Digit Recognition

Aryan Mathe

November 28, 2022

1 *Digit Recognition Model*

1.0.1 *The motive of this project is to understand the mathematics behind neural networks along with implementing and analyzing the effect various parametres such as learning rate, initial conditions and collective or distributed rendering.*

The neural network consists of one hidden layer, and a essential implementation of activation functions which are ReLU (Rectified Linear Unit) and Entropy Cost function

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[6]:
```
data = pd.read_csv('./data/train.csv')
```

1.1 *Initial Conditions*

[11]:
```
data = np.array(data)

Y_train = data[:, 0]
X_train = data[:, 1:]

m, n = X_train.shape

def init(x, y):
    return np.random.uniform(-1.0, 1.0, size=(x, y))/np.sqrt(x*y)

W1 = init(128, 784)
W2 = init(10, 128)
B1 = np.random.uniform(-1.0, 1.0, size = (128, 1))
B2 = np.random.uniform(-1.0, 1.0, size = (10, 1))

r = 0.5e-6
X_train = X_train.T
```

1.2 *Activation Functions*

$$RelU : f(x) = max(0, x)$$

$$Softmax : f(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

[12]:
```
def softmax(X):
    a = np.exp(X)/np.sum(np.exp(X), axis = 0)
    return a

def relU(X):
    return np.where(X>0, X, 0)

def dRelU(X):
    return np.where(X>0, 1, 0)
```

1.3 *Forward and Backward propagation step*

- This is the most crucial and most mathematical step of the model training.
- The gradient section of the code contains the derivatives of cost function with respect to each of the weights and biases.
- In order to calculate that we also have to calulate the derivative with respect to each layer which forms the network.
- After doing that just simply update the parametres with an appropriately chosen learning rate.

[13]:
```
def forward_backwardPass(X, W1, W2, b1, b2, size, targets):

    # Current parametres
    z1 = W1@X + b1
    A1 = relU(z1)
    z2 = W2@A1 + b2
    A2 = softmax(z2)

    # Error
    E = A2 - targets

    # Gradients
    dE_dW2 = 2*(A2 - targets)@A1.T
    dE_db2 = np.array([2*(A2 - targets).mean(axis = 1)]).T
    dE_dz1 = 2*(W2.T@(A2-targets))*dRelU(z1)
    dE_dW1 = dE_dz1@X.T
    dE_db1 = np.array([dE_dz1.mean(axis = 1)]).T
    dE_dX = W1.T@dE_dz1

    # Updates
    W2 = W2 - r*dE_dW2
    W1 = W1 - r*dE_dW1
    b2 = b2 - r*dE_db2
    b1 = b1 - r*dE_db1
    z1 = z1 - r*dE_dz1
    X = X - r*dE_dX

    return W1, W2, b1, b2, np.max(A2, axis = 0)
```

1.4 *Batch Rendering*

- Divided the overall dataset of **42000** samples into *42 batches* of **1000** each.
- Analyzing the **mean** of the **maximum value** of the **prediction vector** over all the data set.
- This mean value should eventually converge to **1**, the more the convergence the more our model will predict accurately.

[14]:
```
size = 1000
Mean = []
for j in range(42):
    targets = np.zeros((size,10), np.float32)
    targets[range(targets.shape[0]),Y_train[size*j:size*(j+1)]] = 1
    targets = targets.T

    X = X_train[:, size*j:size*(j+1)]

    for i in range(200):
        W1, W2, B1, B2, E = forward_backwardPass(X, W1, W2, B1, B2, size,␣
 ↪targets)
    Mean.append(np.mean(E))
    print(np.mean(E))
```

```
0.9596248916152026
0.9751643412650458
0.9800578945407937
0.9835800507290838
0.9861852863773365
0.9876019151191031
0.9867030092343527
0.9880650190869014
0.9913903745367582
0.9918091369073359
0.9912219837301702
0.9927628726268062
0.9927353432379518
0.9917583783449855
0.99323969068847
0.9944697155548321
0.9940787120265678
0.9942838440076994
0.9939336471051591
```

```
0.995453514986757
0.9944664589288107
0.9940974143548833
0.9955470961736387
0.9946135608945958
0.9947296453710773
0.9947448190302447
0.9959490049275539
0.9951450536193586
0.9955869667306498
0.9956615453878321
0.9966470807901753
0.9966031118796816
0.9962447408809291
0.9969241491523789
0.9967269069316328
0.9968392005862993
0.9967100555691173
0.9967122040598444
0.9970573616822302
0.9975756948218018
0.9967519446679589
0.9971919833924309
```

1.5 Load the testing data

[15]:
```
print(B2.shape)
test = pd.read_csv('./data/test.csv')
test = np.array(test).T
final = np.argmax(softmax(W2@relU(W1@test + B1) + B2), axis = 0)
print(final)
```

```
(10, 1)
[2 0 9 ... 3 9 2]
```

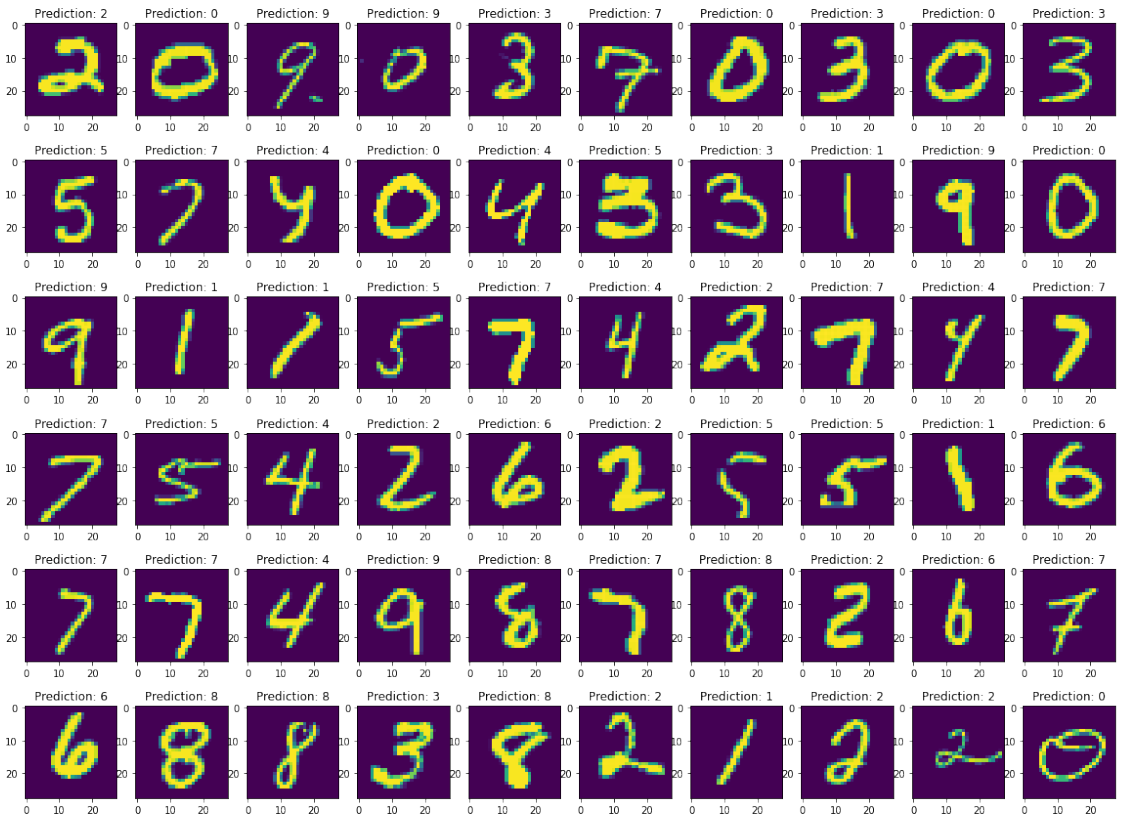
[16]:
```
def plotImage(D):
    D = np.reshape(D, (28, 28))
    return D
```

1.6 *Prediction*

The images below shows the actual image and the title above each image shows the predicted value which is quite accurate for beginner model :)

[17]:
```
rows = 6
cols = 10
fig, ax = plt.subplots(rows, cols, figsize=(20,15))
for j in range(rows):
    for i in range(cols):
        ax[j, i].imshow(plotImage(test[:,10*j + i]))
        ax[j, i].set_title(f"Prediction: {final[10*j + i]}")

plt.show()
```



END OF REPORT