fc3-Copy1

November 10, 2019

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

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=0.2,␣
 ↪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.
# h8 = 32
# h7 = 64
# h6 = 128
# h5 = 256
# h4 = 512
# h3 = 100
h2 = 100
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,h3]),
#          np.random.normal(0,0.1,[h3,h4]),
#          np.random.normal(0,0.1,[h4,h5]),
#          np.random.normal(0,0.1,[h5,h6]),
#          np.random.normal(0,0.1,[h6,h7]),
#          np.random.normal(0,0.1,[h7,h8]),
         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,[h3,1]),
#          np.random.normal(0,0.1,[h4,1]),
#          np.random.normal(0,0.1,[h5,1]),
#          np.random.normal(0,0.1,[h6,1]),
#          np.random.normal(0,0.1,[h7,1]),
#          np.random.normal(0,0.1,[h8,1]),
         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 = 0.005
max_iter = 1000
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)
        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 3233.260722
Iteration 1 loss 3177.695549
Iteration 2 loss 3121.086560
Iteration 3 loss 3065.830675
Iteration 4 loss 3011.785705
Iteration 5 loss 2958.022377
Iteration 6 loss 2903.018927
Iteration 7 loss 2849.027452
Iteration 8 loss 2794.853581
Iteration 9 loss 2740.362266
Iteration 10 loss 2687.069117
Iteration 11 loss 2632.491434
Iteration 12 loss 2576.869651
Iteration 13 loss 2522.956501
Iteration 14 loss 2467.363896
Iteration 15 loss 2412.092453
Iteration 16 loss 2356.072357
Iteration 17 loss 2300.378879
Iteration 18 loss 2244.063832
Iteration 19 loss 2189.353265
Iteration 20 loss 2130.352306
Iteration 21 loss 2073.266140
Iteration 22 loss 2019.484394
Iteration 23 loss 1962.257574
Iteration 24 loss 1909.661359
Iteration 25 loss 1853.820792
Iteration 26 loss 1793.633023
Iteration 27 loss 1747.804592
Iteration 28 loss 1696.323988
Iteration 29 loss 1644.992785
Iteration 30 loss 1599.091125
Iteration 31 loss 1540.306476
Iteration 32 loss 1502.852345
Iteration 33 loss 1449.703929
Iteration 34 loss 1425.160026
Iteration 35 loss 1388.767996
Iteration 36 loss 1342.895030
Iteration 37 loss 1315.185619
Iteration 38 loss 1303.410381
Iteration 39 loss 1280.681211
Iteration 40 loss 1271.697011
Iteration 41 loss 1268.699756
Iteration 42 loss 1293.721013
```

[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)
```

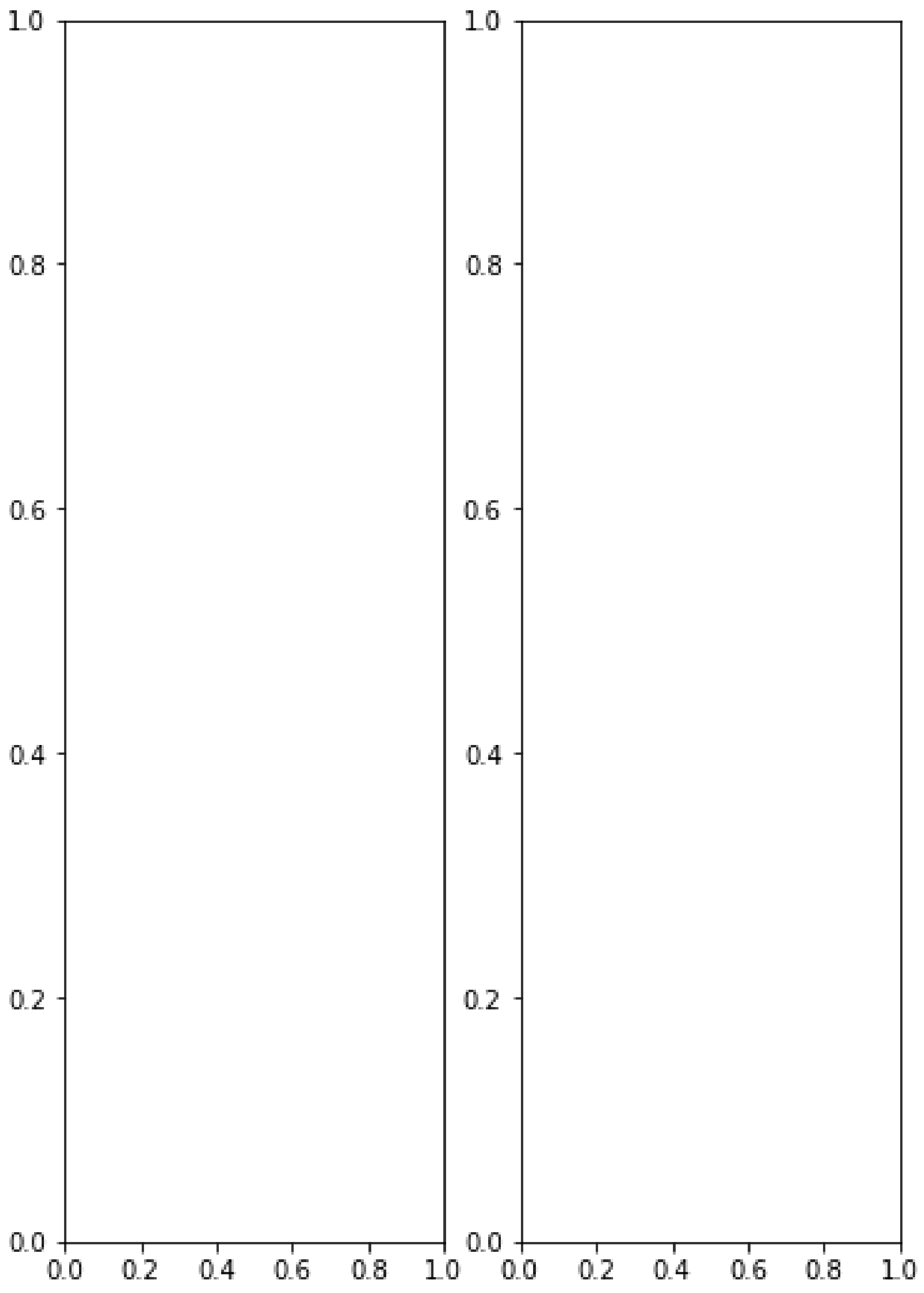
```
        highest = np.argmax(a[L])
#         print(np.argmax(a[L]))
#         print(highest, "--- ", y_this)
        if highest == y_this:
            truePositives.append(True)

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

```
Accuracy:  0.6888888888888889
```

[4]:
```
import helper
img_idx = np.random.random_integers(digits.images.shape[0])
img = digits.images[img_idx]
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]))

# helper.view_classify(img.view( 8, 8), a[L])
import matplotlib.pyplot as plt
# ps = a[L].data.numpy().squeeze()
fig, (ax1, ax2) = plt.subplots(figsize=(6,9), ncols=2)
ax1.imshow(img.resize(1, 28, 28))
ax1.axis('off')
ax2.barh(np.arange(10), a[L])
ax2.set_aspect(0.1)
ax2.set_yticks(np.arange(10))
ax2.set_title('Class Probability')
ax2.set_xlim(0, 1.1)
plt.tight_layout()
```

```
/home/prithvi/anaconda3/envs/pytorch/lib/python3.7/site-
packages/ipykernel_launcher.py:2: DeprecationWarning: This function is
deprecated. Please call randint(1, 1797 + 1) instead



       ␣
 ↪---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call␣
 ↪last)

        <ipython-input-4-2580545c9788> in <module>
```

```
     13 # ps = a[L].data.numpy().squeeze()
     14 fig, (ax1, ax2) = plt.subplots(figsize=(6,9), ncols=2)
---> 15 ax1.imshow(img.resize_(1, 28, 28).numpy().squeeze())
     16 ax1.axis('off')
     17 ax2.barh(np.arange(10), a[L])


    AttributeError: 'numpy.ndarray' object has no attribute 'resize_'
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

[ ]: