MNIST

November 10, 2019

[1]:
```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

digits = load_digits()
y = np.matrix(digits.target).T
X = np.matrix(digits.data)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)

M = X_train.shape[0]
N = X_train.shape[1]

print("Image at index 3: ")
import matplotlib.pyplot as plt
plt.matshow(digits.images[3])
plt.show()

attributes = list(digits)
print("Attributes of data: ", attributes)
print("Label of an image: ", digits.target[3])
```

```
Image at index 3:

<Figure size 480x480 with 1 Axes>


Attributes of data:  ['data', 'target', 'target_names', 'images', 'DESCR']
Label of an image:  3
```

[2]:
```
# Normalize each input feature

def normalize(X):
    return X/np.max(X)

XX = normalize(X_train)
```

```
Xtest = normalize(X_test)
# Let's start with a 3-layer network with sigmoid activation functions,
# 6 units in layer 1, and 5 units in layer 2.

h2 = 150
h1 = 200
W = [[], np.random.normal(0,0.1,[N,h1]),
         np.random.normal(0,0.1,[h1,h2]),
         np.random.normal(0,0.1,[h2,10])]
b = [[], np.random.normal(0,0.1,[h1,1]),
         np.random.normal(0,0.1,[h2,1]),
         np.random.normal(0,0.1,[10,1])]
L = len(W)-1

def RELU(x):
    return np.maximum(0,x)

def relu_der(z):
    return 1.*(z>0)

def softmax(x):
    return np.exp(x)/np.sum(np.exp(x))

def softmax_der(x):
    prob = softmax(x)
#     print(np.shape(prob))
    r = np.multiply(prob, (1-prob))
    return r

def oneHotEncode(y,softmaxClasses):
        yactual = np.matrix(np.zeros(softmaxClasses)).T
        yactual[y] = 1
        return yactual

def crossEntropy(ypred,y,softmaxClasses):
    #one hot encode
    yactual = oneHotEncode(y,softmaxClasses)
    l = np.log(ypred)
    return np.sum(-1 * l.T * yactual)


def delta_cross_entropy(y, X):
          m = y.shape[0]
          grad = softmax(X)
          idx = np.where(grad == y.T*grad)
          grad[idx] = grad[idx] - 1
          grad = grad/m
```

```
        return grad
def act(z):
    return 1/(1+np.exp(-z))

def actder(z):
    az = act(z)
    prod = np.multiply(az,1-az)
    return prod

def ff(x,W,b):
    L = len(W)-1
    a = x
    for l in range(1,L+1):
        z = W[l].T*a+b[l]
        a = act(z)
    return a

def loss(y,yhat):
    return -((1-y) * np.log(1-yhat) + y * np.log(yhat))

# Use mini-batch size 1
# M = 6
alpha = 0.002
max_iter = 500
import math
error = math.inf
for iter in range(0, max_iter):
    loss_this_iter = 0
    prev_iter_loss = error
    order = np.random.permutation(M)
    for i in range(0,M):

        # Grab the pattern order[i]

        x_this = XX[order[i],:].T
        y_this = y_train[order[i],0]

        # Feed forward step

        a = [x_this]
        z = [[]]
        delta = [[]]
        dW = [[]]
        db = [[]]
        for l in range(1,L+1):
            z.append(W[l].T*a[l-1]+b[l])
            if l != L:
```

```
                a.append(RELU(z[l]))
            if l == L:
                a.append(softmax(z[l]))
            # Just to give arrays the right shape for the backprop step
            delta.append([]); dW.append([]); db.append([])

        loss_this_pattern = crossEntropy( a[L], y_this, 10)
#         print(a[L])
        loss_this_iter = loss_this_iter + loss_this_pattern

        # Backprop step

        delta[L] = delta_cross_entropy(oneHotEncode(y_this, 10), a[L])
#         print(np.sum(delta[L]))
        for l in range(L,0,-1):
            db[l] = delta[l].copy()
            dW[l] = a[l-1] * delta[l].T
            if l > 1:
                if l == L:
#                     WxD = W[l]*delta[l]
                    delta[l-1] = np.multiply(softmax_der(z[l-1]), W[l]*delta[l]␣
 ↪)
#                     print("softmax delt: ", np.sum(delta[l-1]))
                else:
                    delta[l-1] = np.multiply(relu_der(z[l-1]), W[l] * delta[l])
#                     print("relu delt: ", np.sum(delta[l-1]))

        # Check delta calculation

        if False:
            print('Target: %f' % y_this)
            print('y_hat: %f' % a[L][0,0])
            print(db)
            y_pred = ff(x_this,W,b)
            diff = 1e-3
            W[1][10,0] = W[1][10,0] + diff
            y_pred_db = ff(x_this,W,b)
            L1 = loss(y_this,y_pred)
            L2 = loss(y_this,y_pred_db)
            db_finite_difference = (L2-L1)/diff
            print('Original out %f, perturbed out %f' %
                 (y_pred[0,0], y_pred_db[0,0]))
            print('Theoretical dW %f, calculated db %f' %
                  (dW[1][10,0], db_finite_difference[0,0]))

        for l in range(1,L+1):
            W[l] = W[l] - alpha * dW[l]
```

```
            b[l] = b[l] - alpha * db[l]

    print('Iteration %d loss %f' % (iter, loss_this_iter))
    error = loss_this_iter
    if loss_this_iter > prev_iter_loss:
        break
```

```
Iteration 0 loss 2888.894581
Iteration 1 loss 2857.868028
Iteration 2 loss 2824.410881
Iteration 3 loss 2795.267401
Iteration 4 loss 2763.295420
Iteration 5 loss 2737.856998
Iteration 6 loss 2707.792653
Iteration 7 loss 2680.391120
Iteration 8 loss 2653.603659
Iteration 9 loss 2627.867402
Iteration 10 loss 2600.422070
Iteration 11 loss 2574.815153
Iteration 12 loss 2549.687100
Iteration 13 loss 2523.776674
Iteration 14 loss 2498.766101
Iteration 15 loss 2475.671592
Iteration 16 loss 2451.858567
Iteration 17 loss 2427.981073
Iteration 18 loss 2405.024751
Iteration 19 loss 2381.743802
Iteration 20 loss 2356.892950
Iteration 21 loss 2336.937758
Iteration 22 loss 2313.959935
Iteration 23 loss 2292.386842
Iteration 24 loss 2270.240024
Iteration 25 loss 2250.118815
Iteration 26 loss 2228.919814
Iteration 27 loss 2207.879187
Iteration 28 loss 2184.679905
Iteration 29 loss 2163.846567
Iteration 30 loss 2142.241244
Iteration 31 loss 2121.068703
Iteration 32 loss 2103.646203
Iteration 33 loss 2083.164993
Iteration 34 loss 2061.038129
Iteration 35 loss 2043.435578
Iteration 36 loss 2022.228947
Iteration 37 loss 2003.369814
Iteration 38 loss 1983.829430
Iteration 39 loss 1965.596243
```

```
Iteration 40 loss 1945.488310
Iteration 41 loss 1929.906530
Iteration 42 loss 1907.048332
Iteration 43 loss 1891.284569
Iteration 44 loss 1869.399107
Iteration 45 loss 1850.935522
Iteration 46 loss 1832.884616
Iteration 47 loss 1815.505535
Iteration 48 loss 1796.123962
Iteration 49 loss 1779.748603
Iteration 50 loss 1762.211274
Iteration 51 loss 1742.482447
Iteration 52 loss 1724.294438
Iteration 53 loss 1708.568628
Iteration 54 loss 1690.693490
Iteration 55 loss 1675.031876
Iteration 56 loss 1657.877028
Iteration 57 loss 1640.123110
Iteration 58 loss 1623.002599
Iteration 59 loss 1601.880853
Iteration 60 loss 1589.618612
Iteration 61 loss 1569.910785
Iteration 62 loss 1555.511603
Iteration 63 loss 1540.167045
Iteration 64 loss 1518.989202
Iteration 65 loss 1505.414430
Iteration 66 loss 1492.650529
Iteration 67 loss 1474.910304
Iteration 68 loss 1459.185362
Iteration 69 loss 1447.303800
Iteration 70 loss 1428.402029
Iteration 71 loss 1412.966393
Iteration 72 loss 1397.220882
Iteration 73 loss 1384.565088
Iteration 74 loss 1368.215173
Iteration 75 loss 1355.704202
Iteration 76 loss 1339.154805
Iteration 77 loss 1323.487313
Iteration 78 loss 1311.786533
Iteration 79 loss 1299.614259
Iteration 80 loss 1285.198274
Iteration 81 loss 1268.429926
Iteration 82 loss 1254.166155
Iteration 83 loss 1242.413512
Iteration 84 loss 1232.473296
Iteration 85 loss 1219.436916
Iteration 86 loss 1203.002408
Iteration 87 loss 1194.134613
```

```
Iteration 88 loss 1176.508261
Iteration 89 loss 1168.941337
Iteration 90 loss 1158.888344
Iteration 91 loss 1140.735076
Iteration 92 loss 1134.299937
Iteration 93 loss 1117.969341
Iteration 94 loss 1113.140167
Iteration 95 loss 1097.458912
Iteration 96 loss 1085.614609
Iteration 97 loss 1074.592261
Iteration 98 loss 1066.681500
Iteration 99 loss 1051.606128
Iteration 100 loss 1044.405852
Iteration 101 loss 1037.387377
Iteration 102 loss 1029.832240
Iteration 103 loss 1016.869086
Iteration 104 loss 1009.307888
Iteration 105 loss 997.637064
Iteration 106 loss 985.720728
Iteration 107 loss 980.907363
Iteration 108 loss 977.741630
Iteration 109 loss 962.341443
Iteration 110 loss 958.829040
Iteration 111 loss 951.679000
Iteration 112 loss 955.495738
```

[3]:
```
m = Xtest.shape[0]
order = np.random.permutation(m)
truePositives = []
for i in range(0,m):

        # Grab the pattern order[i]

        x_this = Xtest[order[i],:].T
        y_this = y_test[order[i],0]

        # Feed forward step

        a = [x_this]
        z = [[]]
        delta = [[]]
        dW = [[]]
        db = [[]]

        for l in range(1,L+1):
            z.append(W[l].T*a[l-1]+b[l])
            if l != L:
```

```
                a.append(RELU(z[l]))
            if l == L:
                a.append(softmax(z[l]))
            # Just to give arrays the right shape for the backprop step
            delta.append([]); dW.append([]); db.append([])
#         highest = np.where(a[L]>=0.5)
        highest = np.argmax(a[L])
#         print(np.argmax(a[L]))
#         print(highest, "--- ", y_this)
        if highest == y_this:
            truePositives.append(True)

accuracy = 100*len(truePositives)/ Xtest.shape[0]
print("Accuracy: ", accuracy)
```

```
Accuracy:  70.53872053872054
```