In [22]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

data = pd.read_csv(r'D:\STUDY\mnist_train.csv')
```

In [23]:
```
data
```

Out[23]:

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x22	28x
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
59995	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
59996	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
59997	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
59998	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
59999	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

60000 rows × 785 columns

In [25]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 785 entries, label to 28x28
dtypes: int64(785)
memory usage: 359.3 MB
```

In [27]:
```
arr = data.isna().sum()
```

In [32]:
```
for i in arr:
    if arr[i] != 0:
        print(i)

print("These are the columns that contains Null values.")
print("{IF YOU SEE NO INDEX, THAT MEANS THERE ARE NO COLUMNS THAT HAVE NULL VAI
```

```
These are the columns that contains Null values.
{IF YOU SEE NO INDEX, THAT MEANS THERE ARE NO COLUMNS THAT HAVE NULL VALUES}
```

In [33]:
```
data = np.array(data)
m, n = data.shape  # m = rows, n = columns
np.random.shuffle(data) # shuffle before splitting into dev and training sets

data_dev = data[0:100].T  # transposing the matrix
Y_dev = data_dev[0]
X_dev = data_dev[1:n]
X_dev = X_dev / 255.

data_train = data[100:m].T
Y_train = data_train[0]
X_train = data_train[1:n]
X_train = X_train / 255.
_,m_train = X_train.shape

Y_train
```

Out[33]:
```
array([6, 1, 5, ..., 1, 7, 5], dtype=int64)
```

In [39]:
```
def init_params():
    W1 = np.random.randn(10, 784)
    b1 = np.random.randn(10, 1)
    W2 = np.random.randn(10, 10)
    b2 = np.random.randn(10, 1)
    return W1, b1, W2, b2
```

In [40]:
```
def ReLU(Z):
    return np.maximum(Z, 0)
```

In [41]:
```
def softmax(Z):
    A = np.exp(Z) / sum(np.exp(Z))
    return A
```

In [42]:
```
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
```

In [43]:
```
def ReLU_deriv(Z):
    return Z > 0
```

In [44]:
```
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
```

In [45]:
```
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
```

In [46]:
```
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 [47]:
```
def get_predictions(A2):
    return np.argmax(A2, 0)
```

In [48]:
```
def get_accuracy(predictions, Y):
    print(predictions, Y)
    return np.sum(predictions == Y) / Y.size
```

In [49]:
```
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, db2, alp
        if i % 10 == 0:
            print("Iteration: ", i)
            predictions = get_predictions(A2)
            print(get_accuracy(predictions, Y))
    return W1, b1, W2, b2
```

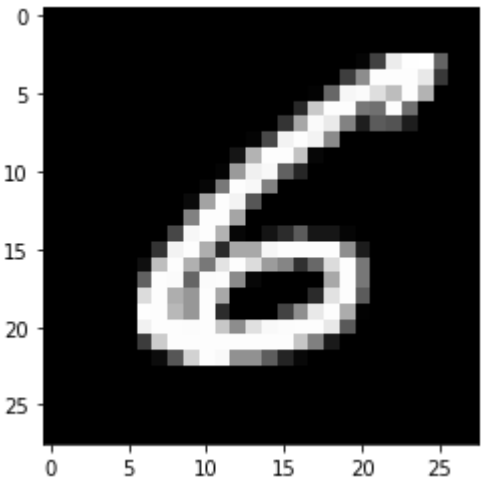
In [51]:
```
n_jobs=-1
W1, b1, W2, b2 = gradient_descent(X_train, Y_train, 0.10, 2500)
```

```
[6 1 5 ... 1 7 3] [6 1 5 ... 1 7 5]
0.6612520868113523
Iteration:  2440
[6 1 5 ... 1 7 3] [6 1 5 ... 1 7 5]
0.6618530884808014
Iteration:  2450
[6 1 5 ... 1 7 3] [6 1 5 ... 1 7 5]
0.6634557595993322
Iteration:  2460
[6 1 5 ... 1 7 5] [6 1 5 ... 1 7 5]
0.663889816360601
Iteration:  2470
[5 1 2 ... 1 7 5] [6 1 5 ... 1 7 5]
0.6496994991652755
Iteration:  2480
[5 1 5 ... 1 7 5] [6 1 5 ... 1 7 5]
0.6687646076794658
Iteration:  2490
[5 1 5 ... 1 7 5] [6 1 5 ... 1 7 5]
0.66669449081803
```

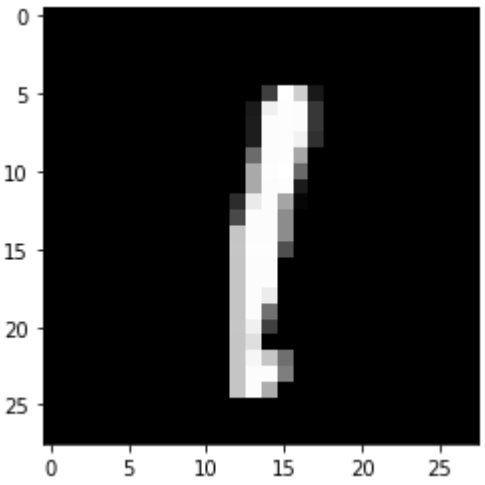
In [52]:
```
def make_predictions(X, W1, b1, W2, b2):
    _, _, _, A2 = forward_prop(W1, b1, W2, b2, X)
    predictions = get_predictions(A2)
    return predictions
```

In [53]:
```
def test_prediction(index, W1, b1, W2, b2):
    current_image = X_train[:, index, None]
    prediction = make_predictions(X_train[:, index, None], W1, b1, W2, b2)
    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()

test_prediction(0, W1, b1, W2, b2)
test_prediction(1, W1, b1, W2, b2)
test_prediction(2, W1, b1, W2, b2)
test_prediction(3, W1, b1, W2, b2)

dev_predictions = make_predictions(X_dev, W1, b1, W2, b2)
print(get_accuracy(dev_predictions, Y_dev))
```

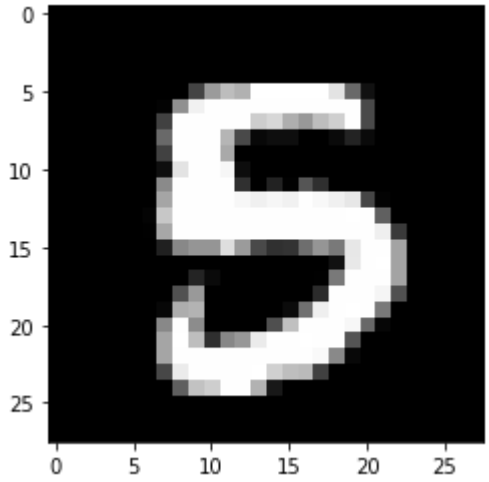
```
Prediction:  [5]
Label:  6
```



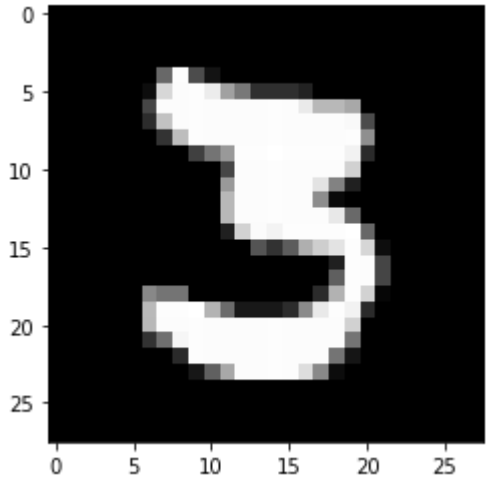

```
Prediction:  [1]
Label:  1
```



```
Prediction:  [5]
Label:  5
```



```
Prediction:  [3]
Label:  3
```



```
[5 0 7 6 9 7 6 5 7 9 9 0 1 6 2 3 2 6 2 7 1 9 6 4 5 3 9 3 2 9 7 1 6 5 3 1 6
 9 2 3 2 6 5 5 1 1 6 4 6 9 4 2 5 6 1 6 4 0 6 2 7 7 9 0 4 0 0 4 4 5 2 3 5 3
 2 9 2 1 1 2 5 9 0 6 1 7 6 2 7 1 1 2 3 7 4 2 1 2 9 7] [7 0 7 8 1 0 6 0 7 5 9
0 1 0 0 3 2 6 2 7 1 3 6 4 9 3 9 9 2 9 7 1 6 0 3 1 6
 9 3 3 5 6 5 5 1 2 6 4 6 9 5 5 8 6 1 6 4 0 6 6 7 7 8 0 4 0 0 4 4 8 2 2 3 3
 3 9 2 8 6 5 4 9 0 6 1 7 6 1 7 1 1 3 3 5 4 2 8 2 7 7]
0.67
```

In [ ]: