Assignment 1

COMP-599 and LING-782/484

Overview

The purpose of this assignment is to help you become familiar with building ML models from start to finish using Pytorch. You will be given a Natural Language Inference dataset with binary labels (1 for entailment, 0 for no entailment). You will have to handle various preprocessing steps (batching, shuffling, converting to tensor), then design and train various types of neural networks to accurately predict the labels. Once trained, you can evaluate your models on the validation split using the F1 score.

Due date: January 28 at 23:59 EST.

Natural Language Inference (NLI): In this task, you are given a sentence called the "premise", and you want to predict if another sentence, the "hypothesis", either *entails* or *does not entail*. Saying that the premise entails a hypothesis means that, if I read the premise, then I would infer that the hypothesis is true. For example, if my premise is "*Two women are embracing while holding to go packages*", then it would **entail** the hypothesis that "*Two women are holding packages*", but **does not entail** the hypothesis that "*The men are fighting outside a deli*".

Points breakdown: The assignment will be broken into the following parts:

1. Write functions for data processing, batching, text-to-vector embedding **(30 points)**
2. Design a baseline pytorch model, select an optimizer and train it **(50 points)**
3. Experiment with more sophisticated models **(20 points)**

Grading: We will use automatic grading via Gradescope. You will have to sign up using your McGill email, Student ID, the following course code: **97JN2M**. You will have to modify the provided *code.py* file before uploading it (make sure the name is correct). You can submit multiple times and your grade will be your final score.

Submission: Make sure you do not change the function names or parameters as they will be needed in order to automatically evaluate your code. For each question, you will be given function signatures for which you will have to fill in the blanks. If you have supplementary code (e.g. to test or train your model), please move them to the ***if __name__ == "__main__"*** scope (see at the end of the provided python file) or define them with new function names. This is important because if your file takes a long time to run because it executes code, then your code might not be correctly graded.

Detailed Instructions: The docstrings of each function you have to complete contain further details about what exactly you will need to implement, along with a description of all the

expected inputs and outputs. The details in the next pages are a higher-level overview of each question and are complementary to the docstrings.

Compute: It is possible to complete this assignment on your personal computer, even without access to a GPU. You will need to use Python 3.7 or Python 3.8. You will also need to install Pytorch by running ***pip3 install torch==1.10.****.
Moreover, you can also complete this assignment without any local setup while accelerating your training code with free GPU access; there are two options, but we recommend using Kaggle First, you can access McGill CS's open-gpus workstations via remote access (see SSH instructions). The second option is to use Kaggle, which allows you to share and run Python scripts in a preset data science environment. To start, you have to create an account, then select Copy & Edit on this script: COMP599 W20 A1 Starter code.Then, you have to select "Verify phone number" on the right side of the screen to enable the access to GPU. When you are happy with your Kaggle script, you can click on the "Save Version" button on the top right of the screen, and choose "Quick Save". To submit to Gradescope, you will have to click on the 3 vertical dots and select "Download Code", then rename the file to "code.py" before submitting.

Part 1

In this section, you will implement a series of functions that will be useful for converting the initial data into batches of torch tensors. To help you get started, you are given a few helper functions to load the data, apply tokenization and convert to indices (e.g. "hello" → 5, where 5 is the index where the embedding for "hello" is located). For more information on how to use them, look at the docstring or the starter code immediately after ***if __name__ == "__main__"***.

1.1 Batching, shuffling, iteration - 20 points

The first task will be to handle the process of creating batches, shuffling the training set, and being able to iterate through the entire dataset. The function will take as input a dictionary containing the "premise" (list of all premises), "hypothesis" (list of all hypotheses), and "label" (list of labels). Note that the dictionary contains the data for an entire split, whereas the loader enables access to a single batch (which might or might not be shuffled, and might have an arbitrary size) in an iterative fashion. Edit the function named ***build_loader.***

What we want ***build_loader*** to do is to return a function that, when called, gives us an iterator that generates one batch at the time. You should have the option to shuffle the inputs before starting. Do not use Pytorch's data loader.

1.2 Converting a batch into inputs - 10 points

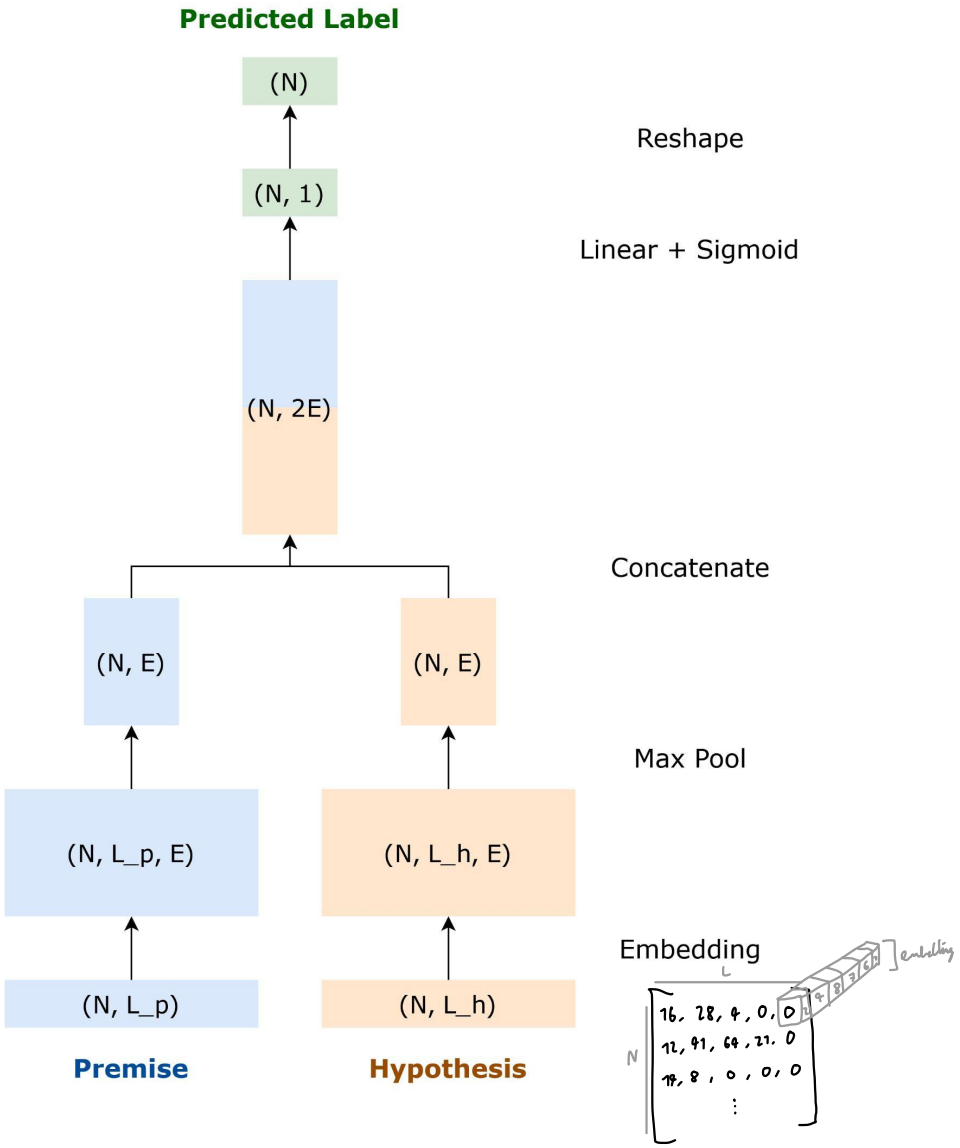
From above, we now have a batch that we can use when iterating through our loader. However, the batch is a nested list. We now want to convert that into a torch tensor of integers (representing the indices), which will require us to pad it with 0's. The function you design here will be applied to the premise, or to the hypothesis, but not to the label (you can handle that easily using existing torch functions). Edit the function named ***convert_to_tensors.***

Part 2

In this section, you will implement the full training procedure. First, you will need to design a very simple neural network. Afterward, you will need to handle various aspects of training: loss function, optimizer, evaluation metric and training loop.

2.1 Design a logistic model with embedding and pooling - 20 points

Use Pytorch's nn.Module to build a simple logistic regression model (in other words, a feed-forward layer with a single output between 0 and 1). Although the architecture will be simple, there are a few steps involved: activation, concatenation and pooling. Edit the function named ***max_pool*** and class named ***PooledLogisticRegression.*** Here's a diagram (N is batch size, L is sequence length, E is embedding dimension)



2.2 Choose an optimizer and a loss function - 5 points

There are many torch optimizers you can use from torch.optim; you can start with SGD or something else you prefer. Make sure you have applied the optimizer to your model. As for loss, you will have to implement binary-cross entropy using basic torch math functions (without using torch's BCE loss). Edit the functions named ***assign_optimizer*** and ***bce_loss.***

2.3 Forward and backward pass - 10 points

Implement a function that performs one step of the training process. It should take in your network, a batch, an optimizer, and make sure that the loss is back-propagated, the weights are updated by your optimizer, and the gradients are cleared at the end of the step. Edit the functions named ***forward_pass*** and ***backward_pass.***

2.4 Evaluation - 5 points

Implement F1 scoring from scratch with torch operations (do not use external F1 implementation) to evaluate the performance of your model on the validation split. Make sure to set the network on evaluation mode and not to backpropagate gradients. Edit the function named ***f1_score.***

2.5 Training loop - 10 points

Create a complete training loop using everything from above. For every epoch, it will train your network on each batch until you have made an entire pass through the training set; at that point, you will evaluate your model on both the training and validation sets (without computing the gradients!) and return the validation F1 score; you can print the score as you train. You might want to save the results to disk so you can review them later! Edit the functions named ***eval_run*** and ***train_loop.***

Part 3

Now that you have the full training procedure ready, you can experiment with different architectures! For each new architecture, you can train your model for a few epochs (it should be very fast if you run it on Kaggle or Colab).

3.1 Design 1-layer neural network with activation - 10 points

This follows the same idea as the logistic model, but before passing your pooled tensors to the logistic regression layer, it should go through a single feed-forward layer, and apply a ReLU activation. Edit the class named ***ShallowNeuralNetwork***.

3.2 Create deeper networks - 10 points

Modify your model to accept an arbitrary number of layers. The activation will be applied after every intermediate layer. Edit the class named ***DeepNeuralNetwork***.