LING572 HW10: PyTorch
Due: 11pm on March 15, 2023
Author: Dhwani Serai (dserai)

This assignment explores text classification using a version of the Deep Averaging Network from Iyyer et al 2015.[1] Through the assignment, you will:

Q1 (25 points): Read PyTorch tutorials at `https://pytorch.org/tutorials/beginner/basics/intro.html`, and answer the following questions:

(a) 2 pts: What is a tensor? What is the shape of a tensor?
Ans: A tensor is a data structure similar to arrays and matrices. In PyTorch, we use tensors to encode the inputs and outputs of a model, as well as the model's parameters. A tensor has 3 attributes i.e. shape, datatype and type of device on which it is stored. The shape of a tensor is a tuple of its dimensions. For example If the dimensions of a tensor are 2 by 3 then its shape will be (2,3)

(b) 2 pts: What is the difference between tensor multiplication and matrix multiplication between two tensors?
Ans: Tensor multiplication (torch.mul) does broadcasting on both tensors to ensure shape compatibility and then returns an element wise product between the two tensors.
Matrix multiplication (torch.matmul) performs broadcasting and then returns a matrix product between the two tensors.

(c) 5 pts: What does "optimizer" mean in PyTorch? Which package inside of PyTorch defines some optimizers? Specify the names of at least three optimizers implemented in that package.
Ans: Optimization is the process of adjusting model parameters to reduce model error in each training step. Optimizer in PyTorch defines the optimization algorithms used to define how optimization is performed.
torch.optim is a package implementing various optimizers. Some common optimizers that are implemented in this package are - Adam, RMSProp, SGD, XGBoost and AdaGrad.

(d) 3 pts: Suppose for a training instance, the system output is stored in a variable called *output*, and the gold standard is stored in a variable called *target*. The *nn package* in PyTorch has a function that calculates the mean squared error (MSE) loss. Write down the code that calls the function and calculate the loss. Your answer should be no more than two lines of code.
Ans:

```
import torch
import torch.nn as nn
loss = nn.MSELoss()
mse_loss = loss(output, target)
```

(e) 7 pts: When you define a new neural network using PyTorch, you define a class that specifies the structure of the network. The class should have at least two methods. What are they? What

[1] `https://www.aclweb.org/anthology/P15-1162/`

should each method specify? For instance, which method specifies the number of layers and the number of nodes in each layer? Which method specifies the activation function used in each layer?

Ans: The 2 prerequisite methods in a class Neural Network are-

1. init : to initialize the neural network layers and number of nodes in those layers

2. forward : to define the network structure which includes any activation function used in the Neural Network

(f) 6 pts: What does *frozen parameter* mean in PyTorch? Why do we need them? How can you mark some parameters as frozen ones?

Ans: When a parameter is such that we don't want to update it any longer and keep its values in the following training iteration, it will be a frozen parameter. Each model parameter has a grade flag requirement. When set to False, it will prevent backpropagation of the parameter, preserving or freezing its values in the layer.

Frozen parameters are useful in finetuning pretrained models.

Q2 (5 points): Install Anaconda. This is a necessary component for running the code for this assignment. We will use the conda environment made available for previous homeworks in this assignment, but you will need to install anaconda in order to use that environment. These are "free points". From your home directory, please execute the following steps in your home folder: **Note:** I had installed anaconda not miniconda The first two lines of `run_hw10.sh` show how to now activate the environment that we have supplied with all of the necessary libraries.

Q3 (15 points): Implement the forward pass of a LinearLayer and train a model.

- Find the '# TODO:' comment in `/dropbox/22-23/572/hw10/code/model.py`. Implement the .forward method there, following the instructions in the comment and the docstring.
- Run `python main.py --num_epochs 6 > q3.out`. Please fill out the information below:

Epoch num	Train loss	Dev loss
0	0.54577	0.39595
1	0.31534	0.33952
2	0.21518	0.33031
3	0.13682	0.37068
4	0.08616	0.43732
5	0.04479	0.59443

Test set accuracy of best model: 0.858543

Total runtime: 268.43

Q4 (15 points): L_2 regularization prevents over-fitting by penalizing large weight values. In particular, if $\mathcal{L}(\theta)$ is our loss function, L_2 regularization replaces that loss with

$$\mathcal{L}'(\theta) = \mathcal{L}(\theta) + \lambda \|\theta\|_2^2$$

where $\|\theta\|_2^2$ is the squared L^2 norm (i.e. the sum of the squares of all parameters in θ). You will:

- Add L_2 regularization in `main.py`. Search for '# TODO:' and find the one right above the line `L2 = 0.0`. Replace this with the squared L_2 norm of the model's parameters. Hint: `model.parameters()` returns an iterator over the parameters, which are each a `torch.tensor`. Note: you should use the `--L2` flag (stored in textttargs.L2) to only compute the regularization term when requested from the command-line.

- Run `python main.py --num_epochs 6 --L2 > q4.out`, training for 6 epochs with L_2 regularization. Please fill out the information below:

Epoch num	Train loss	Dev loss
0	0.54518	0.39327
1	0.31477	0.34058
2	0.21492	0.33060
3	0.13682	0.37147
4	0.08578	0.43322
5	0.04603	0.58601

  Test set accuracy of best model: 0.8585
  Total runtime: 381.65

Q5 (15 points): Another method for preventing over-fitting is early stopping. On this approach, we define a hyper-parameter patience (p), which is an integer. We then train for a large number of epochs, but if best_loss on the dev set is not improving for p epochs, we stop training. Here, best_loss is the lowest loss on the dev set among all the epochs so far.

- Implement early stopping. The final '# TODO:' in `main.py` occurs instructs you to implement this early stopping protocol in the main training loop. Note: you may need to edit code outside the immediate 'if' statement that the comment appears in.

- Run `python main.py --num_epochs 12 --patience 3 --L2 > q5.out` and fill out the information below (if early stopping stops your model before 12 epochs, you will have empty rows in this table, which is acceptable):

Epoch num	Train loss	Dev loss
0	0.54518	0.39327
1	0.31477	0.34058
2	0.21492	0.33060
3	0.13682	0.37147
4	0.08578	0.43322
5	0.04603	0.58601
6		
7		
8		
9		
10		
11		

Test set accuracy of best model: 0.8584
Total runtime: 369.84

Submission: Submit the following to Canvas:

- Your note file *readme.(txt | pdf)* that includes the tables and additional information above, and any notes that you want the TA to read.
- hw.tar.gz that includes all the files specified in dropbox/22-23/572/hw10/submit-file-list, plus any source code (and binary code) used by the shell scripts.
- Make sure that you run **check_hw10.sh** before submitting your hw.tar.gz.