The lead TA for this assignment is Zae Myung Kim (kim01756@umn.edu). Please communicate with the lead TA via Slack or office hours. All questions MUST be discussed in the homework channel (i.e., #HW0). Questions through emails, Direct Messages, and other channels will not be answered.

The goal of this assignment is to make sure you get used to programming with PyTorch and implement a simple neural network based text classifier. Don't worry if you don't have deep learning programming experience. By following the steps below, you can train your own classifier from scratch.

First, please carefully read tutorial slides and notebooks on Scikit-learn based text classifier programming and PyTorch based text classifier programming, and try to run the same scripts on your local machine with Jupyter Notebooks in Google Colab. Now, let's build a simple text classifier using PyTorch. In the tutorials, you developed a multi-layer perceptron (MLP) based binary classifier for predicting whether a tweet is about a real disaster or not. In this homework, you will simply stack one more layer to your MLP and develop **a two-layer MLP text classifier** using Pytorch on **a new dataset**.

Step 1: Choose a dataset from TorchText.datasets

You can choose any dataset from PyTorch's torchtext. If you are using a laptop or local machine, choose a small dataset, such as IMDb (TRAIN/TEST: 25000/25000 samples) or SST2 (TRAIN/TEST: 67349/1821 samples). The TRAIN split is used to train your model, and TEST split is used to evaluate the trained model's performance. Your TEST set must not be used in any way during training. Below is an example script for loading the original IMDB dataset.

```
# import datasets
from torchtext.datasets import IMDB

train_iter = IMDB(split='train')

def tokenize(label, line):
    return line.split()

tokens = []
for label, line in train_iter:
    tokens += tokenize(label, line)
```

Step 2: Stack two layers of MLP

You implemented a one-layer MLP classifier in the tutorial. This homework simply requires adding one more layer to your one-layer MLP, as shown in the pseudo-code below. Note that you added a 100-dimensional intermediate layer between the embedding layer and the last class layer by setting num_layer = 100.

```
class MLP(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super().__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=True)
        num_layer = 100
        # self.fc = nn.Linear(embed_dim, num_class)
        self.fc1 = nn.Linear(embed_dim, num_layer)
        self.fc2 = nn.Linear(num_layer, num_class)

        # initialize the weights
        self.init_weights()
```

```
    def init_weights(self):
        ..

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        return self.fc2(self.fc1(embedded))
```

Step 3: Training and Analysis

Following the PyTorch tutorial, you can start training your model after loading a dataset and designing a classifier model. You can check out the following questions throughout different experiments:

- On the test set, what is the accuracy?
- What is the performance (i.e., accuracy on the test set) of a two-layer MLP compared to a single-layer MLP?
- What happens in the performance if you increase the number of dimensions of the intermediate layer from 100 to 200?
- Do you have a look at the test set samples that are incorrectly predicted by the model? Why are the errors occurring?

Deliverables

This assignment has no credit. Jupyter notebooks in Google Colab can be submitted but this is completely optional. Please submit your notebook file to Canvas by Jan 26, 11:59pm.

NOTE: This assignment will be the basis of your next assignment and class project that require more advanced Python/PyTorch programming, analysis, and deep learning knowledge.