By: Ritvik Kapila

ENTRY NUMBER: 2017EE10484

ASSIGNMENT - 1

BACK PROPAGATION

ELL409 | 6th Semester 2019-20 | Electrical Engineering Department

The report depicts the affect of various hyperparameters on the model performance. The varying parameters include:

1. Learning Rate
2. Batch Size
3. Activation Function
4. Cost
5. Regularization Parameter
6. Initialization of weights
7. Number of neurons in the hidden layers
8. Number of hidden layers

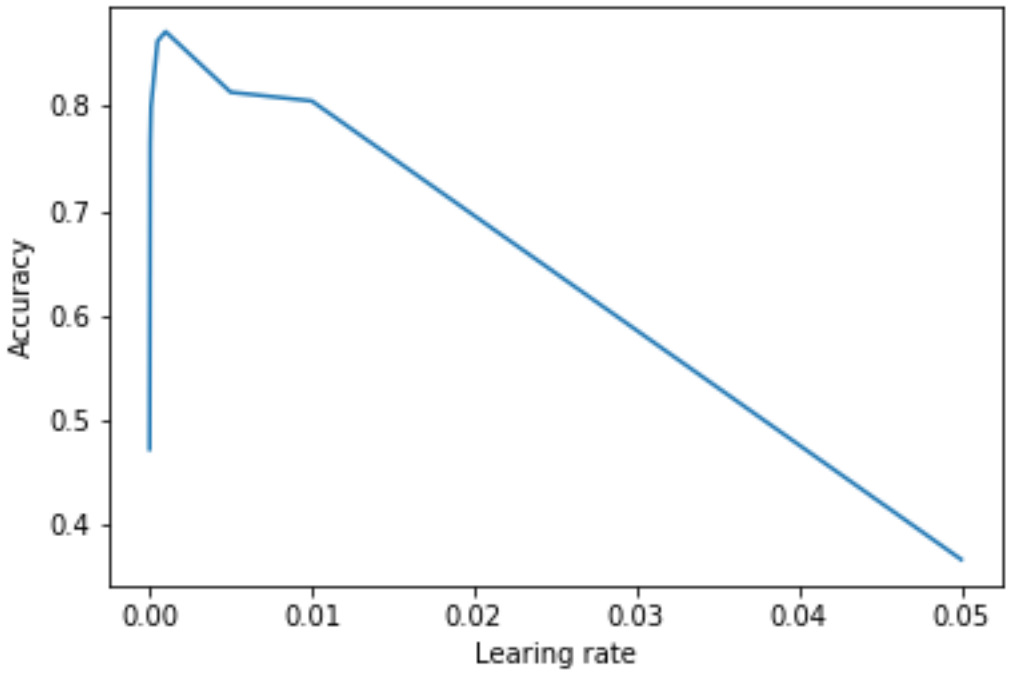
The model that I created is very flexible and can form a neural network varying the number of hidden layers to as high as possible from 0 to infinity, as well as varying the number if neurons in each layer. This helps in finding an optimum solution for the MNIST dataset, or for that matter, any dataset given to us by training on it for various hyperparameters.

For various number of layers and number of neurons in each layer, we need to find the corresponding learning rate and proper initialization of weights that leads to minimum error and maximum efficiency on both train and test data.

I can also vary the learning rate, activation functions and batch size to improve the accuracy of my model and, also the time taken by the model to train on a particular dataset. Batch gradient descent also helps in reducing the effect of outliers on our model.

1. Accuracy vs Learning Rate

The accuracy varies on the learning rate as follows:



From the figure, we can see that accuracy for very low learning rates is really small, as the minima is not being achieved due to the slow movement towards the local minima of error values. For a learning rate higher than a particular threshold, the accuracy falls by a large value because of the divergence of the error due to the weights.

As we keep increasing the learning rate, we reach a point where our accuracy gets stuck at around 9-10 percent. The significance of this value is that all our datapoints are being mapped to a single output as the error diverges due to the high learning rate.

We are experiencing a maximum accuracy at learning rate = 0.001 for

Batch Size = 2

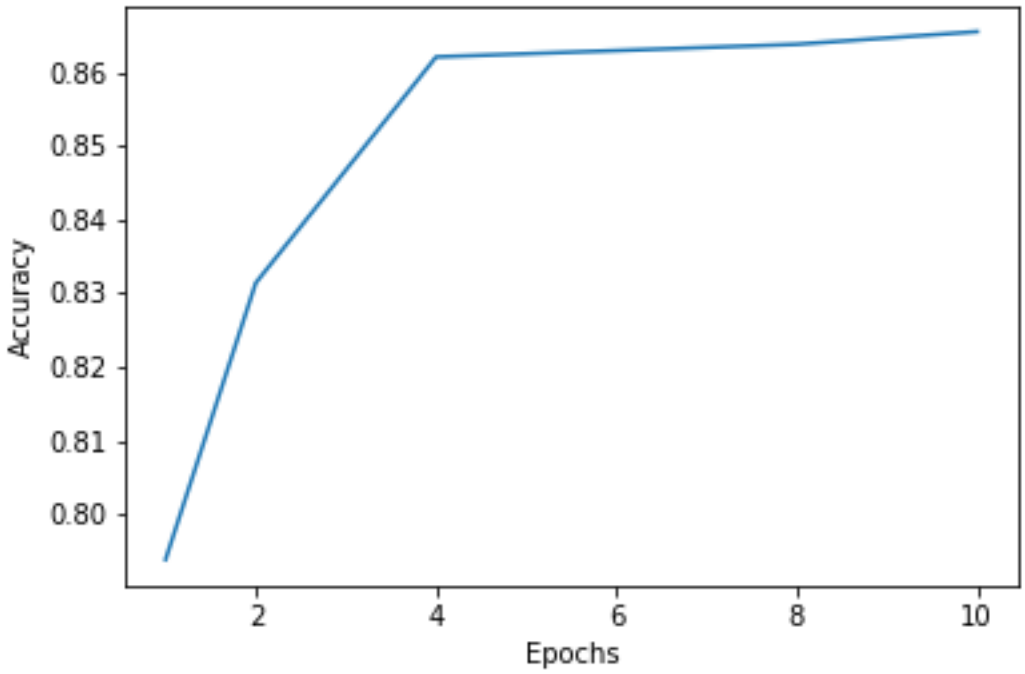
Epochs = 1

Regularization Parameter = 0

Neural Network = [784,200,10] with (tanh, softmax) activation

2. Accuracy vs Epochs

The accuracy varies with the number of epochs as follows:



The graph is in coherence with the fact that as I increase the number of iterations over the training data, the accuracy keeps increasing. Also, as I go ahead in increasing the epochs, the change in accuracy keeps on reducing and finally tends to a constant value approximately.

We get an accuracy of approximately 90% for

Learning Rate = 0.005

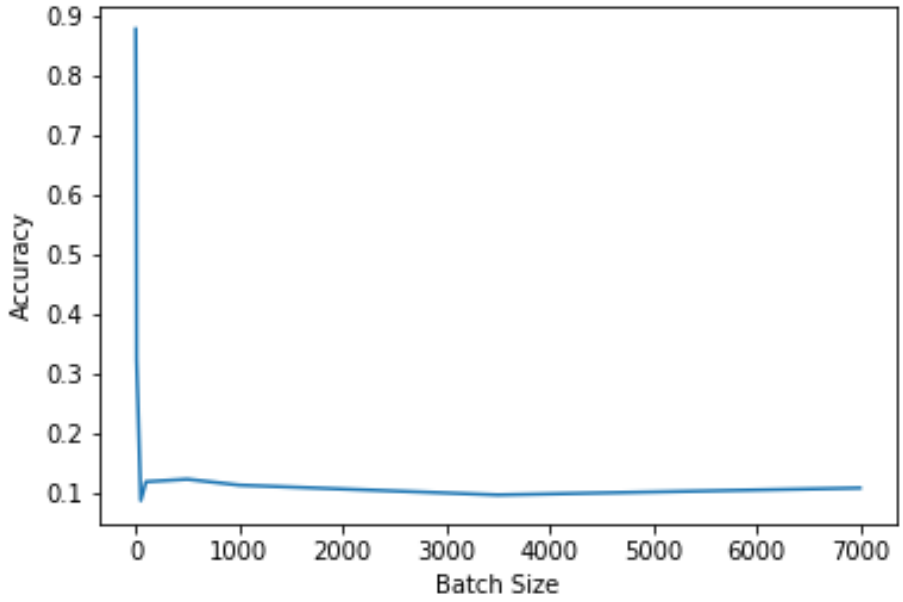
Batch Size = 5

Regularization Parameter = 0

Neural Network = [784,200,10] with (tanh, softmax) activation

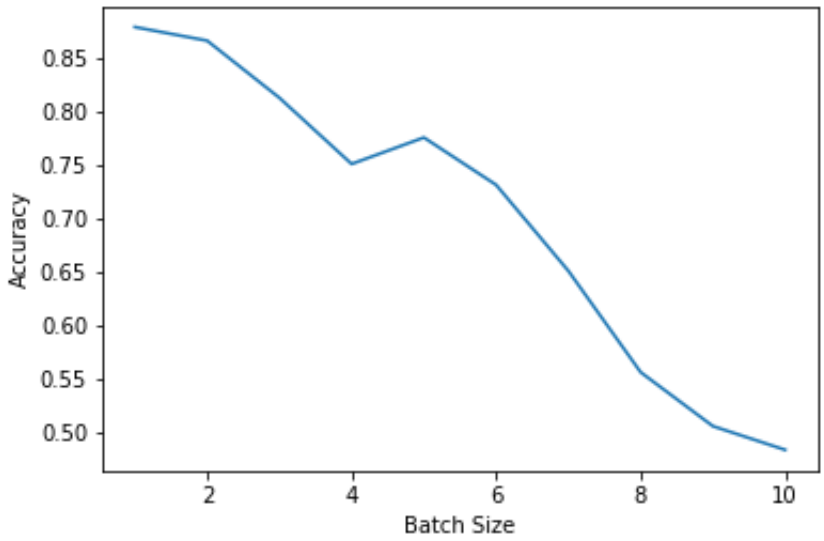
3. Variation with Batch Size

I considered the variation of accuracy for 2 different sets of batch sizes:



Here the batch size goes from 1 to the entire length of the training data i.e. 7000. The accuracy keeps decreasing as I increase my batch size for 1 complete traversal over my training data. This means that for a batch size of 3500, my weights would be updated only twice. Hence the graph shows higher accuracy for stochastic gradient descent as compared to having a large batch size, as expected.

Now we try to zoom into the graph and check the variation from Batch Size of 1 to 10 to find the optimum in order to maximize our accuracy.



The model gives highest accuracy more than 90% for stochastic gradient descent where we have the batch size = 1. On further increasing the batch size, the accuracy reduces for the neural network.

Learning Rate = 0.001

Epochs = 1

Regularization Parameter = 0

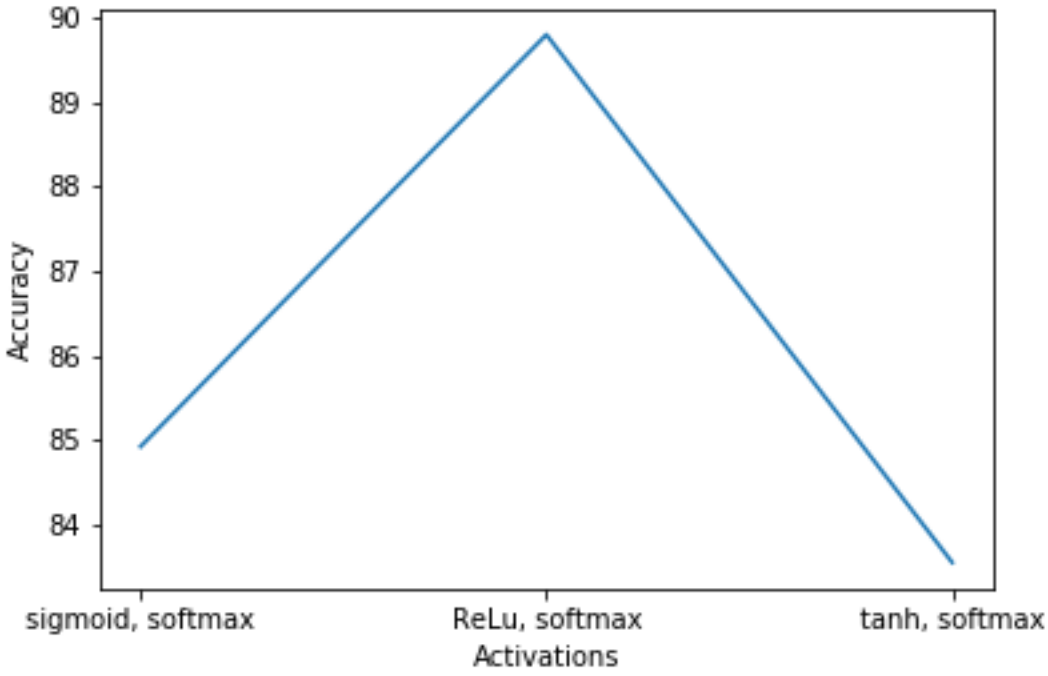
Neural Network = [784,200,10] with (tanh,softmax) activation

4. Variation with Activation Functions

I have used 4 activation functions namely:

1. Tanh
2. Sigmoid
3. ReLu
4. Softmax for the last layer to obtain probabilities

The accuracies for these functions are as follows:



We can clearly see that I have obtained above 90% accuracy for ReLu activation function, followed by sigmoid at 85% and tanh at 84%. Hence the model is most efficient when the neurons are activated using ReLu function for

Learning Rate = 0.005

Epoch = 1

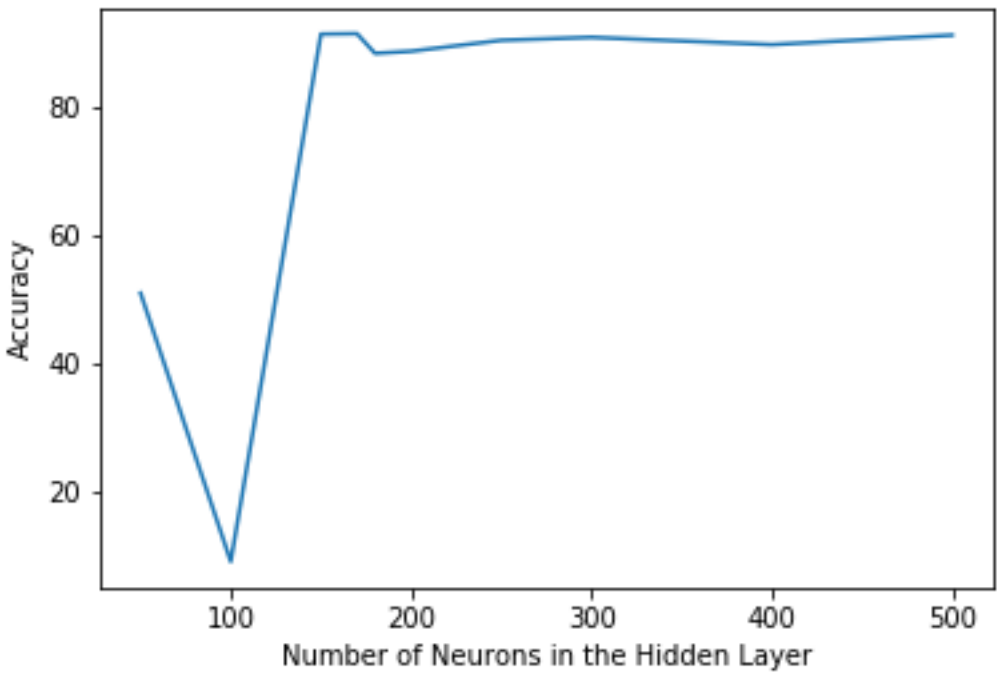
Batch Size = 5

Regularization Parameter = 0

Neural Network = [784,200,10]

5. Neurons in the Hidden Layers

I predicted the labels of the data by varying the number of neurons in the hidden layer. The number of neurons were varied from 50 to 500 and the following results were obtained:



The neural network obtained a maximum accuracy of 91.27% for 170 neurons in the hidden layer for the following hyperparameters

Learning Rate = 0.09

Epoch = 1

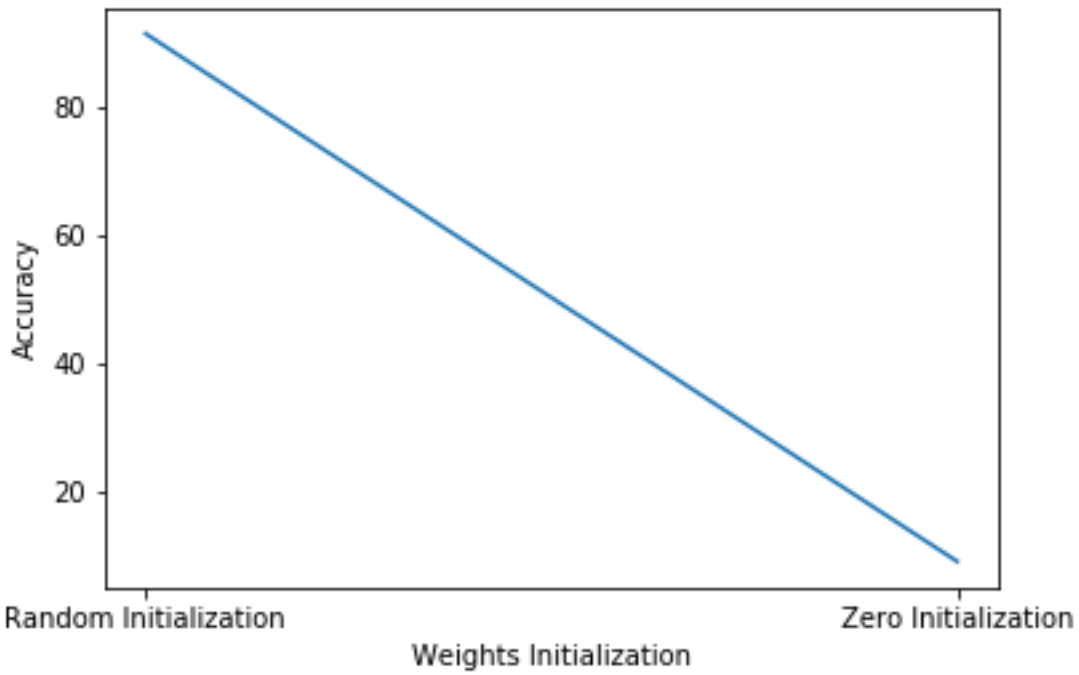
Batch Size = 2

Regularization Parameter = 0

Neural Network = [784,200,10] with (ReLu, softmax) activation

6. Effect of Weights Initialization

In all the above cases, I have trained my model using random weights. But what will happen if I change the initialization of weights. When I trained the model using zero initialized weights, I found the following variation



We obtained 91.27% accuracy for random initialization and 9.31% for zero initialization of weights. This validates the fact that on initializing the weights as zeros, we obtain a very bad accuracy for

Learning Rate = 0.09

Epoch = 1

Batch Size = 2

Regularization Parameter = 0

Neural Network = [784,150,10] with (ReLu, softmax) activation

7. Varying the number of hidden layers

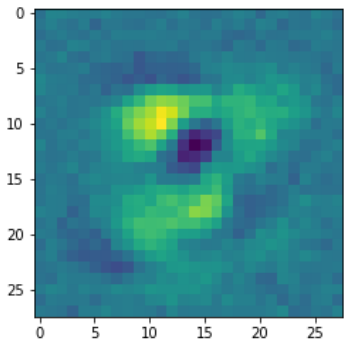
For different number of hidden layers, we will have different optimum hyperparameters. The tuning of these hyperparameters can be done and an optimum accuracy can be found for various hidden layers.

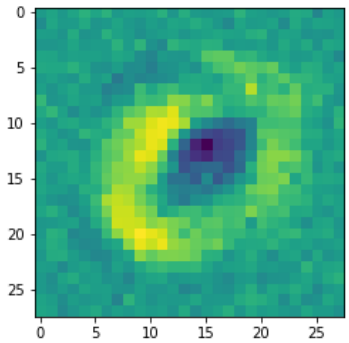
Neural Network = [784, 200, 10] with (ReLu, softmax) activation: For a single hidden layer with 200 neurons, I obtained 93.4% accuracy on training data and 91% accuracy on test data.

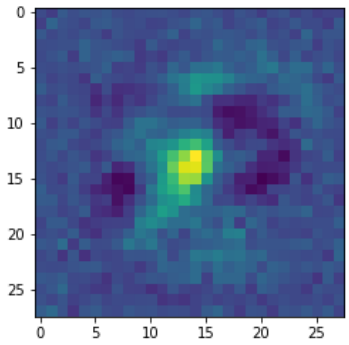
Neural Network = [784, 50, 100, 10] (ReLu, tanh, softmax) activation: For the same hyperparameters, with 50 and 100 neurons in 2 hidden layers with different activations of ReLu and tanh, I obtained 10.4% accuracy.

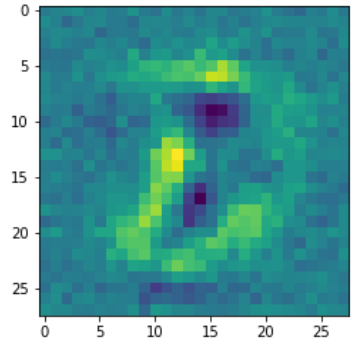
Neural Network = [784, 50, 100, 10] (ReLu, tanh, sigmoid, softmax) activation: Now I further increased the number of hidden layers to 3 for the same hyperparameters, with 50 and 100 and 150 neurons in 3 hidden layers with different activations of ReLu, tanh and tanh, I obtained 9.31% accuracy. The accuracy is very low because the parameters were tuned according to 1 hidden layer.

8. Visualization of Hidden Layer neurons









PROBLEMS FACED DURING TRAINING OF THE MODEL

1. OVERFITTING

As we saw that increasing the number of epochs reduces our training error, but there is a catch which is visible from the result that I encountered from training my network for **30 epochs**. My model achieved an accuracy of 97.14%.

```
# Finding Accuracy of the model for given activations

n_neurons=[785,201,11]
w=[]
for i in range(0,len(n_neurons)-1):
    a=np.random.randn(n_neurons[i+1],n_neurons[i])
    w.append(a)
w=np.true_divide(w,1000)


upd_w = gradient_descent(x, y, w, [785,201,11], ['ReLu', 'softmax'], 0.09, 2, 30, 0.000)
# upd_w = gradient_descent(x, y, upd_w1, [785,201,11], ['tanh', 'tanh', 'softmax'], 0.01, 10, 4)
accuracy=0
for i in range(0,len(x)):
    v=back_propagation(x[i],y[i],upd_w, [785,201,11], ['ReLu', 'softmax'])[1]
    if max_one_hot(v[len(upd_w)])==y[i]:
        accuracy=accuracy+1
accuracy/7000

```

```
0.9714285714285714
```

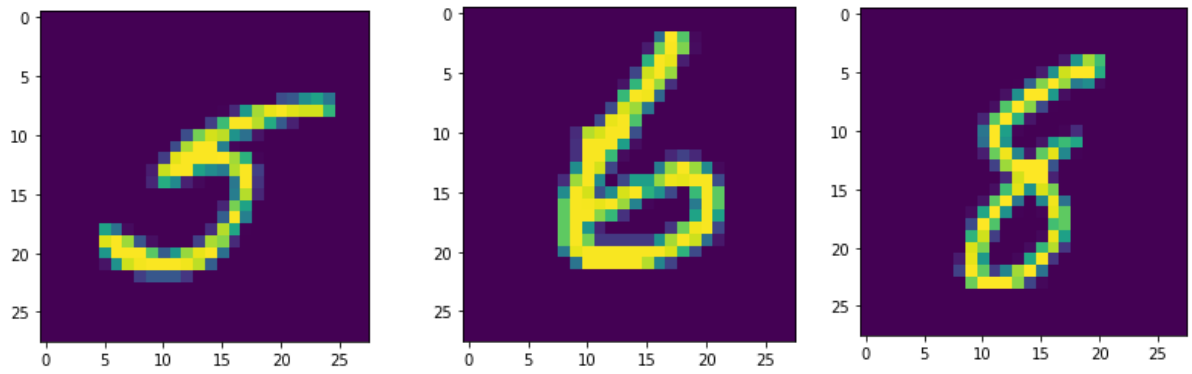
But when I applied this model to predict the labels for the test data, I obtained the following result. Most of the predicted labels were 8. This is a clear indication that I had overfit my neural network on the training data.

```
# Exporting the outputs of the neural network on the test data to a csv file

dftestoutput=pd.DataFrame(np.array([]))
# newee=pd.DataFrame(np.array([1]))
# dftestoutput=dftestoutput.append(newee)
for i in range(0,3000):
    v=back_propagation(x_test[i],y[i],upd_w, [785,201,11], ['tanh', 'softmax'])[1]
    newee=pd.DataFrame(np.array([max_one_hot(v[2])]))
    dftestoutput=dftestoutput.append(newee)
print(dftestoutput)
dftestoutput.to_csv('out.csv')
```

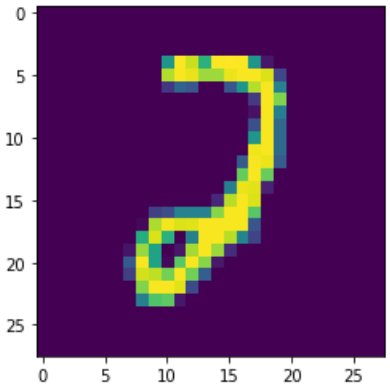
```
     0
0   8.0
0   8.0
0   8.0
0   8.0
0   8.0
..  ...
0   5.0
0   8.0
0   1.0
0   8.0
0   8.0

[3000 rows x 1 columns]
```

2. SIMILAR LABELS AND MISCLASSIFICATIONS

Some labels in training dataset are very much similar to each other. Hence, they create a problem for the model to correctly identify them. An example is the similarity between (5,6) and (6,8) and (5,8)



Another problem that I found in the dataset was that some of the data points are so confusing even to read for us. This is an example of such an outlier:



It is clear that these outliers make things difficult for the model to train.