Homework 3

University of Central Florida
Electrical and Computer Engineering

Current Topics in Machine Learning (EEL4815), Fall 2023

Issued: 1, Nov, 2023 **Due**: 15, Nov, 2023.

(Coding assignment: Neural Networks for Regression and Classification) In this exercise, we will implement Neural Networks using publicly available and commonly used datasets. For problems 2, 3, and 4, we will utilize the open source Keras and Tensorflow Python libraries.

1. **Problem 1:** In this problem, we will use a linear binary classifier (no activation function) with weight matrix $\mathbf{W}$ of size $M \times F$ to illustrate the use of the gradient descent algorithm in updating $\mathbf{W}$ at each iteration. We choose $F = 5$ and $M = 2$ to represent the number of features (per one example) and number of labels considered, respectively. This means that, for some observation vector $\mathbf{x}$ (of size F), the output is simply given by vector $\mathbf{y}(\mathbf{x}) = \mathbf{W}\mathbf{x} = (y_1(\mathbf{x}), y_2(\mathbf{x}))^\top$. We generate $N = 1000$ training data points (each of size F) by using Gaussian PDF with mean of 0 (0.1) and variance of 1 (1) for labels belonging to class 0 (1). The true labels are defined in the one-hot encoded vector $\mathbf{t}$ for data points $n = \{1, \ldots, 1000\}$ (or $n \in [N]$) as $\mathbf{t}_n = (0, 1)^\top$ for class 0 and $\mathbf{t}_n = (1, 0)^\top$ for class 1. If we consider all the data points, the error function is $e(\mathbf{W}) = \sum_{n\in[N]} e_n(\mathbf{W})$. Let the error function of one data point be $e_n(\mathbf{W}) = 0.5(y_1(\mathbf{x}_n) - t_{n1})^2 + 0.5(y_2(\mathbf{x}_n) - t_{n2})^2$, where $y_k(\mathbf{x}_n) = \sum_{i\in[F]} w_{ki}x_{ni}$, for $k = \{1, 2\}$. The gradient is then $\frac{\partial e_n(\mathbf{W})}{\partial w_{ki}} = (y_k(\mathbf{x}_n) - t_{nk})x_{ni}$. If the updated weight matrix is given by $\mathbf{W}'$, for each iteration, we use the following equation.

$$\begin{bmatrix} w'_{11} & \ldots & w'_{15} \\ w'_{21} & \ldots & w'_{25} \end{bmatrix} = \begin{bmatrix} w_{11} & \ldots & w_{15} \\ w_{21} & \ldots & w_{25} \end{bmatrix} - \eta \begin{bmatrix} \sum_{n\in[N]} \frac{\partial e_n(\mathbf{W})}{\partial w_{11}} & \cdots & \sum_{n\in[N]} \frac{\partial e_n(\mathbf{W})}{\partial w_{15}} \\ \sum_{n\in[N]} \frac{\partial e_n(\mathbf{W})}{\partial w_{21}} & \cdots & \sum_{n\in[N]} \frac{\partial e_n(\mathbf{W})}{\partial w_{25}} \end{bmatrix} \quad (1)$$

   Choose the learning rate (step size) $\eta = 0.00005$ and train for 50 iterations. Initialize the weight matrix $\mathbf{W}$ from a uniform PDF with bounds of $[-1, 1]$. Plot the training error (total MSE) and classification accuracy w.r.t. the number of iterations. The following code shows how to use the `numpy` library to initialize the weights.

```
import numpy as np
number_of_classes_M = 2
number_of_features_F = 5
initial_weights = np.random.uniform(low=-1, high=1,
    size=(number_of_classes_M, number_of_features_F))
```

2. **Problem 2:** In this problem, we utilize Tensorflow and Keras to train a 2-hidden layers regression ($M = 1$) NN using the Boston Housing price regression dataset [1]. In this dataset, we have $N = 404$ data points where each one consists of $F = 13$ features. The output of the network (scalar) is

$$y(\mathbf{x}) = \mathbf{W}^{(3)}\sigma^{(2)}\Big(\mathbf{W}^{(2)}\sigma^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}\Big) + \mathbf{b}^{(3)}\,. \quad (2)$$

   The superscripts are used to identify the layer number (1 and 2 represent the hidden layers and 3 represents the output layer). The required imports, loading the dataset, and the preprocessing of the the data are given in the following code.

[1] https://keras.io/api/datasets/boston_housing/

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
number_of_feature = 13
number_of_units_in_hidden_layer = 100
#### load the BOSTON regression dataset
(x_train, y_train), (x_test, y_test) =
    tf.keras.datasets.boston_housing.load_data(
    path="boston_housing.npz", test_split=0.2, seed=113)
#### pre-process the data
x_train -= np.mean(x_train)
x_train /= np.std(x_train)
x_test -= np.mean(x_test)
x_test /= np.std(x_test)
```

In order to construct the NN regression model, use the following code to define, specify, and add the layers. We use ReLU activation functions for $\sigma^{(1)}$ and $\sigma^{(2)}$.

```
NN_regression_model = tf.keras.Sequential()
hidden_layer_1 = layers.Dense(units=number_of_units_in_hidden_layer,
    activation='relu', input_shape=) # define layer
NN_regression_model.add(hidden_layer_1) # add layer
hidden_layer_2 = layers.Dense(units=number_of_units_in_hidden_layer,
    activation='relu') # define layer
NN_regression_model.add(hidden_layer_2) # add layer
output_layer = layers.Dense(units=1, activation=None) # define layer
    (units = 1 for regression)
NN_regression_model.add(output_layer) # add layer
```

Next, we specify the optimizer (with learning rate of 0.001), compile the model, and train using 500 epochs. Note that the batch size is one which means that, for every epoch, we use the whole dataset to train. For training, we use the `.fit` method. For evaluation, we use `.evaluate`.

```
NN_regression_model.fit(x_train, y_train, epochs=500, batch_size = 1)
train_error_mse,_ = NN_regression_model.evaluate(x_train, y_train)
test_error_mse ,_ = NN_regression_model.evaluate(x_test, y_test)
print("MSE on training data = {} ; MSE of testing data =
    {}".format(train_error_mse, test_error_mse))
```

The above code reports the testing error after 500 epochs. Write a code that report the testing error after every 10 epochs. Plot the training and testing error w.r.t. epochs. Compare and roughly indicate the over-fitting region in the plot.

3. **Problem 3:** In this problem, similar to Problem 2, we utilize Tensorflow and Keras to train a 1-hidden layer classification NN using the MNIST digits ($M = 10$) dataset [2]. In this dataset, we have $N = 60000$ training data points and 10000 examples for testing where each one consists of a gray scale image with $F = 28 * 28 = 784$ features (or pixels). The output of the network (vector of length

[2] https://keras.io/api/datasets/mnist/

M) is

$$\mathbf{y}(\mathbf{x}) = \sigma^{(2)}\left(\mathbf{W}^{(2)}\sigma^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}\right). \tag{3}$$

The predicted class of some example $\mathbf{x}$ is then given as $\text{argmax}_{k\in[10]} y_k(\mathbf{x})$. For the hidden layer, we choose 100 units (dimension of the output of the of the layer). The activation functions are ReLU for $\sigma^{(1)}$ and softmax for $\sigma^{(2)}$. This is done by selecting `activation='softmax'` when you construct the NN. To load and preprocess the dataset, use the following code.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
number_of_features = 28*28
#### load the MNIST dataset
(x_train, y_train),(x_test, y_test)=tf.keras.datasets.mnist.load_data()
x_train,x_test=x_train.astype('float32'),x_test.astype('float32')
# normalize and vectorize the data
x_train /= 255
x_test /= 255
x_train,x_test= x_train.reshape(60000,784,1),x_test.reshape(10000,784,1)
# one-hot encoding (OHE)
y_train_OHE = []
for label in y_train:
   all_zero_temp = 10*[0]
   all_zero_temp[label] = 1.0
   y_train_OHE.append(all_zero_temp)
y_train_OHE = np.asarray(y_train_OHE)
y_test_OHE = []
for label in y_test:
   all_zero_temp = 10*[0]
   all_zero_temp[label] = 1.0
   y_test_OHE.append(all_zero_temp)
y_test_OHE = np.asarray(y_test_OHE)
```

Modify the regression code of Problem 2 to construct the classification NN. When you compile the model, use `optimizer='adam' ,loss='categorical_crossentropy',metrics='accuracy'`. Train using 100 epochs and batch size of 100. Report the training and testing accuracy.

4. **Problem 4:** In this problem, we utilize Tensorflow and Keras to train a convolutional NN for classification using the Fashion MNIST ($M = 10$) dataset [3]. In this dataset, we have $N = 60000$ training data points and 10000 examples for testing where each one consists of a gray scale image with $F = 28 * 28 = 784$ features (or pixels). The dataset is loaded and pre-processed using the following code.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
number_of_features = 28*28
#### load the MNIST dataset
(x_train, y_train), (x_test,
   y_test)=tf.keras.datasets.fashion_mnist.load_data()
```

[3] https://keras.io/api/datasets/fashion_mnist/

```
#### building the input vector from the 28x28 pixels
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# normalizing the data
x_train /= 255
x_test /= 255
# one-hot encoding (OHE)
y_train_OHE = []
for label in y_train:
   all_zero_temp = 10*[0]
   all_zero_temp[label] = 1.0
   y_train_OHE.append(all_zero_temp)
y_train_OHE = np.asarray(y_train_OHE)
y_test_OHE = []
for label in y_test:
   all_zero_temp = 10*[0]
   all_zero_temp[label] = 1.0
   y_test_OHE.append(all_zero_temp)
y_test_OHE = np.asarray(y_test_OHE)
```

Here, we use (*i*) convolutional layer with 128 units and filter size of 3×3, and (*ii*) max pooling layer with a pool size of 4×4. Using keras, the layers can be added using the following code. The flatten layer is added here to vectorize the output tensor.

```
#### Construct the NN classification model
NN_classification_model = tf.keras.Sequential()
# define conv layer
Conv2d_layer = layers.Conv2D(128, kernel_size=(3,3), strides=(1,1),
    padding='valid', activation='relu', input_shape=(28,28,1))
NN_classification_model.add(Conv2d_layer)# add layer
# define max pooling layer
MaxPool_layer = layers.MaxPool2D(pool_size=(4,4))
NN_classification_model.add(MaxPool_layer)#add layer
# define flatten layer (reshaping from tensor to vector)
flatten_layer = layers.Flatten()
NN_classification_model.add(flatten_layer) #add flatten layer
```

Continue the construction of the NN by adding two dense layers (with ReLU activation and 100 units), and an output layer. Compile the model using the the same line of code as of Problem 3. Train using 100 epochs and batch size of 100, and report the training and testing accuracy.