Exercise - DL Tutorial 1

Please complete the following notebook and submit your solutions to manuel.milling@informatik.uni-augsburg.de

student name:

Solutions from exercise sheet 3 (class methods below).

In [1]:
```
#Equations from handout 3 and for are referred to as (3.X) and (4.X)

import numpy as np
#numpy random seed
np.random.seed(42)

trainx, trainy, testx, testy = np.load('mnist.npy', allow_pickle=True)
print("Trainx shape: {}".format(trainx.shape))
print("Trainy shape: {}".format(trainy.shape))
print("Testx shape:  {}".format(testx.shape))
print("Testy shape:  {}".format(testy.shape))

def sigmoid(X):
    return 1/(1 +np.exp(-X))

def softmax(X):
    #more stable
    eps = X.max()
    return np.exp(X + eps)/(np.sum(np.exp(X + eps), axis=1).reshape((X.shape[0],
1)))

def fcc_one_layer(X, W, b, activation):
    return activation(np.matmul(X, W) + b)

def cross_entropy(pred_logits, y):
    num_data_points = pred_logits.shape[0]
    correct_logits = pred_logits[np.arange(num_data_points),y]
    return np.mean(-np.log(correct_logits))

def accuracy(logits, labels):
    class_predictions = np.argmax(logits, axis=1)
    return np.mean(class_predictions == labels)
```

```
Trainx shape: (60000, 784)
Trainy shape: (60000,)
Testx shape:  (10000, 784)
Testy shape:  (10000,)
```

1. Implement the error of the last layer.

In [2]:
```
def delta_last_layer(h, y):
    """
    :param h: softmax activations of shape (num_examples, num_classes)
    :param y: correct labels of shape num_classes
    :return: delta of softmax
    """
    num_data_points = h.shape[0]
    # get H^n_mi for (4.31)
    correct_logits = h[np.arange(num_data_points), y]
    # (4.31) i no equal j
    h_i_neq_j = - np.reshape(correct_logits, (correct_logits.shape[0], 1)) * h
    # (4.31) i=j
    h_i_eq_j = correct_logits*(1- correct_logits)
    #replace the i=j terms in i not equal j matrix
    h_i_neq_j[np.arange(num_data_points),y] = h_i_eq_j
    #(4.30)
    h_i_neq_j = h_i_neq_j / np.reshape(correct_logits, (correct_logits.shape[0],
1))
    #transpose h --> delta shape
    return - np.transpose(h_i_neq_j)
```

In [3]:
```
def delta_last_layer_easy_approach(h, y):
    # create one hot vectors for every row
    one_hots = np.zeros((h.shape[0], h.shape[1]))
    one_hots[np.arange(h.shape[0]), y] = 1.0
    # (4.30) and (4.31) can be reshsaped
    return (h - one_hots).T
```

1. Implement the derivative of the sigmoid function in terms of the sigmoid function.

In [4]:
```
def del_sigmoid(h):
    """
    :param h: output of sigmoid function, i.e. h = \sigma(i)
    """
    #dh/dx = d\sigma(x)/dx = \sigma(x)(1-\sigma(x)) = h(1-h)
    return h * (1 - h)
```

1. Implement the backpropagation as a class method.
2. Implement the the optimisation step as a class method.

In [11]:
```
class fcc:
    def __init__(self, n_input, n_hidden1, n_hidden2, n_out):
        # Initialisation and Declaration of class variables
        self.W_i_h1 = np.random.randn(n_input, n_hidden1)
        self.b_h1 = np.random.randn(n_hidden1)
        self.W_h1_h2 = np.random.randn(n_hidden1, n_hidden2)
        self.b_h2 = np.random.randn(n_hidden2)
        self.W_h2_o = np.random.randn(n_hidden2, n_out)
        self.b_out = np.random.randn(n_out)
        # not necessary, but for better overview
        self.X = None
        self.h1 = None
        self.h2 = None
        self.out = None
        self.dW_i_h1 = None
        self.db_h1 = None
        self.dW_h1_h2 = None
        self.db_h2 = None
        self.dW_h2_o = None
        self.db_out = None

        #calculation of network parameters
        n_trainable_bias = self.b_h1.shape[0] + self.b_h2.shape[0] + self.b_out.sha
pe[0]
        n_trainable_weights = self.W_i_h1.shape[0] * self.W_i_h1.shape[1] + self.W_
h1_h2.shape[0] * self.W_h1_h2.shape[1] + self.W_h2_o.shape[0] * self.W_h2_o.shape
[1]
        print("Number of parameters: {}".format(n_trainable_bias + n_trainable_weig
hts))

    def forward_propagation(self, X):
        self.X = X
        # (3.4)
        self.h1 = fcc_one_layer(X, self.W_i_h1, self.b_h1, sigmoid)
        # (3.4)
        self.h2 = fcc_one_layer(self.h1, self.W_h1_h2, self.b_h2, sigmoid)
        # (3.4)
        self.out = fcc_one_layer(self.h2, self.W_h2_o, self.b_out, softmax)
        return self.out

    def backprop(self, y):
        self.num_train_ex = y.shape[0]
        self.delta_out = delta_last_layer_easy_approach(self.out, y)
        # (4.27)
        self.dW_h2_o = np.transpose(np.matmul(self.delta_out, self.h2))/self.num_tr
ain_ex
        # (4.28)
        self.db_out = np.mean(self.delta_out, axis=1)
        # (4.26)
        self.delta_h2 = np.matmul(self.W_h2_o, self.delta_out) * np.transpose(del_s
igmoid(self.h2))
        # (4.27)
        self.dW_h1_h2 = np.transpose(np.matmul(self.delta_h2, self.h1)) / self.num_
train_ex
        # (4.28)
        self.db_h2 = np.mean(self.delta_h2, axis=1)
        # (4.26)
        self.delta_h1 = np.matmul(self.W_h1_h2, self.delta_h2) * np.transpose(del_s
igmoid(self.h1))
        # (4.27)
        self.dW_i_h1 = np.transpose(np.matmul(self.delta_h1, self.X)) / self.num_tr
ain_ex
        # (4.28)
        self.db_h1 = np.mean(self.delta_h1, axis=1)
```

1. Implement the training routine.

In [15]:
```
learning_rate = 0.1
neural_net = fcc(784, 400, 400, 10)
```

```
Number of parameters: 478410
```

Normal Gradient Descent

In [13]:

```
#1000 trainingssteps
num_iterations = 1000
for i in range(num_iterations):
    # evaluate after each 100 steps
    if i % 100 == 0:
        print
("-----------------------------------------------------------------------------------
--------------------")
        print("Iteration:\t\t{}".format(i))
        logits = neural_net.forward_propagation(testx)
        print("Test Loss:\t\t{}".format(cross_entropy(logits, testy)))
        print("Test Accurcy:\t\t{}".format(accuracy(logits, testy)))
    logits = neural_net.forward_propagation(trainx)
    if i%100 == 0:
        print("Train Loss:\t\t{}".format(cross_entropy(logits, trainy)))
        print("Train Accuracy:\t\t{}".format(accuracy(logits, trainy)))
        print
("-----------------------------------------------------------------------------------
--------------------")
    neural_net.backprop(trainy)
    neural_net.gradient_step(learning_rate)
print
("-----------------------------------------------------------------------------------
--------------------")
print("Iteration:\t\t{}".format(i))
logits = neural_net.forward_propagation(testx)
print("Test Loss:\t\t{}".format(cross_entropy(logits, testy)))
print("Test Accurcy:\t\t{}".format(accuracy(logits, testy)))
logits = neural_net.forward_propagation(trainx)
print("Train Loss:\t\t{}".format(cross_entropy(logits, trainy)))
print("Train Accuracy:\t\t{}".format(accuracy(logits, trainy)))
```

```
-------------------------------------------------------------------------------
---------------------
Iteration:              0
Test Loss:              19.479279548133853
Test Accurcy:           0.1353
Train Loss:             19.653523403657122
Train Accuracy:         0.13226666666666667
-------------------------------------------------------------------------------
---------------------
-------------------------------------------------------------------------------
---------------------
Iteration:              100
Test Loss:              2.282769668031916
Test Accurcy:           0.5941
Train Loss:             2.4319569058890083
Train Accuracy:         0.57965
-------------------------------------------------------------------------------
---------------------
-------------------------------------------------------------------------------
---------------------
Iteration:              200
Test Loss:              1.573850466163723
Test Accurcy:           0.7007
Train Loss:             1.6520382064719452
Train Accuracy:         0.6910666666666667
-------------------------------------------------------------------------------
---------------------
-------------------------------------------------------------------------------
---------------------
Iteration:              300
Test Loss:              1.296911010188398
Test Accurcy:           0.7445
Train Loss:             1.3371711957554957
Train Accuracy:         0.7404666666666667
-------------------------------------------------------------------------------
---------------------
-------------------------------------------------------------------------------
---------------------
Iteration:              400
Test Loss:              1.142345153274019
Test Accurcy:           0.7731
Train Loss:             1.1521656169086645
Train Accuracy:         0.7698666666666667
-------------------------------------------------------------------------------
---------------------
-------------------------------------------------------------------------------
---------------------
Iteration:              500
Test Loss:              1.0402612586580389
Test Accurcy:           0.7876
Train Loss:             1.0261024142379838
Train Accuracy:         0.7897
-------------------------------------------------------------------------------
---------------------
-------------------------------------------------------------------------------
---------------------
Iteration:              600
Test Loss:              0.9656370013641344
Test Accurcy:           0.8009
Train Loss:             0.9324061314836917
Train Accuracy:         0.8044166666666667
-------------------------------------------------------------------------------
---------------------
-------------------------------------------------------------------------------
```

Stochastic Gradient Descent

In [16]:
```
#rerun initialisation of neural_network before executing

batch_size = 64
permutation = np.arange(trainx.shape[0])
epochs = 10
for i in range(epochs):
    print
("---------------------------------------------------------------------------------
--------------------")
    print("Epoch:\t\t{}".format(i))
    logits = neural_net.forward_propagation(testx)
    print("Test Loss:\t\t{}".format(cross_entropy(logits, testy)))
    print("Test Accurcy:\t\t{}".format(accuracy(logits, testy)))
    print
("---------------------------------------------------------------------------------
--------------------")
    #create new permuatation of trainings examples
    np.random.shuffle(permutation)
    #loop over epoch (= one permutation of all data)
    for j in range(int(trainx.shape[0]/batch_size)):
        #take one minibatch
        batch = permutation[j*batch_size:(j+1) * batch_size]
        trainx_batch = trainx[batch]
        trainy_batch = trainy[batch]
        logits = neural_net.forward_propagation(trainx_batch)
        # print every 100 training steps
        if j%100 == 0:
            print("Train Loss:\t\t{}".format(cross_entropy(logits, trainy_batch)))
            print("Train Accuracy:\t\t{}".format(accuracy(logits, trainy_batch)))
        neural_net.backprop(trainy_batch)
        neural_net.gradient_step(learning_rate)
logits = neural_net.forward_propagation(testx)
print
("---------------------------------------------------------------------------------
--------------------")
print("Final:\t\t")
print("Test Loss:\t\t{}".format(cross_entropy(logits, testy)))
print("Train Accuracy:\t\t{}".format(accuracy(logits, testy)))
```

```
--------------------------------------------------------------------------------
---------------------
Epoch:          0
Test Loss:              20.258685506517264
Test Accurcy:           0.0926
--------------------------------------------------------------------------------
---------------------
Train Loss:             19.821905152851123
Train Accuracy:         0.109375
Train Loss:             1.8949693999235606
Train Accuracy:         0.5625
Train Loss:             2.0527701690070286
Train Accuracy:         0.671875
Train Loss:             1.6121803157171737
Train Accuracy:         0.65625
Train Loss:             2.077289663492683
Train Accuracy:         0.71875
Train Loss:             1.453578726376255
Train Accuracy:         0.71875
Train Loss:             1.3055405866063823
Train Accuracy:         0.640625
Train Loss:             1.0969084533218267
Train Accuracy:         0.828125
Train Loss:             0.7339847306076936
Train Accuracy:         0.796875
Train Loss:             0.717445025227063
Train Accuracy:         0.828125
--------------------------------------------------------------------------------
---------------------
Epoch:          1
Test Loss:              0.829246421718839
Test Accurcy:           0.8072
--------------------------------------------------------------------------------
---------------------
Train Loss:             0.7272308116995612
Train Accuracy:         0.859375
Train Loss:             0.4546252790733857
Train Accuracy:         0.875
Train Loss:             0.41340715768929903
Train Accuracy:         0.890625
Train Loss:             0.5671620821361496
Train Accuracy:         0.875
Train Loss:             0.607704523731818
Train Accuracy:         0.859375
Train Loss:             0.9237634688270586
Train Accuracy:         0.828125
Train Loss:             0.6420787375359898
Train Accuracy:         0.828125
Train Loss:             0.6920089752589389
Train Accuracy:         0.796875
Train Loss:             0.5326557310728961
Train Accuracy:         0.875
Train Loss:             0.5589444050035213
Train Accuracy:         0.859375
--------------------------------------------------------------------------------
---------------------
Epoch:          2
Test Loss:              0.6006060028525041
Test Accurcy:           0.8547
--------------------------------------------------------------------------------
---------------------
Train Loss:             0.21403026054300162
Train Accuracy:         0.921875
Train Loss:             0.5835133616491388
```

In [ ]: