Digit Recognition

Name - Abhas Kumar
Roll No - 1905513

In [89]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [90]:
```
data = pd.read_csv("data.csv")
```

In [91]:
```
data.head()
```

Out[91]:

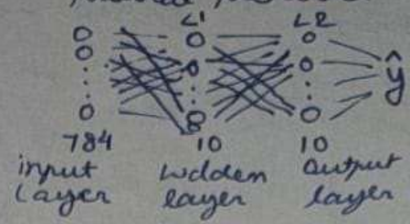
	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pix
0	1	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 785 columns

In [92]:
```
data = np.array(data)
m, n = data.shape
np.random.shuffle(data)

data_test = data[0:1000].T
Y_test = data_test[0]
X_test = data_test[1:n]
X_test = X_test / 255.

data_train = data[1000:m].T
Y_train = data_train[0]
X_train = data_train[1:n]
X_train = X_train / 255.
_,m_train = X_train.shape
```

$$X = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^m \end{bmatrix}^T = [x^1 \; x^2 \; \ldots \; x^m]$$

Neural Network



$$A^{(0)} = \underset{784 \times m}{X}$$

L1: $\underset{10\times m}{Z^{(1)}} = \underset{10\times 784}{W^{(1)}} \cdot \underset{784\times m}{A^{(0)}} + \underset{10\times 1}{b^{(1)}}$

Now, to introduce non-linearity in the neuron we will use ReLU(x) function (due to its simplicity)

ReLU: Rectified linear unit. $ReLU(x) = \begin{cases} x ; & x > 0 \\ 0 ; & x \le 0 \end{cases}$

$\therefore A^{(1)} = g(Z^{(1)}) = ReLU(Z^{(1)})$

L2: $\underset{10\times m}{Z^{(2)}} = \underset{10\times 10}{W^{(2)}} \underset{10\times m}{A^{(1)}} + \underset{10\times 1}{b^{(2)}}$

Here, activation function will be softmax fn.

$$sf(x) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}} \qquad \therefore A^{(2)} = softmax(Z^{(2)})$$

Backward propogation:

L2:

$\rightarrow \underset{10\times m}{dZ^{(2)}} = \underset{10\times m}{A^{(2)}} - \underset{10\times m}{Y}$

$\rightarrow \underset{10\times 10}{dW^{(2)}} = \frac{1}{m} \underset{10\times m}{dZ^{(2)}} \underset{m\times 10}{A^{(1)}}$ (~~avg~~ error wt)

$\rightarrow db^{(2)} = \frac{1}{m} \sum dZ^{(2)}$ (avg error bias)

L1:

$\rightarrow dZ^{(1)} = W^{(2)} dZ^{(2)}$

$\rightarrow dW^{(1)} = \frac{1}{m} dZ^{(1)} X$

$\rightarrow db^{(1)} = \frac{1}{m} \sum dZ^{(1)}$

Update:

$W^{(1)} = W^{(1)} - \alpha dW^{(1)}$ $\alpha \rightarrow$ learning rate

$b^{(1)} = b^{(1)} - \alpha db^{(1)}$

$W^{(2)} = W^{(2)} - \alpha dW^{(2)}$

$b^{(2)} = b^{(2)} - \alpha db^{(2)}$

Forward prop $\rightarrow$ Backward prop $\rightarrow$ Upade

In [93]:

```
def init_params():
    W1 = np.random.rand(10, 784) - 0.5
    b1 = np.random.rand(10, 1) - 0.5
    W2 = np.random.rand(10, 10) - 0.5
    b2 = np.random.rand(10, 1) - 0.5
    return W1, b1, W2, b2

def ReLU(Z):
    return np.maximum(Z, 0)

def softmax(Z):
    A = np.exp(Z) / sum(np.exp(Z))
    return A

def forward_prop(W1, b1, W2, b2, X):
    Z1 = W1.dot(X) + b1
    A1 = ReLU(Z1)
    Z2 = W2.dot(A1) + b2
    A2 = softmax(Z2)
    return Z1, A1, Z2, A2

def ReLU_deriv(Z):
    return Z > 0

def one_hot(Y):
    one_hot_Y = np.zeros((Y.size, Y.max() + 1))
    one_hot_Y[np.arange(Y.size), Y] = 1
    one_hot_Y = one_hot_Y.T
    return one_hot_Y

def backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y):
    one_hot_Y = one_hot(Y)
    dZ2 = A2 - one_hot_Y
    dW2 = 1 / m * dZ2.dot(A1.T)
    db2 = 1 / m * np.sum(dZ2)
    dZ1 = W2.T.dot(dZ2) * ReLU_deriv(Z1)
    dW1 = 1 / m * dZ1.dot(X.T)
    db1 = 1 / m * np.sum(dZ1)
    return dW1, db1, dW2, db2

def update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha):
    W1 = W1 - alpha * dW1
    b1 = b1 - alpha * db1
    W2 = W2 - alpha * dW2
    b2 = b2 - alpha * db2
    return W1, b1, W2, b2
```

In [94]:

```
def get_predictions(A2):
    return np.argmax(A2, 0)

def get_accuracy(predictions, Y):
    #print(predictions, Y)
    return np.sum(predictions == Y) / Y.size

def gradient_descent(X, Y, alpha, iterations):
    W1, b1, W2, b2 = init_params()
    for i in range(iterations):
        Z1, A1, Z2, A2 = forward_prop(W1, b1, W2, b2, X)
        dW1, db1, dW2, db2 = backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y)
        W1, b1, W2, b2 = update_params(W1, b1, W2, b2, dW1, db1, dW2, db
        if i % 10 == 0:
            print("Iteration: ", i)
            predictions = get_predictions(A2)
            print(get_accuracy(predictions, Y))
    return W1, b1, W2, b2
```

In [111]:

```
W1, b1, W2, b2 = gradient_descent(X_train, Y_train, 0.10, 1500)
```

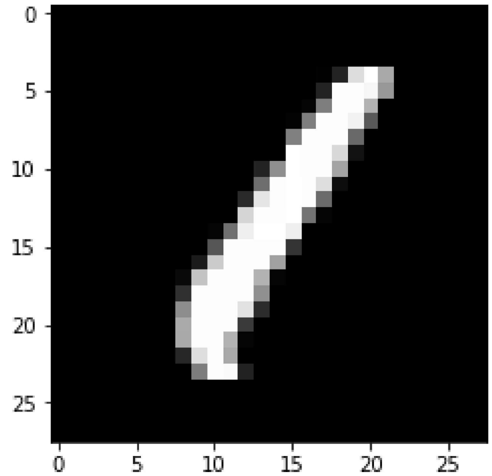
```
0.8934634146341464
Iteration:  1360
0.8936829268292683
Iteration:  1370
0.8938780487804878
Iteration:  1380
0.8940487804878049
Iteration:  1390
0.8941219512195122
Iteration:  1400
0.8941707317073171
Iteration:  1410
0.8942926829268293
Iteration:  1420
0.894609756097561
Iteration:  1430
0.8948048780487805
Iteration:  1440
0.8949512195121951
Iteration:  1450
```

In [112]:

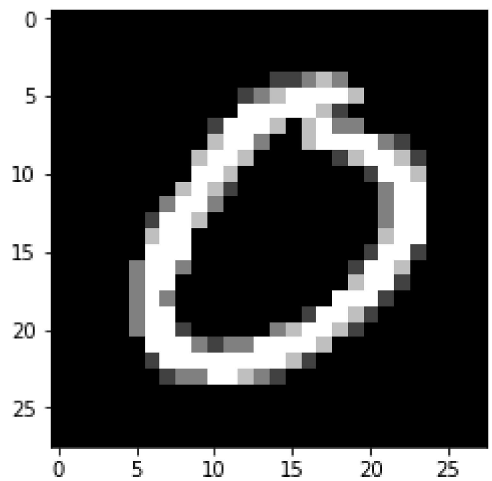
```
def make_predictions(X, W1, b1, W2, b2):
    _, _, _, A2 = forward_prop(W1, b1, W2, b2, X)
    predictions = get_predictions(A2)
    return predictions

def test_prediction(index, W1, b1, W2, b2):
    current_image = X_train[:, index, None]
    prediction = make_predictions(X_train[:, index, None], W1, b1, W2, b
    label = Y_train[index]
    print("Prediction: ", prediction)
    print("Label: ", label)

    current_image = current_image.reshape((28, 28)) * 255
    plt.gray()
    plt.imshow(current_image, interpolation='nearest')
    plt.show()
```

In [118]:
```
for i in range(5,10):
    test_prediction(i, W1, b1, W2, b2)
```

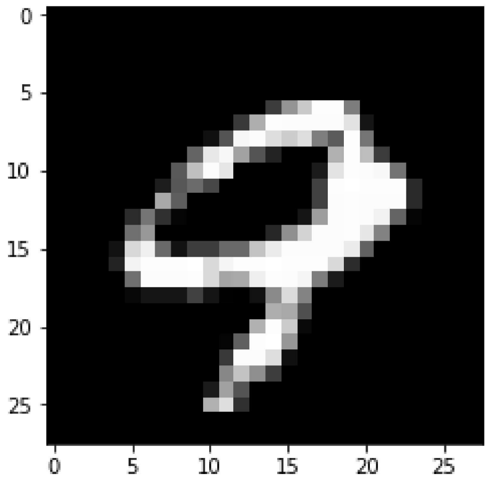
```
Prediction:  [1]
Label:  1
```



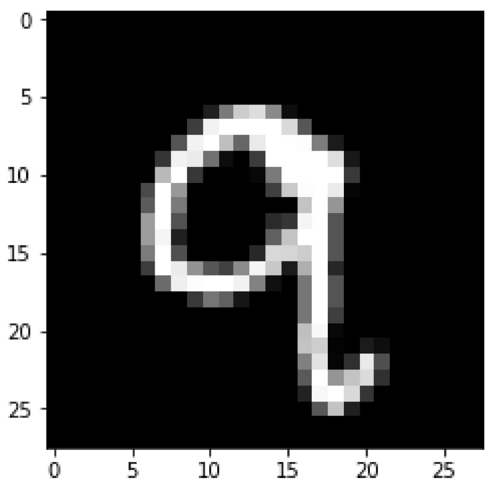
```
Prediction:  [0]
Label:  0
```



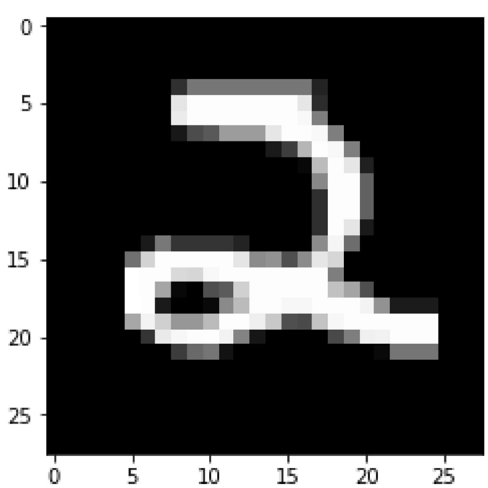
```
Prediction:  [9]
Label:  9
```



```
Prediction:  [9]
Label:  9
```



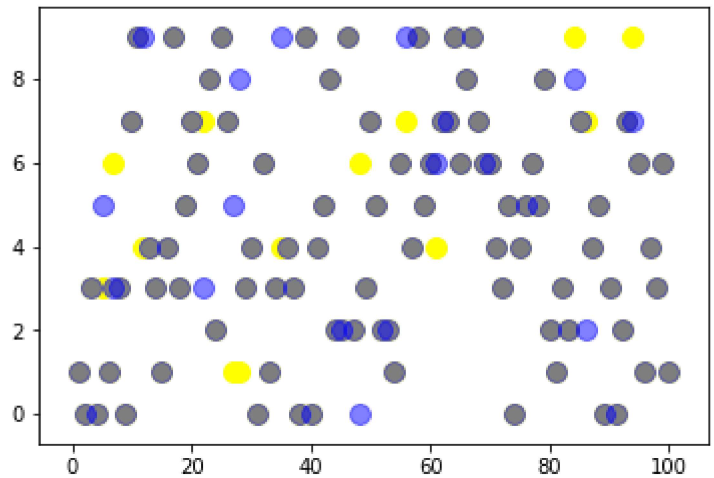
```
Prediction:  [2]
Label:  2
```



In [114]:
```
dev_predictions = make_predictions(X_test, W1, b1, W2, b2)
get_accuracy(dev_predictions, Y_test) * 100
```

Out[114]: 88.6

In [115]:
```
x_axis = range(1, 101)
plt.scatter(x_axis, dev_predictions[0:100], s=100, color="yellow")
plt.scatter(x_axis, Y_test[0:100], s=100, color="blue", alpha=0.5)
plt.show()
```



In [ ]:
```

```