bar document icon to upload your dataset and python file there.

In this lab, you will learn to implement an LSTM model for sentiment analysis. Sentiment analysis [1, 2] is a classification task to identify the sentiment of language. It usually classifies data into two to three labels: positive, negative or/and neutral. IMDB [3] is one of the most widely used dataset for binary sentiment classification. It uses only positive and negative labels. IMDB contains 50,000 movie review data collected from popular movie rating service IMDB. You may find more details at `https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews`.

To start this assignment, please open the provided Jupyter Notebook `LabRNN.ipynb`. We have provided implementation for the training recipe, including optimizers, learning rate schedulers, and weight initialization. You may **NOT** change any of the hyperparameter settings (i.e., the object `HyperParams`) in this lab.

Grading Instructions. For this part, we mainly grade on the logic and conciseness of the code. If a mistake is found in this part that lead to erroneous conclusions for questions in the later part (e.g., Lab 2), we will consider this and provide partial/full credit for the later part, to avoid applying the same penalty twice. Note that TA and Professors have the final discretion to adjust grade according to the given submission.

(a) (5 pts) Implement your own data loader function. First, read the data from the dataset file on the local disk. Then split the dataset into three sets: train, validation, and test by $7:1:2$ ratio. Finally return `x_train`, `x_valid`, `x_test`, `y_train`, `y_valid` and `y_test`, where `x` represents reviews and `y` represent labels.

(b) (5 pts) Implement the `build_vocab` function to build a vocabulary based on the training corpus. You should first compute the frequency of all the words in the training corpus. Remove the words that are in the `STOP_WORDS`. Then filter the words by their frequency ($\geq$ `min_freq`) and finally generate a corpus variable that contains a list of words.

(c) (5 pts) Implement the `tokenization` function. For each word, find its index in the vocabulary. Return a list of integers that represents the indices of words in the example.

(d) (5 pts) Implement the `__getitem__` function in the `IMDB` class. Given an index i, you should return the i-th review and label. The review is originally a string. Please tokenize it into a sequence of token indices. Use the `max_length` paramete r to truncate the sequence so that it contains at most `max_length` tokens. Convert the label string ('positive' / 'negative') to a binary index, such as 'positive' is 1 and 'negative' is 0. Return a dictionary containing three keys: '`ids`', '`length`', '`label`' which represent the list of token ids, the length of the sequence, the binary label.

(e) (10 pts) Implement the LSTM model for sentiment analysis.

   (a) (5 pts) In `__init__`, a LSTM model contains an embedding layer, an lstm cell, a linear layer, and a dropout layer. You can call functions from Pytorch's nn library. For example, nn.Embedding, nn.LSTM, nn.Linear.

   (b) (5 pts) In `forward`, decide where to apply dropout. The sequences in the batch have different lengths. Write/call a function to pad the sequences into the same length. Apply a fully-connected (fc) layer to the output of the LSTM layer. Return the output features which is of size [batch size, output dim].

(f) (5 pts) As a sanity check, train the LSTM model for 5 epochs with **SGD optimizer**. Do you observe a steady and consistent decrease of the loss value as training progresses? Report your observation on the learning dynamics of training loss, and validation loss on the IMDB dataset. Do they meet your expectations and why? (**Hint**: trust what you have observed and report as is.)

(g) (10 pts) Gated Recurrent Unit (GRU) is a simplified version of LSTM that performs good on language tasks. Implement the GRU model similar instructions as (e), as follows:

   (a) (5 pts) In `__init__`, a GRU model includes an embedding layer, an gated recurrent unit, a linear layer, and a dropout layer. You can call functions from `torch.nn` library. For example, nn.Embedding, nn.GRU, nn.Linear.

   (b) (5 pts) In `forward`, decide where to apply dropout. The sequences in the batch have different lengths. Write/call a function to pad the sequences into the same length. Apply a Fully-Connected (FC) layer to the output of the GRU layer. Return the output features which is of size [batch size, output dim].

(h) (**Bonus**, 5 pts) Repeat (f) for GRU model and report your observations on the training curve. What do you hypothesize is the issue that results in the current training curve? (**Hint:** Check slides of Lecture 9 for the recommendations on training RNNs.)

Please do **NOT** run more than 5 epochs as several epochs suffices to achieve the expected results. You are required to carry the completed part to Lab 2.

3 Lab 2: Training and Improving Recurrent Neural Network(25 pts)

In this lab, you will explore the training of RNN models on IMDB dataset, and propose some design improvements over the existing implementation. you should continue to work on `LabRNN.ipynb` with your implementation of the IMDB dataloaders, transformations, RNNs (LSTM and GRU). You will start playing with the hyperparameter settings to improve the training on IMDB sentiment analysis tasks, and study better design choices that crafts a better recurrent neural network.

For question (a) and (b), you may follow the hint to properly set the learning rate for your optimizers. Despite that, you may **NOT** change any hyperparameters defined in `LabRNN.ipynb`, including the `HyperParams` object, RNN model configurations, and optimizer configurations.

(a) (5 pts) We start to look at the optimizers in training RNN models. As SGD might not be good enough, we switch to advanced optimizers (i.e., Adagrad, RMSprop, and Adam) with adaptive learning rate schedule. We start with exploring these optimizers on training a LSTM. You are asked to employ each of the optimizer on LSTM and report your observations. Which optimizer gives the best training results, and why?

(b) (5 pts) Repeat (a) on training your GRU model, and provide an analysis on the performance of different optimizers on a GRU model. How does a GRU compare with a LSTM, in terms of model performance and model efficiency?

Up to this stage, you should expect **at least 84%** accuracy on IMDB dataset. You may find that optimizers solve most of the existing dilemmas in RNN training. Next, we will start playing with the structures of RNN models and see if we can design a better RNN model. Starting here, you may change any of the hyperparameters **except random seed**. Yet, our recommendation of hyperparameter change is limited to learning rate, number of layers in RNN models, and the dimension of embedding/hidden units.

(c) (5 pts) Try to make your RNN model deeper by changing the number of layers. Is your RNN model achieving a better accuracy on IDMB classification? You may use LSTM as an example. (**Hint:** you do not need to explore more than 4 recurrent layers in a RNN model.).

(d) (5 pts) Try to make your RNN model wider by changing the number of hidden units. Is your RNN model achieving a better accuracy on IDMB classification? You may use LSTM as an example. (**Hint:** you do not need to explore a hidden dimension of more than 320 on IMDB).

(e) (5 pts) Embedding tables contain rich information of the input words and help build a more powerful representation with word vectors. Try to increase the dimension of embeddings. Is your RNN model achieving a better accuracy on IMDB classification? You may use LSTM as an example. (**Hint:** you do not need to explore an embedding dimension of larger than 256).

(f) (**Bonus**, 5 pts) A better way to scale up RNN models is simultaneously scaling up the number of hidden units, number of layers, and embedding dimension. This is called **compound scaling**, which is widely adopted in emerging ML research. You are asked to make **no more than 50** trials to perform a compound scaling on your implemented RNN models. Is the model crafted via compound scaling performing better than the models you obtain in (d), (e), and (f)? You may use LSTM as an example. (**Hint:** You may use the accuracy-parameter trade-off as a criterion.)

(g) (**Bonus**, 5 pts) RNNs can be bidirectional. Use the best model discovered in (f) and make it bidirectional. Do you observe better accuracy on IMDB dataset and why?

Please do **NOT** run more than 5 epochs as several epochs suffices to achieve the expected results. Completing (a) $\sim$ (e) and (g) should give an accuracy of 84$\sim$86%, and completing (f) should give an accuracy of 86$\sim$88%. Yet, you are only required to achieve **84%** to successfully complete Lab 2. You are required to submit the completed version of `LabRNN.ipynb` for Lab 2.

Info: Additional requirements:

- **DO NOT** copy code directly online or from other classmates. We will check it! The result can be severe if your codes fail to pass our check.

As this assignment requires much computing power of GPUs, we suggest:

- Plan your work in advance and start early. We will **NOT** extend the deadline because of the unavailability of computing resources.
- Be considerate and kill Jupyter Notebook instances when you do not need them.
- **DO NOT** run your program forever. Please follow the recommended/maximum training budget in each lab.
- Please do not train any models for more than 5 epochs, especially when you are doing (f) and (g) in Lab 2.

References

[1] P. D. Turney, "Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews," *arXiv preprint cs/0212032*, 2002.

[2] B. Pang and L. Lee, "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts," *arXiv preprint cs/0409058*, 2004.

[3] S. Dooms, T. De Pessemier, and L. Martens, "Movietweetings: a movie rating dataset collected from twitter," in *Workshop on Crowdsourcing and human computation for recommender systems, CrowdRec at RecSys*, vol. 2013, p. 43, 2013.