OR 610 – Assignment 02

Abhishek Shambhu

Code with commented outputs:

```
"""
OR 610 Assignment 02 Solution
@author: Abhishek Shambhu
"""
####Question:

#1. Organic DL Without using a deep learning framework, code a one-hidden layer neural network model
#(multi-layer perceptron) for classification, i.e. the last layer is softmax and with
#cross-entropy loss function. Implement back-propagation. Use your code to classify MNIST.
#You can download MNIST data from http://yann.lecun.com/exdb/mnist/ and read it as a numpy array
#using the following function.

#import struct import numpy as np
#def read_idx(filename):
#   with open(filename, 'rb') as f:
#       zero, data_type, dims = struct.unpack('>HBB', f.read(4))
#       shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
#       return np.fromstring(f.read(), dtype=np.uint8).reshape(shape)

#Run a few numerical experiments with different non-linear activation functions,
#different number of neurons and different learning rates.

#2. Regularization Add l2 regularization to your model. Compare the accuracy and training speed.

####Solution:
#1.One hidden layer neural network without using a Deep Learning framework
import struct
import numpy as np
import time

def read_idx(filename):
    with open(filename, 'rb') as f:
        zero, data_type, dims = struct.unpack('>HBB', f.read(4))
        shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
        return np.frombuffer(f.read(), dtype=np.uint8).reshape(shape)

X_train = read_idx(r'D:\sem3\or610\HW2\train-images.idx3-ubyte')
y_train = read_idx(r'D:\sem3\or610\HW2\train-labels.idx1-ubyte')
X_test = read_idx(r'D:\sem3\or610\HW2\t10k-images.idx3-ubyte')
y_test = read_idx(r'D:\sem3\or610\HW2\t10k-labels.idx1-ubyte')
```

```
class MLP:
  """ Defining a One-Hidden Layer Neural Network Model (Multi-Layer Perceptron) for
classification
  Parameters:
    n: number of iterations
    l_rate: learning rate
    hidden_neurons: number of hidden neurons
    act_func: activation function used on the hidden layer
    encode: One hot encode target values to binary form if not
    size: size of each batch (number of samples in each batch)
  """
  def __init__(self, n=1000, l_rate=0.05, hidden_neurons=10, act_func = 'sigmoid', encode=True,
size=1):
    self.n = n
    self.l_rate = l_rate
    self.encode=encode
    self.hidden_neurons = hidden_neurons
    self.size = size
    self.act_func = act_func

  def hidden_layer(self, z):
    if self.act_func=='relu':
      return np.clip(z, 0, 250)
    else:
      return 1/(1+ np.exp(-z))

  def softmax(self, z):
    z = z- np.max(z)
    return np.exp(z)/np.sum(np.exp(z), axis=1).reshape(-1,1)

  def onehot(self, data):
    n_classes = np.unique(data)
    onehot_array = np.zeros((data.shape[0], np.max(data)+1))
    onehot_array[np.arange(data.shape[0]), data] = 1
    if n_classes.shape[0] > 2:
      return onehot_array[:,n_classes]
    else:
      return onehot_array[:,n_classes[0]]

  def forward_prop(self, X):
    Zh = np.dot(X, self.Wh) + self.bias_h
    Ah = self.hidden_layer(Zh)
    Zout = np.dot(Ah, self.Wout) + self.bias_out
    Aout = self.softmax(Zout)
    return Zh, Ah, Zout, Aout

  def backward_prop(self, Zh, Ah, Zout, Aout, X_train, y_train):
    output_error = Aout - y_train
    DJ_DWout = np.dot(Ah.T, output_error)
```

```
        if self.act_func == 'relu':
            hidden_deriv = Ah
            hidden_deriv[hidden_deriv > 0] = 1
            hidden_deriv[hidden_deriv <= 0] = 0
        else:
            hidden_deriv = Ah * (1-Ah)
        DJ_DWhid = np.dot(output_error, self.Wout.T) * hidden_deriv
        grad_Whid = np.dot(X_train.T, DJ_DWhid)
        grad_Bhid = np.sum(DJ_DWhid, axis=0)
        grad_Wout = DJ_DWout
        grad_Bout = np.sum(output_error, axis=0)
        self.Wh -= grad_Whid * self.l_rate
        self.bias_h -= grad_Bhid * self.l_rate
        self.Wout -= grad_Wout * self.l_rate
        self.bias_out -= grad_Bout * self.l_rate

    def fit(self, X_train, y_train, X_valid, y_valid):
        if self.encode:
            y_train_oh = self.onehot(y_train)
        n_classes = y_train_oh.shape[1]
        n_features = X_train.shape[1]
        self.Wh = np.random.randn(n_features, self.hidden_neurons)
        self.Wout = np.random.randn(self.hidden_neurons, n_classes)
        self.bias_h = np.zeros((1,self.hidden_neurons))
        self.bias_out = np.zeros((1,n_classes))
        m = y_train.shape[0]
        self.metrics = {'cost': [],'train_accuracy': [], 'valid_accuracy':[]}

        for i in range(self.n):
            shuffled_values = np.random.permutation(m)
            X_shuffled = X_train[shuffled_values]
            y_shuffled = y_train_oh[shuffled_values]
            for batch in range(0, m, self.size):
                x_batch = X_shuffled[batch:batch+self.size]
                y_batch = y_shuffled[batch:batch+self.size]
                Zh, Ah, Zout, Aout = self.forward_prop(x_batch)
                self.backward_prop(Zh, Ah, Zout, Aout, x_batch, y_batch)

            Zh, Ah, Zout, Aout = self.forward_prop(X_train)
            cost = self.cost_function(Aout, y_train_oh)
            train_predictions = self.predict(Aout)

            Zh, Ah, Zout, Aout = self.forward_prop(X_valid)
            valid_predictions = self.predict(Aout)

            train_accuracy = np.sum(train_predictions ==
y_train).astype(np.float)/train_predictions.shape[0]
            valid_accuracy = np.sum(valid_predictions ==
y_valid).astype(np.float)/valid_predictions.shape[0]
```

```
            if not (i+1)%20:
                print("Iterations: {}\t Training Accuracy: {:.3f}\t Validation Accuracy: {:.3f}".format(i+1,
train_accuracy, valid_accuracy))
            self.metrics['cost'].append(cost)
            self.metrics['train_accuracy'].append(train_accuracy)
            self.metrics['valid_accuracy'].append(valid_accuracy)
        return self

    def cost_function(self, Aout, y):
        return np.average(-y*np.log(Aout) - ((1-y)*np.log(1-Aout)))

    def predict(self, output):
        return np.argmax(output, axis=1)

#2.One hidden layer neural network without using a Deep Learning framework and using L2
Regularization
class MLP_l2:
    """ Defining a One-Hidden Layer Neural Network Model (Multi-Layer Perceptron) for
classification with l2 regularization
    Parameters:
        l2: l2 regularization rate
        n: number of iterations
        l_rate: learning rate
        hidden_neurons: number of hidden neurons
        act_func: activation function used on the hidden layer
        encode: One hot encode target values to binary form if not
        size: size of each batch (number of samples in each batch)
    """
    def __init__(self, l2=0.0, n=1000, l_rate=0.05, hidden_neurons=10, act_func = 'sigmoid',
encode=True, size=1):
        self.l2 = l2
        self.n = n
        self.l_rate = l_rate
        self.encode=encode
        self.hidden_neurons = hidden_neurons
        self.size = size
        self.act_func = act_func

    def hidden_layer(self, z):
        if self.act_func=='relu':
            return np.clip(z, 0, 250)
        else:
            return 1/(1+ np.exp(-z))

    def softmax(self, z):
        z = z- np.max(z)
        return np.exp(z)/np.sum(np.exp(z), axis=1).reshape(-1,1)

    def onehot(self, data):
        n_classes = np.unique(data)
```

```
        onehot_array = np.zeros((data.shape[0], np.max(data)+1))
        onehot_array[np.arange(data.shape[0]), data] = 1
        if n_classes.shape[0] > 2:
            return onehot_array[:,n_classes]
        else:
            return onehot_array[:,n_classes[0]]

    def forward_prop(self, X):
        Zh = np.dot(X, self.Wh) + self.bias_h
        Ah = self.hidden_layer(Zh)
        Zout = np.dot(Ah, self.Wout) + self.bias_out
        Aout = self.softmax(Zout)
        return Zh, Ah, Zout, Aout

    def backward_prop(self, Zh, Ah, Zout, Aout, X_train, y_train):
        output_error = Aout - y_train
        DJ_DWout = np.dot(Ah.T, output_error)
        if self.act_func == 'relu':
            hidden_deriv = Ah
            hidden_deriv[hidden_deriv > 0] = 1
            hidden_deriv[hidden_deriv <= 0] = 0
        else:
            hidden_deriv = Ah * (1-Ah)
        DJ_DWhid = np.dot(output_error, self.Wout.T) * hidden_deriv
        grad_Whid = np.dot(X_train.T, DJ_DWhid)
        grad_Bhid = np.sum(DJ_DWhid, axis=0)
        grad_Wout = DJ_DWout
        grad_Bout = np.sum(output_error, axis=0)
        l2_Whid = self.Wh * self.l2
        l2_Wout = self.Wout * self.l2
        self.Wh -= (grad_Whid + l2_Whid) * self.l_rate
        self.bias_h -= grad_Bhid * self.l_rate
        self.Wout -= (grad_Wout * self.l_rate) + l2_Wout
        self.bias_out -= grad_Bout * self.l_rate

    def fit(self, X_train, y_train, X_valid, y_valid):
        if self.encode:
            y_train_oh = self.onehot(y_train)
        n_classes = y_train_oh.shape[1]
        n_features = X_train.shape[1]
        self.Wh = np.random.randn(n_features, self.hidden_neurons)
        self.Wout = np.random.randn(self.hidden_neurons, n_classes)
        self.bias_h = np.zeros((1,self.hidden_neurons))
        self.bias_out = np.zeros((1,n_classes))
        m = y_train.shape[0]
        self.metrics = {'cost': [],'train_accuracy': [], 'valid_accuracy':[]}

        for i in range(self.n):
            rearrange_values = np.random.permutation(m)
            X_shuffled = X_train[rearrange_values]
```

```
            y_shuffled = y_train_oh[rearrange_values]
            for batch in range(0, m, self.size):
                x_batch = X_shuffled[batch:batch+self.size]
                y_batch = y_shuffled[batch:batch+self.size]
                Zh, Ah, Zout, Aout = self.forward_prop(x_batch)
                self.backward_prop(Zh, Ah, Zout, Aout, x_batch, y_batch)
            Zh, Ah, Zout, Aout = self.forward_prop(X_train)
            cost = self.cost_function(Aout, y_train_oh)
            train_predictions = self.predict(Aout)
            Zh, Ah, Zout, Aout = self.forward_prop(X_valid)
            valid_predictions = self.predict(Aout)
            train_accuracy = np.sum(train_predictions ==
y_train).astype(np.float)/train_predictions.shape[0]
            valid_accuracy = np.sum(valid_predictions ==
y_valid).astype(np.float)/valid_predictions.shape[0]

            if not (i+1)%20:
                print("Iterations: {}\t Training Accuracy: {:.3f}\t Validation Accuracy: {:.3f}".format(i+1,
train_accuracy, valid_accuracy))
            self.metrics['cost'].append(cost)
            self.metrics['train_accuracy'].append(train_accuracy)
            self.metrics['valid_accuracy'].append(valid_accuracy)
        return self

    def cost_function(self, Aout, y):
        m = y.shape[0]
        l2_cost = (self.l2/(2*m))*(np.sum(self.Wh**2)+np.sum(self.Wout**2))
        return np.average(-y*np.log(Aout) - ((1-y)*np.log(1-Aout))) + l2_cost

    def predict(self, output):
        return np.argmax(output, axis=1)

## Scaling to adjust weights
X_train = X_train.astype(np.float)/255.
X_test = X_test.astype(np.float)/255.

## Reshaping
X_train = X_train.reshape(-1,784)
X_test = X_test.reshape(-1,784)

# Running a NN without L2 Regularization and check accuracy and speed
#Sigmoid hidden activation, 100 hidden layers, 0.001 learning rate, 1000 batch size
model1 = MLP(n=100,hidden_neurons=100, size=1000, l_rate=0.001)
start = time.time()
model1.fit(X_train=X_train[:50000],
       y_train=y_train[:50000],
       X_valid=X_train[50000:],
       y_valid=y_train[50000:])
end =time.time()
runtime = end - start
```

```
print("Runtime for NN without L2 Regularization having Sigmoid hidden activation, 100 hidden layers, 0.001 learning rate, 1000 batch size:" + str(runtime) +"s")

'''
Iterations: 20   Training Accuracy: 0.914      Validation Accuracy: 0.917
Iterations: 40   Training Accuracy: 0.938      Validation Accuracy: 0.932
Iterations: 60   Training Accuracy: 0.950      Validation Accuracy: 0.939
Iterations: 80   Training Accuracy: 0.958      Validation Accuracy: 0.943
Iterations: 100  Training Accuracy: 0.964      Validation Accuracy: 0.946
Runtime for NN without L2 Regularization having Sigmoid hidden activation, 100 hidden layers, 0.001 learning rate, 1000 batch size:71.7282361984253s
'''

#ReLU Activation Function, 100 hidden layers, 0.001 learning rate, 1000 batch size
model2 = MLP(n=100,hidden_neurons=100, size=1000, l_rate=0.001, act_func='relu')
start = time.time()
model2.fit(X_train=X_train[:50000],
     y_train=y_train[:50000],
     X_valid=X_train[50000:],
     y_valid=y_train[50000:])
end =time.time()
runtime = end - start
print("Runtime for NN without L2 Regularization having ReLU Activation Function, 100 hidden layers, 0.001 learning rate, 1000 batch size:" + str(runtime) +"s")

'''
Iterations: 20   Training Accuracy: 0.906      Validation Accuracy: 0.904
Iterations: 40   Training Accuracy: 0.931      Validation Accuracy: 0.921
Iterations: 60   Training Accuracy: 0.947      Validation Accuracy: 0.936
Iterations: 80   Training Accuracy: 0.954      Validation Accuracy: 0.942
Iterations: 100  Training Accuracy: 0.959      Validation Accuracy: 0.943
Runtime for NN without L2 Regularization having ReLU Activation Function, 100 hidden layers, 0.001 learning rate, 1000 batch size:66.75671648979187s
'''

#Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 1000 batch size
model3 = MLP(n=100,hidden_neurons=30, size=1000, l_rate=0.001)
start = time.time()
model3.fit(X_train=X_train[:50000],
     y_train=y_train[:50000],
     X_valid=X_train[50000:],
     y_valid=y_train[50000:])
end =time.time()
runtime = end - start
print("Runtime for NN without L2 Regularization having Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 1000 batch size:" + str(runtime) +"s")

'''
Iterations: 20   Training Accuracy: 0.888      Validation Accuracy: 0.895
Iterations: 40   Training Accuracy: 0.913      Validation Accuracy: 0.913
```

Iterations: 60 Training Accuracy: 0.925 Validation Accuracy: 0.921
Iterations: 80 Training Accuracy: 0.934 Validation Accuracy: 0.926
Iterations: 100 Training Accuracy: 0.939 Validation Accuracy: 0.931
Runtime for NN without L2 Regularization having Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 1000 batch size:47.72391986846924s
'''

```
#ReLU Activation Function, 30 hidden layers, 0.001 learning rate, 1000 batch size
model4 = MLP(n=100,hidden_neurons=30, size=1000, l_rate=0.001, act_func='relu')
start = time.time()
model4.fit(X_train=X_train[:50000],
     y_train=y_train[:50000],
     X_valid=X_train[50000:],
     y_valid=y_train[50000:])
end =time.time()
runtime = end - start
print("Runtime for NN without L2 Regularization having ReLU Activation Function, 30 hidden layers, 0.001 learning rate, 1000 batch size:" + str(runtime) +"s")
```

'''
Iterations: 20 Training Accuracy: 0.867 Validation Accuracy: 0.876
Iterations: 40 Training Accuracy: 0.896 Validation Accuracy: 0.901
Iterations: 60 Training Accuracy: 0.915 Validation Accuracy: 0.920
Iterations: 80 Training Accuracy: 0.917 Validation Accuracy: 0.922
Iterations: 100 Training Accuracy: 0.927 Validation Accuracy: 0.929
Runtime for NN without L2 Regularization having ReLU Activation Function, 30 hidden layers, 0.001 learning rate, 1000 batch size:48.041693449020386s
'''

```
#Sigmoid hidden activation, 300 hidden layers, 0.0001 learning rate, 1000 batch size
model5 = MLP(n=100,hidden_neurons=300, size=1000, l_rate=0.0001)
start = time.time()
model3.fit(X_train=X_train[:50000],
     y_train=y_train[:50000],
     X_valid=X_train[50000:],
     y_valid=y_train[50000:])
end =time.time()
runtime = end - start
print("Runtime for NN without L2 Regularization having Sigmoid hidden activation, 300 hidden layers, 0.0001 learning rate, 1000 batch size:" + str(runtime) +"s")
```

'''
Iterations: 20 Training Accuracy: 0.886 Validation Accuracy: 0.891
Iterations: 40 Training Accuracy: 0.913 Validation Accuracy: 0.913
Iterations: 60 Training Accuracy: 0.925 Validation Accuracy: 0.923
Iterations: 80 Training Accuracy: 0.933 Validation Accuracy: 0.928
Iterations: 100 Training Accuracy: 0.938 Validation Accuracy: 0.933
Runtime for NN without L2 Regularization having Sigmoid hidden activation, 300 hidden layers, 0.0001 learning rate, 1000 batch size:42.71775031089783s
'''

```
#ReLU Activation Function, 300 hidden layers, 0.0001 learning rate, 1000 batch size
model6 = MLP(n=100,hidden_neurons=300, size=1000, l_rate=0.0001, act_func='relu')
start = time.time()
model4.fit(X_train=X_train[:50000],
     y_train=y_train[:50000],
     X_valid=X_train[50000:],
     y_valid=y_train[50000:])
end =time.time()
runtime = end - start
print("Runtime for NN without L2 Regularization having ReLU Activation Function, 300 hidden
layers, 0.0001 learning rate, 1000 batch size:" + str(runtime) +"s")

'''
Iterations: 20   Training Accuracy: 0.881      Validation Accuracy: 0.882
Iterations: 40   Training Accuracy: 0.917      Validation Accuracy: 0.920
Iterations: 60   Training Accuracy: 0.929      Validation Accuracy: 0.929
Iterations: 80   Training Accuracy: 0.935      Validation Accuracy: 0.934
Iterations: 100  Training Accuracy: 0.940      Validation Accuracy: 0.935
Runtime for NN without L2 Regularization having ReLU Activation Function, 300 hidden layers,
0.0001 learning rate, 1000 batch size:40.78611636161804s
'''

# Running a NN with L2 Regularization and check accuracy and speed
#Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 300 batch size, 0.1 l2
model7 = MLP_l2(n=100,hidden_neurons=30, size=300, l_rate=0.001, l2=0.1)
start = time.time()
model7.fit(X_train=X_train[:50000],
     y_train=y_train[:50000],
     X_valid=X_train[50000:],
     y_valid=y_train[50000:])
end =time.time()
runtime = end - start
print("Runtime for NN with L2 Regularization having Sigmoid hidden activation, 30 hidden
layers, 0.001 learning rate, 300 batch size:" + str(runtime) +"s")

'''
Iterations: 20   Training Accuracy: 0.615      Validation Accuracy: 0.625
Iterations: 40   Training Accuracy: 0.688      Validation Accuracy: 0.701
Iterations: 60   Training Accuracy: 0.752      Validation Accuracy: 0.768
Iterations: 80   Training Accuracy: 0.700      Validation Accuracy: 0.716
Iterations: 100  Training Accuracy: 0.762      Validation Accuracy: 0.776
Runtime for NN with L2 Regularization having Sigmoid hidden activation, 30 hidden layers,
0.001 learning rate, 300 batch size:42.96507692337036s
'''

#Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 300 batch size, 0.5 l2
model8 = MLP_l2(n=100,hidden_neurons=30, size=300, l_rate=0.001, l2=0.5)
start = time.time()
model8.fit(X_train=X_train[:50000],
```

```
    y_train=y_train[:50000],
    X_valid=X_train[50000:],
    y_valid=y_train[50000:])
end =time.time()
runtime = end - start
print("Runtime for NN with L2 Regularization having Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 300 batch size:" + str(runtime) +"s")

'''
Iterations: 20   Training Accuracy: 0.344      Validation Accuracy: 0.346
Iterations: 40   Training Accuracy: 0.291      Validation Accuracy: 0.283
Iterations: 60   Training Accuracy: 0.441      Validation Accuracy: 0.449
Iterations: 80   Training Accuracy: 0.496      Validation Accuracy: 0.512
Iterations: 100  Training Accuracy: 0.374      Validation Accuracy: 0.379
Runtime for NN with L2 Regularization having Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 300 batch size:43.55163931846619s
'''


'''
Findings:
  1. With lower L2 regularization rates model accuracy is better.
  2. Sigmoid and ReLU Activation function seems to have almost the same effect on the model.
'''
```

Summary of results:

Model	Parameters	Training Accuracy	Validation Accuracy	Time
Model1	Sigmoid hidden activation, 100 hidden layers, 0.001 learning rate, 1000 batch size	0.964	0.946	71.7282361984253s
Model2	ReLU Activation Function, 100 hidden layers, 0.001 learning rate, 1000 batch size	0.959	0.943	66.75671648979187s
Model3	Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 1000 batch size	0.939	0.931	47.72391986846924s
Model4	ReLU Activation Function, 30 hidden layers, 0.001 learning rate, 1000 batch size	0.927	0.929	48.041693449020386s
Model5	Sigmoid hidden activation, 300 hidden layers, 0.0001 learning rate, 1000 batch size	0.938	0.933	42.71775031089783s
Model6	ReLU Activation Function, 300 hidden layers, 0.0001 learning rate, 1000 batch size	0.940	0.935	40.78611636161804s
Model7	Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 300 batch size, 0.1 l2	0.762	0.776	42.96507692337036s
Model8	Sigmoid hidden activation, 30 hidden layers, 0.001 learning rate, 300 batch size, 0.5 l2	0.374	0.379	43.55163931846619s