CISC/CMPE 452/COGS 400 Assignment 2 - Backpropagation (15 points)

Please put your name and student id here

```
FirstName LastName, #12345678
```

- The notebook file has clearly marked blocks where you are expected to write code. Do not write or modify any code outside of these blocks.
- Make sure to restart and run all the cells from the beginning before submission. Do not clear out the outputs. You will only get credit for code that has been run.
- Mark will be deducted based on late policy (-1% of the course total marks per day after due date until the end date after which no assignments will be accepted)

Part 1 (9 points)

Build Model1 (7 points)

Use Pytorch to implement a three-layer Neural Network (input layer - hidden layer - output layer) and update the weights with backpropagation

- 1. Implement forward and calculate the output (1 point)
- 2. Calculate errors and loss (3 points)
- 3. Update the weights with backpropagation (1 points)
- 4. Predict function (1 point)
- 5. Activation function (Sigmoid function) (1 point)

Evaluator Function (1 point)

Implement the evaluator function with Pytorch or Numpy only

- Evaluation metrics include confusion matrix, accuracy, recall score, precision and F1 score

Train and Evaluate Model1 (1 point)

Train Model1 with customized hidden size, learning rate, number of iterations and batch size
Use the predict function to predict the labels with the test dataset
Evaluate the prediction results

- Evaluation metrics include confusion matrix, accuracy, recall score, precision and F1 score

Part 2 (6 points)

Use another machine learning framework (**scikit-learn, Tensorflow and Pytorch**) to build MLP e.g.

1. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)
2. https://www.tensorflow.org/api_docs/python/tf/keras/Sequential (https://www.tensorflow.org/api_docs/python/tf/keras/Sequential)
3. https://pytorch.org/tutorials/beginner/examples_nn/polynomial_nn.html#sphx-glr-beginner-examples-nn-polynomial-nn-py (https://pytorch.org/tutorials/beginner/examples_nn/polynomial_nn.html#sphx-glr-beginner-examples-nn-polynomial-nn-py)

Build Model2-1 (2 points)

Implement Model2-1 with the same hidden nodes and optimization function as the model in Part 1
Train and validate model. Use the best model on validation dataset to test on the test dataset

Train and Evaluate Model2-1 (1 point)

Evaluate the prediction results

- Evaluation metrics include confusion matrix, accuracy, recall score, precision and F1 score

Build Model2-2 (2 points)

Add one more hidden layer (2 hidden layers in total) to the model
Describe Model2-2 (number of hidden nodes)
Train and validate model. Use the best model on validation dataset to test on the test dataset

Train and Evaluate Model2-2 (1 point)

Evaluate the prediction results

- Evaluation metrics include confusion matrix, accuracy, recall score, precision and F1 score

In [ ]:
```
import torch
import matplotlib.pyplot as plt
from torchvision.datasets import MNIST
```

In [ ]:
```
# you can go to Edit - Notebook settings to select GPU under the Hardw
# check the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

In [ ]:
```
# build the dataset (train, validation and test)
def load_MNIST(n_val=10000, n_sample=1000, sample=False):
    n_val = n_val
    n_sample = n_sample
    train = MNIST(root = '.', train = True, download = True)
    test = MNIST(root = '.', train = False, download = True)

    # data preprocessing
    x_train, x_test = train.data/255, test.data/255
    x_train = x_train.reshape(x_train.shape[0], -1)
    x_test = x_test.reshape(x_test.shape[0], -1)
    y_train = torch.nn.functional.one_hot(train.targets)
    y_test = torch.nn.functional.one_hot(test.targets)

    data_dict = {}
    if sample:
        data_dict['x_train'] = x_train[:-n_val][:n_sample]
        data_dict['y_train'] = y_train[:-n_val][:n_sample]
        data_dict['x_val'] = x_train[-n_val:][:n_sample//10]
        data_dict['y_val'] = y_train[-n_val:][:n_sample//10]
        data_dict['x_test'] = x_test[:n_sample//10]
        data_dict['y_test'] = y_test[:n_sample//10]
    else:
        data_dict['x_train'] = x_train[:-n_val]
        data_dict['y_train'] = y_train[:-n_val]
        data_dict['x_val'] = x_train[-n_val:]
        data_dict['y_val'] = y_train[-n_val:]
        data_dict['x_test'] = x_test
        data_dict['y_test'] = y_test
    return data_dict
```

In [ ]:
```
# you can start with a small sample dataset by setting sample=True
data_dict = load_MNIST(sample=False)
print('Train data shape:', data_dict['x_train'].shape)
print('Train labels shape:', data_dict['y_train'].shape)
print('Validation data shape:', data_dict['x_val'].shape)
print('Validation labels shape:', data_dict['y_val'].shape)
print('Test data shape:', data_dict['x_test'].shape)
print('Test labels shape:', data_dict['y_test'].shape)
```

In [ ]:
```
# plot an example
plt.imshow(data_dict['x_train'][0].reshape(28, 28))
plt.title(data_dict['y_train'][0].argmax().item())
plt.show()
```

In [ ]:
```
def evaluator(y_test, y_pred):
    ###########################################################################
    # enter code here to implement the evaluation metrics including co
    # you can only use Numpy or Pytorch to implement the metrics


    ###########################################################################
```

Part 1

In [ ]:
```
class NN(object):
    def __init__(self, learning_rate, n_iters, batch_size, hidden_size
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.batch_size = batch_size
        self.hidden_size = hidden_size
        self.device = device
        self.dtype = dtype
        self.history = {}
        self.history['train_acc'], self.history['val_acc'], self.histo

    # 5. activation function
    def sigmoid(self, x):
        ###########################################################################
        # enter code here to implement the activation function


        ###########################################################################

    def train(self, x, y, x_val, y_val, verbose=1):
        n_train = x.shape[0]
        n_val = x_val.shape[0]
        input_size = x.shape[1]
```

```
        input_size = x.shape[1]
        num_classes = y.shape[1]

        # weight initialization
        self.W1 = torch.randn(input_size, self.hidden_size, dtype=self
        self.W2 = torch.randn(self.hidden_size, num_classes, dtype=sel

        # TODO: train the weights with the input data and labels
        for i in range(self.n_iters):
            loss = 0
            data = getBatch(x, y, self.batch_size)
            for x_batch, y_batch in data:
                # 1. forward
                ##########################################################
                # enter code here to calculate the hidden layer output
                hidden =
                output =
                ##########################################################

                # 2. error and loss
                ##########################################################
                # enter code here to calculate the output error, MSE l
                output_error =
                loss +=
                delta_output =
                delta_hidden =
                ##########################################################

                # 3. backward
                ##########################################################
                # enter code here to calculate delta weights and updat


                ##########################################################

            # calculate the accuracy and save the training history
            y_pred = self.predict(x)
            train_acc = torch.sum(torch.argmax(y, dim=1) == y_pred) /
            self.history['train_acc'].append(train_acc)
            self.history['loss'].append(loss)

            y_pred = self.predict(x_val)
            val_acc = torch.sum(torch.argmax(y_val, dim=1) == y_pred)
            self.history['val_acc'].append(val_acc)
            if verbose:
                print('epoch %d, loss %.4f, train acc %.3f, validation
                  % (i + 1, loss, train_acc, val_acc))

    # 4. predict function
    def predict(self, x):
        ##########################################################
```

```
        # enter code here to implement the predict function
        # TODO: use the trained weights to predict labels and return t
        # remember to use torch.argmax() to return the true labels


        ###################################################################
        return y_pred

def getBatch(x, y, batch_size):
    n_epoch = x.shape[0] // batch_size
    for i in range(n_epoch):
        x_batch = x[i * batch_size : (i+1) * batch_size]
        y_batch = y[i * batch_size : (i+1) * batch_size]
        yield x_batch, y_batch
    x_batch = x[(i+1) * batch_size:]
    y_batch = y[(i+1) * batch_size:]
    yield x_batch, y_batch
```

In [ ]:
```
###################################################################
# enter code here to train Model1
# TODO: set your desired hidden size, learning rate, number of iterati
# remeber to load the dataset to the device (e.g. data_dict['x_train']


###################################################################
```

In [ ]:
```
plt.plot(model.history['train_acc'], label='train_acc')
plt.plot(model.history['val_acc'], label='val_acc')
plt.legend()
plt.show()
```

In [ ]:
```
###################################################################
# enter code here to evaluate Model1 with test set
# TODO: use the trained Model1 to predict the labels of test set and e


###################################################################
```

In [ ]:
```

```

Part 2

Model2-1

In [ ]:
```
##########################################################################
# enter code here to implement Model2-1


##########################################################################
```

In [ ]:
```
##########################################################################
# enter code here to train Model2-1


##########################################################################
```

In [ ]:
```
##########################################################################
# enter code here to evaluate Model2-1


##########################################################################
```

Model2-2

In [ ]:
```
##########################################################################
# enter code here to implement Model2-2


##########################################################################
```

In [ ]:
```
##########################################################################
# enter code here to train Model2-2


##########################################################################
```

In [ ]:
```
##########################################################################
# enter code here to evaluate Model2-2


##########################################################################
```