HOMEWORK 1: NEURAL NETWORKS FOR SEQUENCE TAGGING

10-418/10-618 Machine Learning for Structured Data (Fall 2022)
http://www.cs.cmu.edu/~mgormley/courses/10418/
OUT: Wednesday, September 7th
DUE: Friday, September 16th
TAs: Mukuntha, Harnoor, Eric

START HERE: Instructions

- **Collaboration Policy**: Please read the collaboration policy here: http://www.cs.cmu.edu/~mgormley/courses/10418/syllabus.html
- **Late Submission Policy:** See the late submission policy here: http://www.cs.cmu.edu/~mgormley/courses/10418/syllabus.html
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code. Please follow instructions at the end of this PDF to correctly submit all your code to Gradescope.
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, please use the provided template. 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. Alternatively, submissions can be written in LaTeX. Each derivation/proof should be completed in the boxes provided. You are responsible for ensuring that your submission contains exactly the same number of pages and the same alignment as our PDF template. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader.
  - **Programming:** You will submit your code for programming questions on the homework to Gradescope (https://gradescope.com). After uploading your code, our grading scripts will autograde your assignment by running your program in a Docker container. When you are developing, check that the version number of the programming language environment (e.g. Python 3.10.4) and versions of permitted libraries (e.g. `numpy` 1.23.2 and `PyTorch` 1.12.1) match those used on Gradescope. You have 10 free Gradescope programming submissions. After 10 submissions, you will begin to lose points from your total programming score. We recommend debugging your implementation on your local machine (or the Linux servers) and making sure your code is running correctly first before submitting your code to Gradescope.
- **Materials:** The data and reference output that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Written Questions (23 points)

1 Background Reading (2 points)

A key outcome of this homework assignment is familiarizing yourself with two libraries: PyTorch for module-based automatic differentiation and optimization and Weights & Biases for logging/plotting. Towards that end, if you have never used these libraries before, you should expect to spend significant time reading the documentation for them in addition to attending recitation.

To emphasize that reading the documentation is a required step in this homework, you should complete the following readings (at least) before you begin.

- PyTorch Tutorial. At a bare minimum, you should read the Quickstart section before you begin. We recommend you also read the full collection of the Introduction to PyTorch, i.e. Learn the Basics || Quickstart || Tensors || Datasets & DataLoaders || Transforms || Build Model || Autograd || Optimization || Save & Load Model

  https://pytorch.org/tutorials/beginner/basics/intro.html

- Weights & Biases Tutorial. Please read the Quickstart.

  https://docs.wandb.ai/quickstart

1. (1 point) **Select One:** Did you read the PyTorch tutorial (at least the Quickstart)?

   ◯ Yes

   ◯ No

2. (1 point) **Select One:** Did you read the Weights & Biases Quickstart tutorial?

   ◯ Yes

   ◯ No

The following questions should be completed as you work through the programming portion of this assignment. In the following three sections, plot the metrics and report metrics values such that your deep neural network has converged. For your reference, the TAs got convergence with the default hyperparameters (batch_size = 64, learning_rate = $1e-3$, epochs = 20) in the starter code.

2 Plot the metrics for ConvEncoder (9 points)

1. (3 points) Average Validation Accuracy

Your Answer

2. (3 points) Average Test Accuracy

Your Answer

3. (3 points) Loss Curve

Your Answer

3 Plot the metrics for LSTMEncoder (6 points)

1. (2 points) Average Validation Accuracy

Your Answer

2. (2 points) Average Test Accuracy

Your Answer

3. (2 points) Loss Curve

Your Answer

4 Test Results (6 points)

1. (6 points) Report the tag-wise test accuracy

POS Tag	Test Accuracy (ConvEncoder)	Test Accuracy (LSTMEncoder)
NOUN		
VERB		
ADJ		
PRON		
PROPN		
ADV		

5 Collaboration Questions

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found here.

1. Did you receive any help whatsoever from anyone in solving this assignment? If so, include full details.
2. Did you give any help whatsoever to anyone in solving this assignment? If so, include full details.
3. Did you find or come across code that implements any part of this assignment? If so, include full details.

Your Answer

6 Programming (60 points)

6.1 Setting up your environment

Please do the following before beginning this programming assignment.

1. Install Python using Miniconda https://docs.conda.io/en/latest/miniconda.html
2. Install pytorch using Conda https://pytorch.org/
3. Intall the `tqdm` package with `pip install tqdm`
4. Create a free Weights & Biases account at https://wandb.ai. Then install the `wandb` package with `pip install wandb` followed by `wandb login`
5. Read the `README.md` file, and extract the `data/csv_data.tar.gz` file into the `data/` directory.

6.2 Problem Description

In this assignment, you will be implementing sequence labelling models to perform Part of Speech (POS) tagging. Given an input sentence $\mathbf{x} = [x_1, x_2, \ldots, x_T]$ where x_t are tokens, the task is to predict a sequence of POS tags $\mathbf{y} = [y_1, y_2, \ldots, y_T]$, where y_t is the POS tag corresponding to token x_t. Here, we have $y_t \in \mathcal{T}, \forall t$, where $\mathcal{T} = \{l_1, l_2, ...l_L\}$ is the set of POS tags.

We ask you to implement two models, one based on a one-dimensional Convolutional Neural Network (CNN), and another that uses a bidirectional LSTM. First, we construct a vocabulary of all the word tokens from the training data, and use this to convert input sentences into a sequence of token indices. Both models use a learnable embedding layer, that take in an input sequence $\mathbf{x} = [x_1, x_2, ..., x_T]$ of token indices and encode them into their corresponding embeddings $\mathbf{e} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_T]$, where each embedding is a fixed length vector $\mathbf{e}_t \in \mathbb{R}^d$ corresponds to a token index x_t. Then the CNN or the bidirectional LSTM is used to produce a sequence of representations $\mathbf{h} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T]$ corresponding to each input token. Each representation $\mathbf{h}_t$ is then passed through a feed-forward layer to predict the POS tag y_t corresponding to x_t.

6.3 Handling Data

The data we will use for this assignment comes from Universal Dependencies (UD)[1] which is a framework for consistent annotation of grammar across different human languages. More information on this dataset can be found here: . The dataset we will use is the English ParTUT data. The original data follows the CoNLL-U format[2]. However, we have preprocessed this data into a more convenient .csv format.

The code to handle the data is given to you and can be found in the `data_utils.py` file. The `load_conllu()` method returns a list of sentences and a list of their corresponding POS tags. The `Vocabulary` class converts the sentences to a numerical tensor representation. Similarly, the `Tags` class converts the POS tags to a numerical tensor representation.

6.4 Dataset Class for ParTUT

The `Dataset` class handles the processing of the input data and their corresponding labels. Please refer to the official Pytorch's Dataset Class documentation for more information: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html.

The `get_item()` method returns the sentence at index *ind* and its corresponding POS tag sequence. Please make sure to return the numericalized sentence and POS tag sequence as numpy arrays.

[1] https://universaldependencies.org
[2] https://universaldependencies.org/format.html

The `collate_fn` helps to batch the data. This method should return a list of sentence sequences as tensors and their corresponding POS tag sequences as tensors for one batch. In addition to this, you will pad the sentence and POS tags sequences in this method. Please read the comments in the code to understand how to correctly implement it.

6.5 Networks

6.5.1 1D CNN: `ConvEncoder`

The first model is a one dimensional Convolutional Neural Network. We ask you to implement a one-dimensional convolution operation with a flattened convolution kernel, using PyTorch operations - specifically by using padding, indexing, reshaping, the `nn.Linear` layer, and the `nn.GELU` layer, **without** using the `nn.Conv1d` implementation from the library.

The convolution model first uses an embedding layer, as specified in Section 6.2. Then we define a convolutional filter $\mathbf{W} \in \mathbb{R}^{wd \times k}$, $\mathbf{b} \in \mathbb{R}^k$ where d is the token embedding dimension, k is the output dimension for the convolution encoder, and w is the window size of the convolution filter (the number of input tokens it uses in generating an output). We assume that w, the filter size is odd. The convolution layer generates an output representation $\mathbf{h}_t$ corresponding to the input token embedding $\mathbf{e}_t$ by calculating $\mathbf{h}_t = \text{GELU}(\mathbf{W}^T\mathbf{e}'_t + \mathbf{b})$ over the flattened input window $\mathbf{e}'_t = [\mathbf{e}_{t-(w-1)/2}, ..., \mathbf{e}_{t+(w-1)/2}]$, st. $\mathbf{e}'_t \in \mathbb{R}^{wd}$. When implementing this, we use zero padding of size $(w-1)/2$ on both sides of the input to ensure that we can generate representations corresponding to tokens in the beginning and at the end.

We first ask you to implement `get_windows()`, a function that takes an input of size `(B, input_len, emb_size)` of padded inputs, iterates over the input tensor with the provided window size, and generates an `outputs` tensor, of shape `(B, num_windows, window_size, emb_size)` where `B` is the batch size, and the output tensor at `outputs[i][j]` contains the window of unflattened input embeddings from placing the filter at the j^{th} index from the start of the padded input tensor's i^{th} datapoint. These input windows are then flattened into a tensor of $\mathbf{e}'_t$s, which you are required to pass through the feed-forward layer that implements the convolution filter. After the convolution operation, use a GELU activation function (see `nn.GELU`) to generate the final representations $\mathbf{h}_t$.

> Recall that, although PyTorch implements a high-level module for convolution, the goal of this section is to familiarize you with the low-level modules and how they can be pieced together.

6.5.2 Bidirectional LSTM: `LSTMEncoder`

The second model is a two layer bidirectional LSTM. For this, you are allowed to use the `nn.LSTM` module from PyTorch. In this module, implement an embedding layer similar to the `ConvEncoder` and as described in Section 6.2. Then use nn.LSTM to generate a sequence of representations $\mathbf{h}_t$ corresponding to inputs $\mathbf{h}_t$ using two stacked LSTMs.

> Recall that, although you *could* implement an LSTM layer from scratch using low-level modules in PyTorch, it would likely be very inefficient. The goal of this section is to familiarize you with how to read PyTorch documentation and employ the existing flexible high-level module for LSTMs.

6.5.3 Feed Forward Neural Network: `MLP`

This is the output layer of the network. Please implement a feed-forward neural network with one hidden layer of the given size, with an output dimension L, where L is the number of POS tags (Section 6.2). Use a tanh non-linearity on the hidden layer.

6.5.4 The Full Model: `POSTagger`

This is a module we have implemented for you, that puts together the encoder and the MLP to obtain a POS tagger.

6.6 Training

Please complete the `train_one_epoch()` and `train()` methods so that the model can learn to perform the required task. In order to provide a demonstration of best practices, most of the code is implemented for this section. In order to learn model weights, we need to calculate the `nn.CrossEntropyLoss()`. This method also handles the logging of various metrics using wandb.ai. More information about wandb.ai can be found here: https://docs.wandb.ai. Please sign up at wandb.ai and refer to the recitation for a brief overview of this tool for logging. If you do not want to log any run, you can set the `use_wandb` flag to `False`.

6.7 Evaluation:

To evaluate the model on the dev (validation) dataset, we compute accuracy (already implemented). Most of the evaluate method is also implemented. Refer to the starter code to finish this method. In addition to the average accuracy, accuracy per POS tag has also been calculated and is logged at wandb.ai.

6.8 Submission Checklist

- **Finish the TODOs:** Please complete the TODOs in the starter code to finish the assignment.
- **Submit to Autolab:** Set the `use_wandb` flag to `False` and submit the code to the programming slot on Gradescope.
- **Plot the metrics:** Setting the `use_wandb` flag to `True`, plot the metrics on wandb and attach the screenshots in the PDF and submit the PDF to the written slot on Gradescope.