Neural Network from Scratch

Ananda Biswas

Contents

1 Objective

Here I take on **MNIST Handwritten Digits** dataset and build a fully connected **FeedForward Neural Network from scratch** with self implementation of

- Forward Propagation,
- Backward Propagation,
- Gradient Descent.

2 The Math

Given:

- $X \in \mathbb{R}^{n \times m}$ — input data with examples as columns
- m — number of examples
- L — number of layers

The forward propagation equations are:

- $A_0 = X$
- $A_k = W_k H_{k-1} + b_k \cdot \mathbf{1}_m^T \quad \forall\ k = 1, 2, \ldots, L$
- $H_k = g_k(A_k) \quad \forall\ k = 1, 2, \ldots, L-1$
- $\hat{Y} = \text{softmax}(A_L)$

The backward propagation equations are:

- $\nabla W_k = \nabla A_k \cdot H_{k-1}^T\ \forall\ k = 1(1)L$
- $\nabla b_k = \nabla A_k \cdot \mathbf{1}_m\ \forall\ k = 1(1)L$ where $\mathbf{1}_m \in \mathbb{R}^{m \times 1}$
- $\nabla A_L = \frac{1}{m}\left(\hat{Y} - E\right)$ where $E \in \mathbb{R}^{p \times m}$ is the one-hot encoded label matrix for all the data points
- $\nabla A_k = \nabla H_k \odot g'(A_k) = \left(W_{k+1}^T \cdot \nabla A_{k+1}\right) \odot g'(A_k)\ \forall\ k = 1(1)\overline{L-1}$

3 Necessary Imports

[1]:
```
import numpy as np
import cupy as cp

from matplotlib import pyplot as plt

import tensorflow as tf # for loading data
```

4 Loading the data

[2]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434              2s
0us/step
```

[3]:
```
print(f"Train images : {x_train.shape}")
print(f"Train labels : {y_train.shape}")

print(f"Test images : {x_test.shape}")
print(f"Test labels : {y_test.shape}")
```

```
Train images : (60000, 28, 28)
Train labels : (60000,)
Test images : (10000, 28, 28)
Test labels : (10000,)
```

5 Hovering over the data

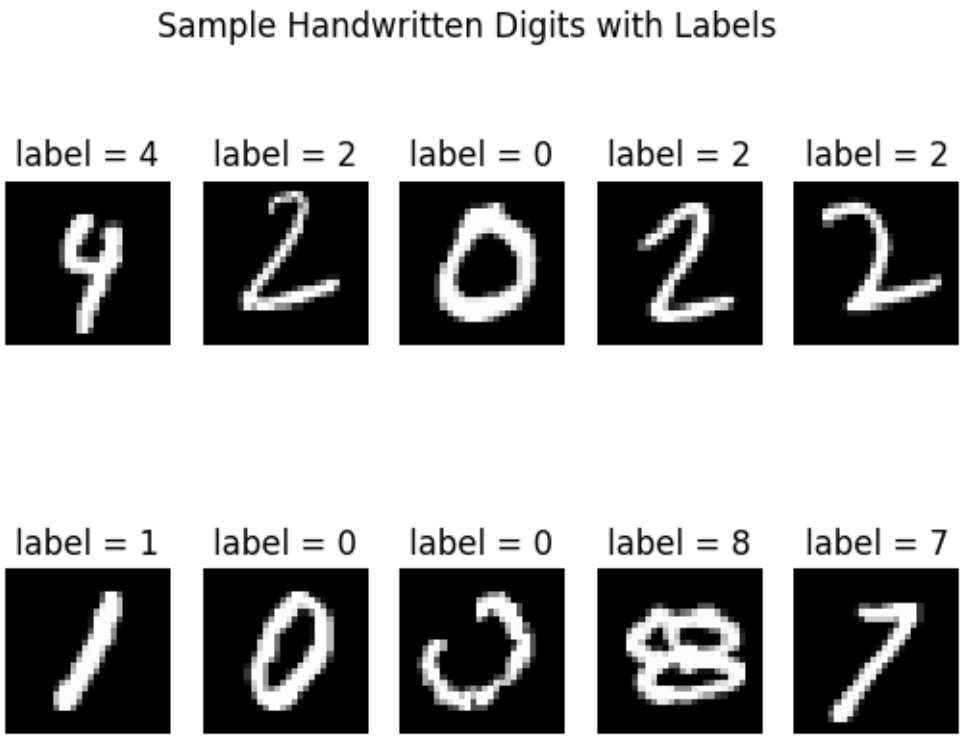
[4]:
```
index = np.random.randint(low = 0, high = 60000, size = 10)

fig, axes = plt.subplots(2, 5)

for i, ax in enumerate(axes.flat) :
  ax.imshow(x_train[index[i]], cmap = "gray")
  ax.axis('off')
  ax.set_title(f"label = {y_train[index[i]]}")

fig.suptitle("Sample Handwritten Digits with Labels")

plt.show()
```



[5]:
```
X = x_train.reshape(60000, 784).T / np.max(x_train)
X = cp.array(X)
print(X.shape)
```

```
(784, 60000)
```

[6]:
```
n, m = X.shape
```

6 Some Preliminaries

6.1 Activation Function and its derivative

[7]:
```
def ReLU(x) :
  return cp.maximum(0, x)
```

[8]:
```
def ReLU_dash(x) :
  return cp.array(x > 0, dtype = cp.int8)
```

6.2 Output Function

[9]:
```
def Softmax(x):
    exp_shifted = cp.exp(x - cp.max(x, axis=0, keepdims=True))
    return exp_shifted / cp.sum(exp_shifted, axis=0, keepdims=True)
```

6.3 One-hot encoding

[10]:
```
def one_hot(x) :
  return cp.array(cp.arange(10) == x, dtype = cp.int8)
```

[11]:
```
train_labels = cp.empty((10, 60000))

for i in range(60000) :
  train_labels[:, i] = one_hot(y_train[i])
```

7 Parameter Initialization

♠ We construct a neural network with

- an **input layer** with 784 neurones,
- **two hidden layers** with 500 and 150 neurones respectively,
- an **output layer** with 10 neurones.
- At the hidden layers, **ReLU** is used as an activation function,
- At the output layer, **Softmax** is used as activation function.
- **Average Sparse Categorical Cross-Entropy** is the loss function here.

[12]:
```
def init_params() :
  W1 = cp.random.randn(500, 784) * 0.01
  b1 = cp.zeros((500, 1))

  W2 = cp.random.randn(150, 500) * 0.01
  b2 = cp.zeros((150, 1))

  W3 = cp.random.randn(10, 150) * 0.01
  b3 = cp.zeros((10, 1))

  return W1, b1, W2, b2, W3, b3
```

8 Forward Propagation

[13]:
```
def forward_propagation(W1, b1, W2, b2, W3, b3, X) :
  A1 = cp.dot(W1, X) + b1
  H1 = ReLU(A1)
```

```
    A2 = cp.dot(W2, H1) + b2
    H2 = ReLU(A2)

    A3 = cp.dot(W3, H2) + b3
    H3 = Softmax(A3)

    return A1, H1, A2, H2, A3, H3
```

9 Backward Propagation

[14]:
```
def backward_propagation(W1, b1, W2, b2, W3, b3, Y_hat, A1, A2, H1, H2, A3, H3,␣
↪X, train_labels) :
    # output layer
    d_A3 = (Y_hat - train_labels) / m

    d_W3 = cp.dot(d_A3, H2.T)
    d_b3 = cp.sum(d_A3, axis=1, keepdims=True)

    # hidden layer 2
    d_H2 = cp.dot(W3.T, d_A3)

    d_A2 = d_H2 * ReLU_dash(A2)

    d_W2 = cp.dot(d_A2, H1.T)
    d_b2 = cp.sum(d_A2, axis=1, keepdims=True)

    # hidden layer 1
    d_H1 = cp.dot(W2.T, d_A2)

    d_A1 = d_H1 * ReLU_dash(A1)

    d_W1 = cp.dot(d_A1, X.T)
    d_b1 = cp.sum(d_A1, axis=1, keepdims=True)

    return d_W1, d_b1, d_W2, d_b2, d_W3, d_b3
```

10 Gradient Descent

[15]:
```
def gradient_descent(iterations, eta) :

    t = iterations

    W1, b1, W2, b2, W3, b3 = init_params()

    i = 1
```

```
  while(i <= t) :
    # forward propagation
    A1, H1, A2, H2, A3, H3 = forward_propagation(W1, b1, W2, b2, W3, b3, X)

    Y_hat = H3

    # backward propagation
    d_W1, d_b1, d_W2, d_b2, d_W3, d_b3 = backward_propagation(W1, b1, W2, b2,␣
↪W3, b3, Y_hat, A1, A2, H1, H2, A3, H3, X, train_labels)

    # update parameters
    W1 = W1 - eta * d_W1
    W2 = W2 - eta * d_W2
    W3 = W3 - eta * d_W3

    b1 = b1 - eta * d_b1
    b2 = b2 - eta * d_b2
    b3 = b3 - eta * d_b3

    i = i + 1

    if(i % 10 == 0) :
      loss = -cp.mean(cp.sum(train_labels * cp.log(Y_hat + 1e-8), axis=0))
      print(f"Iteration {i}, Loss: {loss:.4f}")

  return W1, b1, W2, b2, W3, b3
```

11 Model Training

[16]:
```
W1, b1, W2, b2, W3, b3 = gradient_descent(iterations=1000, eta=0.1)
```

```
Iteration 10, Loss: 2.3020
Iteration 20, Loss: 2.3013
Iteration 30, Loss: 2.3007
Iteration 40, Loss: 2.3000
Iteration 50, Loss: 2.2993
Iteration 60, Loss: 2.2984
Iteration 70, Loss: 2.2974
Iteration 80, Loss: 2.2961
Iteration 90, Loss: 2.2944
Iteration 100, Loss: 2.2921
Iteration 110, Loss: 2.2889
Iteration 120, Loss: 2.2843
Iteration 130, Loss: 2.2772
Iteration 140, Loss: 2.2660
Iteration 150, Loss: 2.2470
```

```
Iteration 160, Loss: 2.2146
Iteration 170, Loss: 2.1622
Iteration 180, Loss: 2.0902
Iteration 190, Loss: 1.9987
Iteration 200, Loss: 1.8743
Iteration 210, Loss: 1.7151
Iteration 220, Loss: 1.5459
Iteration 230, Loss: 1.3899
Iteration 240, Loss: 1.2518
Iteration 250, Loss: 1.1238
Iteration 260, Loss: 1.0097
Iteration 270, Loss: 0.9187
Iteration 280, Loss: 0.8507
Iteration 290, Loss: 0.7999
Iteration 300, Loss: 0.7605
Iteration 310, Loss: 0.7287
Iteration 320, Loss: 0.7019
Iteration 330, Loss: 0.6785
Iteration 340, Loss: 0.6575
Iteration 350, Loss: 0.6381
Iteration 360, Loss: 0.6200
Iteration 370, Loss: 0.6028
Iteration 380, Loss: 0.5864
Iteration 390, Loss: 0.5708
Iteration 400, Loss: 0.5560
Iteration 410, Loss: 0.5422
Iteration 420, Loss: 0.5294
Iteration 430, Loss: 0.5175
Iteration 440, Loss: 0.5067
Iteration 450, Loss: 0.4967
Iteration 460, Loss: 0.4875
Iteration 470, Loss: 0.4790
Iteration 480, Loss: 0.4710
Iteration 490, Loss: 0.4637
Iteration 500, Loss: 0.4567
Iteration 510, Loss: 0.4502
Iteration 520, Loss: 0.4439
Iteration 530, Loss: 0.4380
Iteration 540, Loss: 0.4324
Iteration 550, Loss: 0.4270
Iteration 560, Loss: 0.4218
Iteration 570, Loss: 0.4169
Iteration 580, Loss: 0.4121
Iteration 590, Loss: 0.4075
Iteration 600, Loss: 0.4030
Iteration 610, Loss: 0.3987
Iteration 620, Loss: 0.3945
Iteration 630, Loss: 0.3905
```

```
Iteration 640, Loss: 0.3866
Iteration 650, Loss: 0.3828
Iteration 660, Loss: 0.3791
Iteration 670, Loss: 0.3755
Iteration 680, Loss: 0.3720
Iteration 690, Loss: 0.3686
Iteration 700, Loss: 0.3653
Iteration 710, Loss: 0.3620
Iteration 720, Loss: 0.3589
Iteration 730, Loss: 0.3558
Iteration 740, Loss: 0.3529
Iteration 750, Loss: 0.3499
Iteration 760, Loss: 0.3471
Iteration 770, Loss: 0.3443
Iteration 780, Loss: 0.3416
Iteration 790, Loss: 0.3389
Iteration 800, Loss: 0.3363
Iteration 810, Loss: 0.3338
Iteration 820, Loss: 0.3313
Iteration 830, Loss: 0.3288
Iteration 840, Loss: 0.3264
Iteration 850, Loss: 0.3240
Iteration 860, Loss: 0.3217
Iteration 870, Loss: 0.3194
Iteration 880, Loss: 0.3171
Iteration 890, Loss: 0.3148
Iteration 900, Loss: 0.3126
Iteration 910, Loss: 0.3104
Iteration 920, Loss: 0.3082
Iteration 930, Loss: 0.3061
Iteration 940, Loss: 0.3039
Iteration 950, Loss: 0.3018
Iteration 960, Loss: 0.2998
Iteration 970, Loss: 0.2977
Iteration 980, Loss: 0.2957
Iteration 990, Loss: 0.2937
Iteration 1000, Loss: 0.2916
```

12 Accuracy

[17]:
```python
def get_predictions(x):
    return cp.argmax(x, axis=0)

def get_accuracy(predictions, Y):
    return cp.mean(predictions == Y) * 100
```

[18]:
```
A1_test, H1_test, A2_test, H2_test, A3_test, H3_test = forward_propagation(W1,␣
↪b1, W2, b2, W3, b3, cp.array(x_test.reshape(10000, 784)).T)

# H3_test to get predictions
predictions = get_predictions(H3_test)

accuracy = get_accuracy(predictions, cp.array(y_test))
print("Test set accuracy: {:.2f}%".format(accuracy))
```

```
Test set accuracy: 90.95%
```