CS 4501 Natural Language Processing (Fall 2024)
University of Virginia
Instructor: Yu Meng
Assignment prepared by TAs: Wenqian Ye & Xu Ouyang

Assignment 1: PyTorch Warm-Up (44 points)

Davis Wang

0. *Setup* (0 pts).
Before starting the assignment, add the following code at the beginning of your Python script to ensure reproducibility:

```
# Import required packages
import numpy as np
import torch

# Set random seeds
np.random.seed(42)
torch.manual_seed(42)
```

We recommend using Jupyter Notebook to run your code on your local machine. Refer to this Installation Guide to install it. To run it, you can refer to this Guide on your own machine or you can use Google Colab instead.

1. *Python Lists and NumPy Arrays* (5 pts).
Create a Python list of the first 10 square numbers $(1, 4, 9, ..., 100)$. Convert this list to a NumPy array and reshape it into a 2×5 matrix. Print out the matrix as a 2-dimensional array.

```
list = []
for x in range(1,11):
    list.append(x*x)
print(list)

arr = np.copy(list)
arr = np.reshape(arr, (2,5))
print(arr)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[[  1   4   9  16  25]
 [ 36  49  64  81 100]]
```

2. *PyTorch Tensors and Operations* (9 pts).
   Convert the NumPy array from Question 1 into a PyTorch tensor and perform the following operations. The following operations must be done using PyTorch interfaces.

   (a) Multiply all elements by 2 and print out the sum of all elements. (3 pts)

   (b) Create a new 5×2 tensor by transposing the original tensor. Print out the transposed tensor. (3 pts)

   (c) Perform matrix multiplication between the original tensor and the transposed tensor. Print out the result. (3 pts)

```
tensor = torch.from_numpy(arr)
tensor_a = tensor * 2
print(torch.sum(tensor_a))

tensor_b = torch.transpose(tensor, 0,1)
print(tensor_b)

result = torch.matmul(tensor, tensor_b)
print(result)

tensor(770)
tensor([[  1,  36],
        [  4,  49],
        [  9,  64],
        [ 16,  81],
        [ 25, 100]], dtype=torch.int32)
tensor([[  979,  4604],
        [ 4604, 24354]], dtype=torch.int32)
```

3. *IMDB dataset* (30 pts).

In this question, you will build a simple neural network to predict whether a movie review has positive or negative sentiment. We will use the IMDB movie review dataset. You should use the code provided in the Jupyter Notebook to download and preprocess the dataset.

(a) Implement a simple two-layer neural network using PyTorch's `nn.Module` for binary classification. The network should have:

- An input layer of `input_dim` features.
- One hidden layer with 64 neurons and `ReLU` activation.
- An output layer with 1 neuron and `Sigmoid` activation

Then initialize the model where `input_dim` equals the shape of the data in `X_train`, the Binary Entropy loss function, and the Adam optimizer with 0.001 learning rate. (10 pts)

```
import torch.optim as optim  # Add this import statement

# TODO: Define a simple 2-layer neural network model
class SimpleNN(nn.Module):
    def __init__(self, input_dim):
        super(SimpleNN, self).__init__()
        self.hidden = nn.Linear(input_dim, 64)
        self.relu = nn.ReLU()
        # ReLU activation function
        self.output = nn.Linear(64, 1)
        self.sigmoid = nn.Sigmoid()
        # Sigmoid activation function

    def forward(self, x):
        # Forward pass through the network
        x = self.hidden(x)
        x = self.relu(x)
        x = self.output(x)
        x = self.sigmoid(x)
        return x

input_dim = X_train.shape[1]
model = SimpleNN(input_dim)
criterion = nn.BCELoss()
# Binary entropy loss function
optimizer = optim.Adam(model.parameters(), lr=0.001)
# Adam optimizer -> 0.001 lr
```

(b) Train the neural network for 10 epochs, and print out the training loss and accuracy on the training set at the end of each epoch. (10 pts)

```
## https://pytorch.org/tutorials/beginner/introyt/trainingyt.html#the-training-

num_epochs = 10

for epoch in range(num_epochs):
    model.train()
    train_loss = 0.0
    correct = 0
    total = 0
    # TODO: Complete the training loop, update the training loss and accuracy
    for i, (texts, labels) in enumerate(train_loader):
        batch_size = texts.size(0)
        outputs = model(texts)
        optimizer.zero_grad()

        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * batch_size
        predicted = (outputs > 0.5).float()
        total += labels.size(0)
        correct += (predicted == labels).sum()

        train_loss /= len(train_loader)
        train_acc = correct / total

    print(f"Epoch {epoch+1}/{num_epochs}, "
          f"Train Loss: {train_loss:.4f}, "
          f"Train Acc: {train_acc:.4f}, ")
```

```
Epoch 1/10, Train Loss: 0.0147, Train Acc: 0.8634,
Epoch 2/10, Train Loss: 0.0072, Train Acc: 0.9292,
Epoch 3/10, Train Loss: 0.0107, Train Acc: 0.9492,
Epoch 4/10, Train Loss: 0.0056, Train Acc: 0.9628,
Epoch 5/10, Train Loss: 0.0027, Train Acc: 0.9734,
Epoch 6/10, Train Loss: 0.0002, Train Acc: 0.9854,
Epoch 7/10, Train Loss: 0.0002, Train Acc: 0.9867,
Epoch 8/10, Train Loss: 0.0014, Train Acc: 0.9952,
Epoch 9/10, Train Loss: 0.0003, Train Acc: 0.9988,
Epoch 10/10, Train Loss: 0.0001, Train Acc: 0.9998,
```

(c) Evaluate the test set and report the test accuracy. (10 pts)

```
# TODO: Evaluate the model on the test set
model.eval()
test_correct = 0
with torch.no_grad():
    for i, (texts, labels) in enumerate(test_loader):
        outputs = model(texts)
        predicted = (outputs > 0.5).float()
        test_correct += (predicted == labels).sum()

test_acc = test_correct / len(test_loader.dataset)
print(f"Test Accuracy: {test_acc:.4f}")

Test Accuracy: 0.8510
```