Building (neural network) classifier for MNIST handwritten digit database

Initial Approach :

Implemented a general L layered deep neural network with variable number of neurons in each layer from scratch. The idea was to initially build a basic model with gradient descent with sigmoid function as activation for the first L-1 layers and the softmax for the final output. The softmax function was used as it gives the probabilities for the multi-class classification problem which also reflects our confidence of prediction. The cost function used was the cross entropy cost, which is used for multi class classification problem. We have also used MSE loss to compare with cross entropy in the end. The graph was then plotted for the training set accuracy v.s. iterations to evaluate the model performance and iterate over the initial model.

A cross validation set was made of 1000 training examples randomly chosen from training set.

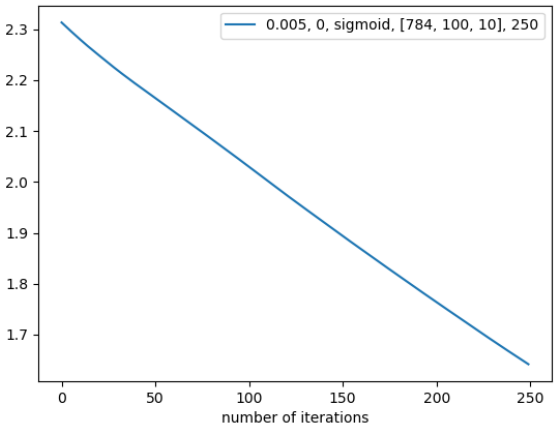
The legend on the graph is on the following order : *learning_rate, Lambda(reg parameter), activation_function, layer_dimensions(INPUT=784, hidden(s), OUTPUT=10), num_iterations*

The following graphs were obtained for one hidden layer with 100 neurons varying the learning rate

rate = 0.005

acc on training set= 0.7395

acc on cross validation set= 0.699

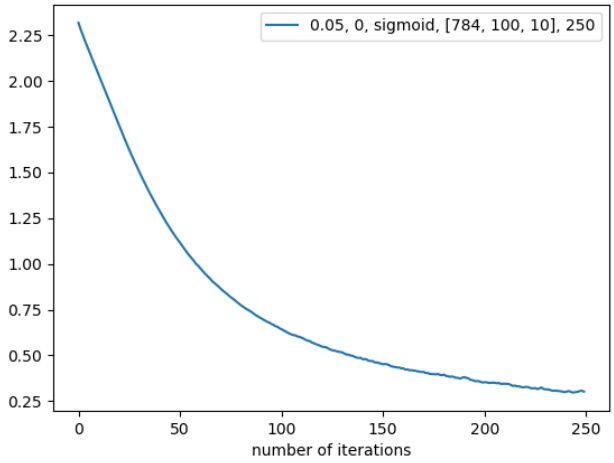


This shows that the backpropagation is working correctly but the learning rate is very low, so we try increasing the learning rate.

rate =0.01

acc on training set= 0.8271666666666667

acc on cross validation set= 0.803



rate = 0.05

acc on training set= 0.9453333333333334

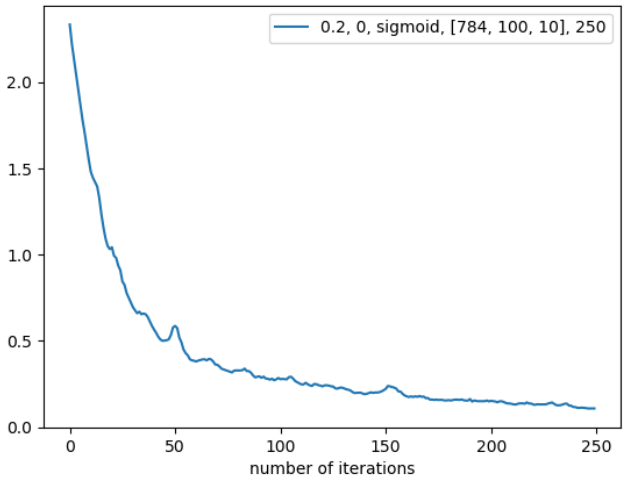
acc on cross validation set= 0.899

We observe that the accuracy has increased drastically both on training and cross validation set by increasing the learning rate (10x) on the same number of iterations and since the graph is still smooth we try to increase the learning rate more to increase accuracy.

rate= 0.1

acc on training set= 0.9658333333333333

acc on cross validation set= 0.916



rate= 0.2

acc on training set= 0.9776666666666667

acc on cross validation set= 0.921

rate= 2

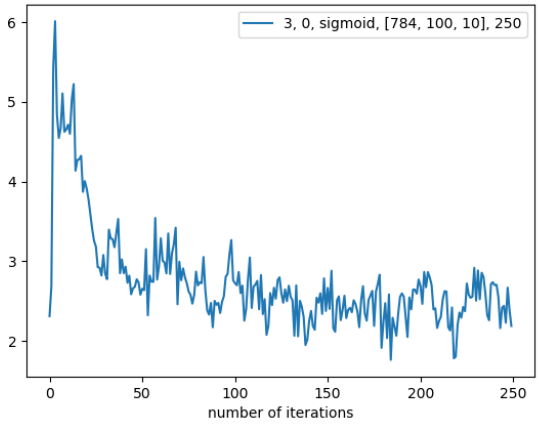
acc on training set= 0.8585

acc on cross validation set= 0.7866

rate= 5

acc on training set= 0.3353333333333333

acc on cross validation set= 0.25246



We see that increasing the learning rate from a very low value to a very high value, first the accuracy increases to a certain point but then after that it goes on to decrease as the learning rate becomes too high that the function iteration starts to overshoot and doesn't converge.

We finally conclude that the graph has finally started to be irregular at a learning rate of 0.2, and since the accuracy on the training set has been increased to 97.7%, we say that this model has reached it's capacity and increasing the learning rate will only overfit the data, so we try a different approach. We try regularization, to avoid overfitting.

We now vary the number of neurons in the hidden layer to see the variation in accuracy for a fixed learning rate, Lambda and number of iterations and activation function.

#neurons =10

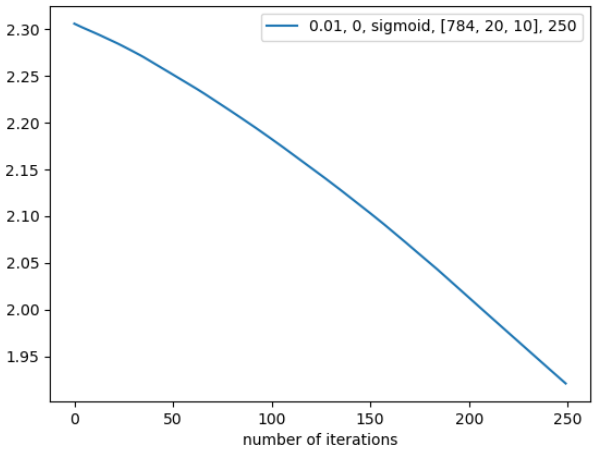
acc on training set= 0.5431666666666667

acc on cross validation set= 0.548

#neurons =20

acc on training set= 0.6546666666666666

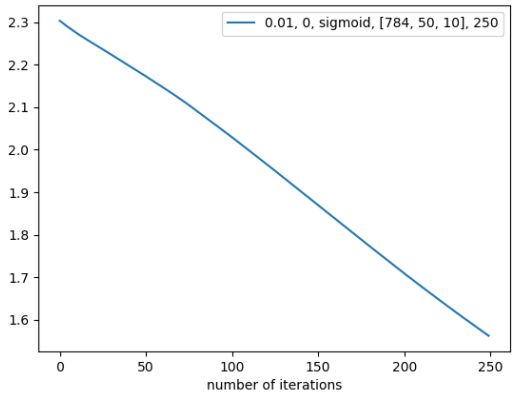
acc on cross validation set= 0.618



#neurons =50

acc on training set= 0.7603333333333333

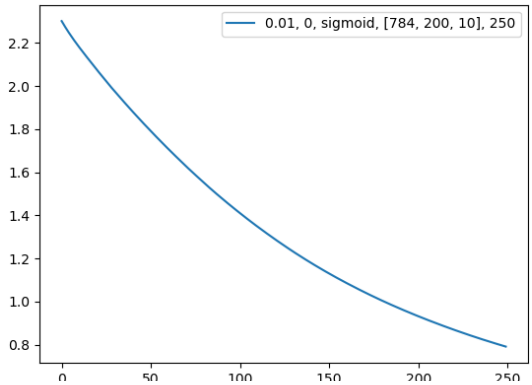
acc on cross validation set= 0.731



#neurons =100

acc on training set= 0.8448333333333333

acc on cross validation set= 0.827



#neurons = 200

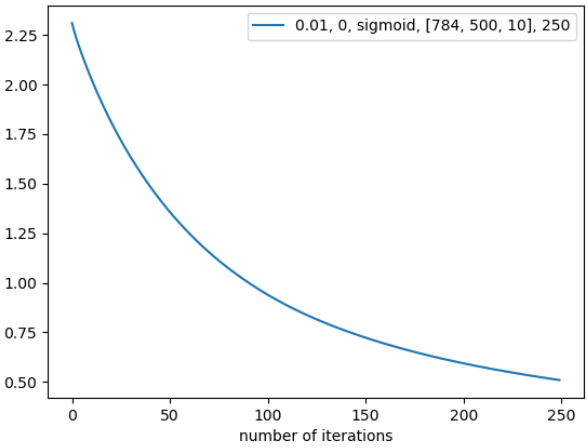
acc on training set= 0.8833333333333333

acc on cross validation set= 0.85

#neurons = 350

acc on training set= 0.9076666666666666

acc on cross validation set= 0.862



#neurons = 500

acc on training set= 0.9156666666666666

acc on cross validation set= 0.869

Hence, we see that our model complexity should be atleast some what good, in a sense that it can learn on the data set properly. However, increasing the model complexity also invites the problem of overfitting, in which the training accuracy increases but cross validation remains fairly the same. This is discussed further.

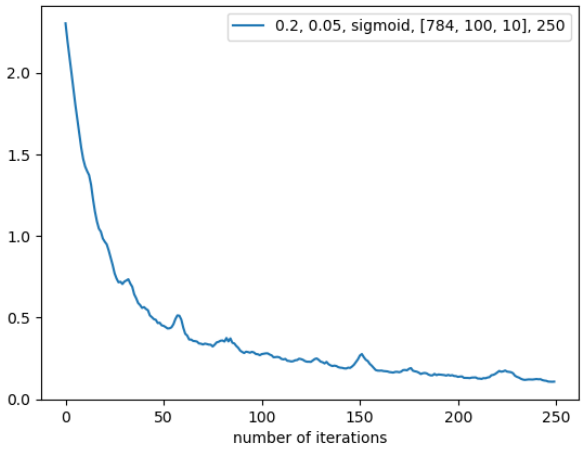
We now try to introduce the regularization parameter Lambda which penalizes high value of weights and biases and prevents overfitting while keeping the learning rate same.

Lambda = 0.05

acc on training set= 0.9785

acc on cross validation set= 0.921

We see that the accuracy on the training data increases only a miniscule amount, and the accuracy on the cross validation set does not increase, which means either the model is not able to do any better with these number of iterations, or we may still be overfitting the data.

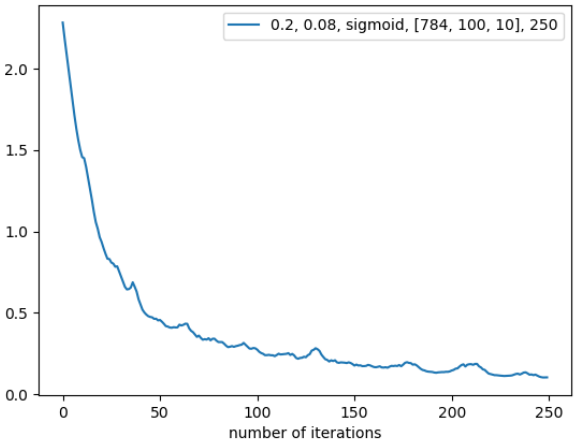


So we try to increase the regularization parameter.

Lambda = 0.1

acc on training set= 0.9616666666666667

acc on cross validation set= 0.918



Lambda =0.15

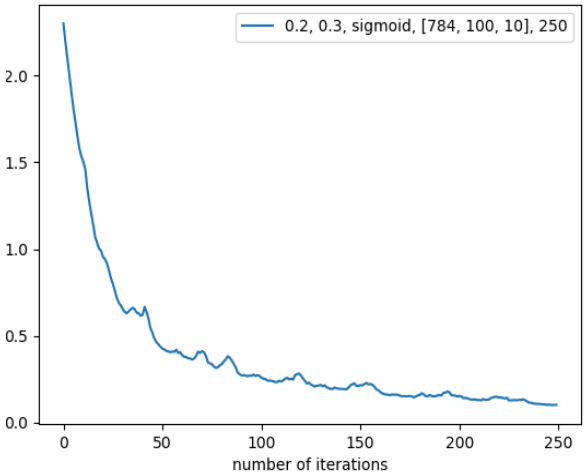
acc on training set= 0.978

acc on cross validation set= 0.918

Lambda = 0.2

acc on training set= 0.9756666666666667

acc on cross validation set= 0.924



Lambda =0.3

acc on training set= 0.9800000000000007

acc on cross validation set= 0.921

Lambda=1

acc on training set= 0.9791666666666666

acc on cross validation set= 0.92

Lambda = 5

acc on training set= 0.9686666666666667

acc on cross validation set= 0.916

The above observations suggest that even after introducing regularization, the accuracy did not increase much, the maxima was achieved at Lambda=0.2, cross validation set accuracy = 92.4

As we increase the lambda from 0 to 0.3, the training and cross validation set accuracy go up marginally, but beyond that the training set accuracy remains fairly same or decreases and the cross valid. Set accuracy starts to decrease as the model now starts to underfit the data because of very large penalty. And because we have reached an accuracy of about 98% on the training data, we say that this model has reached it's capacity , because going beyond this training set accuracy increases the chances that we have overfit the training data.

Hence we conclude we can't do much better than ~92.5% accuracy on this network architecture in 250 iterations, so we try to change our network architecture now in a hope to get better accuracy on both the sets of data.

The general approach from this point onward will be as follows :

Start from a basic model with low learning rate ➔ may need larger learning rate/number of iterations for convergence to a good estimate of parameters ➔increase learning rate till acc starts to decrease ➔ may over fit as complexity increased ➔ introduce regularization parameter to prevent overfitting➔ keep tuning till we can't do any better in a certain number of iterations ➔ Increase the model complexity/capacity ➔ REPEAT

We try to cap number of iterations at around 300 as larger number is practically infeasible.

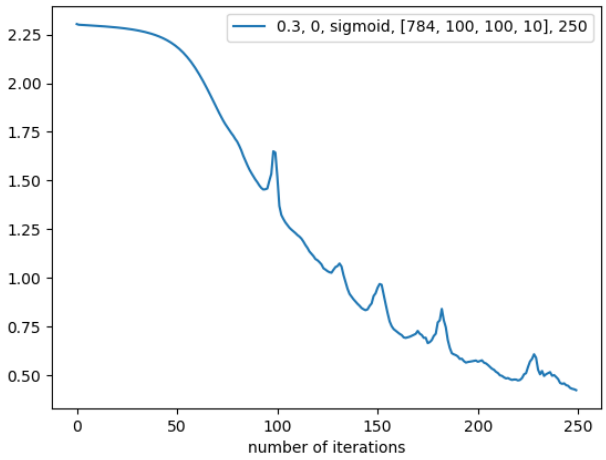
We now increase the model capacity by increasing the number of hidden layers

The following observations are made for 2 hidden layers with 100 neurons each

rate =0.2, Lambda =0

acc on training set= 0.8545

acc on cross validation set= 0.82



rate = 0.3, Lambda = 0

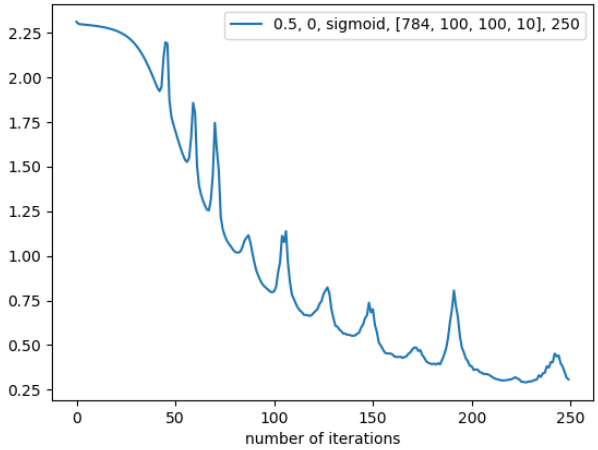
acc on training set= 0.9043333333333333

acc on cross validation set= 0.868

rate = 0.5, Lambda =0

acc on training set= 0.9226666666666666

acc on cross validation set= 0.9



rate =0.5, Lambda =0, Num_iter = 300

acc on training set= 0.9528333333333333

acc on cross validation set= 0.903

We observe that the graphs start to become very irregular, this means that our learning rate is too high, but if we decrease our learning rate, then accuracy decreases on the same number of iterations. Even if we pump up the number of iterations to 300, no significant improve is seen in the accuracy. Hence we are trapped by the problem of the vanishing gradient. We don't try the regularization parameter as there is no question of overfitting as train set accuracy is not much.

Hence we introduce the ***RELU*** activations function in our network, as the learning curve of the sigmoid function is slow and gets saturated quickly, however in RELU the slope is 1 for all positive values, hence this problem of *vanishing gradient* is solved this way. This gives faster learning in the same number of iterations.

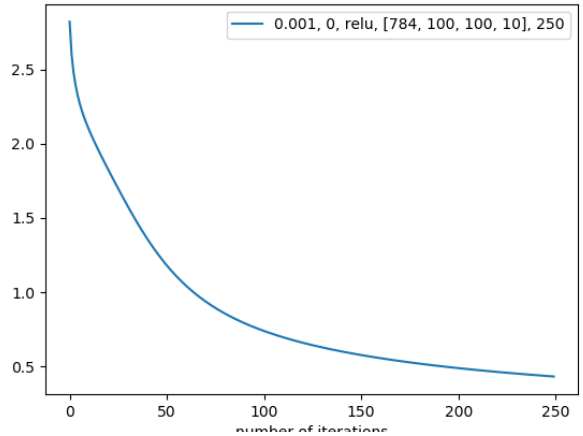
The following are observed when we use *relu* as our activation function in the same network.

rate= 0.001, Lambda =0, num_iter =250

acc on training set= 0.8753333333333333

acc on cross validation set= 0.874

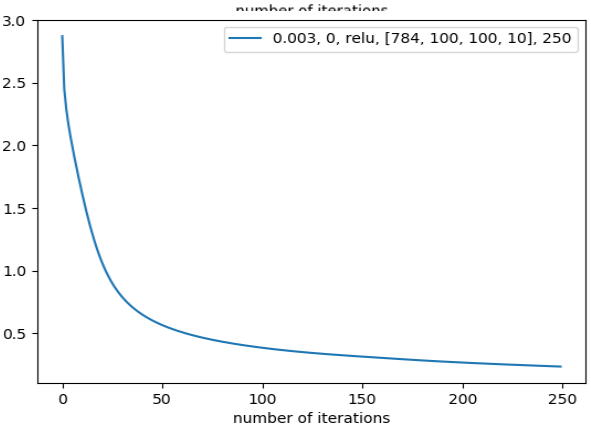
We observe that at a much lower learning rate than previous model, we can increase a good enough accuracy and we observe a very smooth graph, hence we try to increase the learning rate to get better accuracy on both sets.



rate=0.003, Lambda=0,num_iter=250

acc on training set= 0.936

acc on cross validation set= 0.918



rate=0.008, Lambda=0,num_iter=250

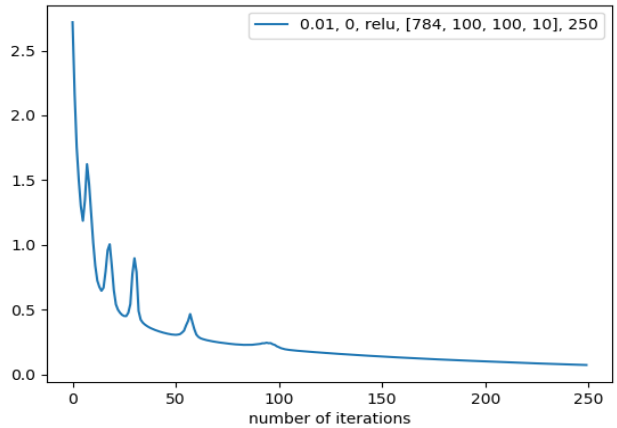
acc on training set= 0.9736666666666667

acc on cross validation set= 0.931

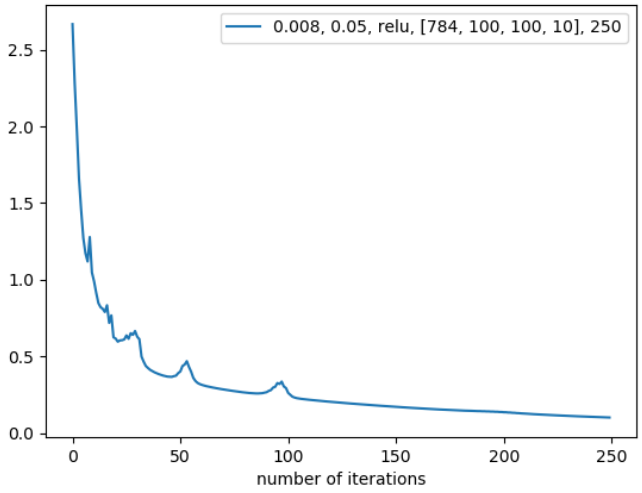
rate=0.01, Lambda=0,num_iter=250

acc on training set= 0.9835

acc on cross validation set= 0.934



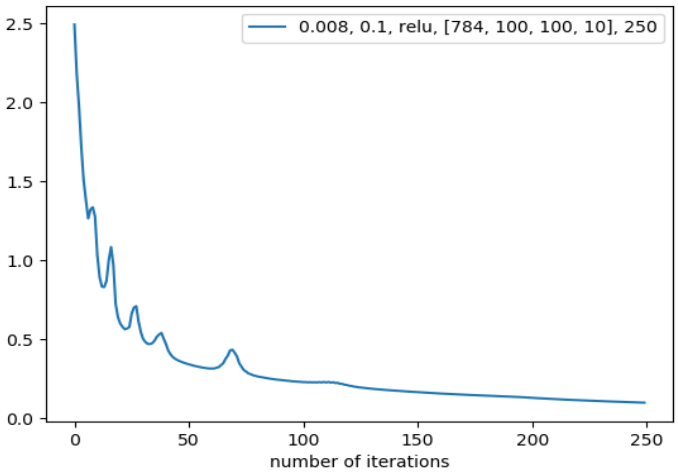
Again wee see that, the accuracy on the training set has reached a saturation level, and we can't do better in 300 iterations, so we try regularization, to improve cross validation set accuracy, ***starting with the same learning rate as above*** *without bringing down training set accuracy.*



rate=0.08, Lambda = 0.05

acc on training set= 0.975

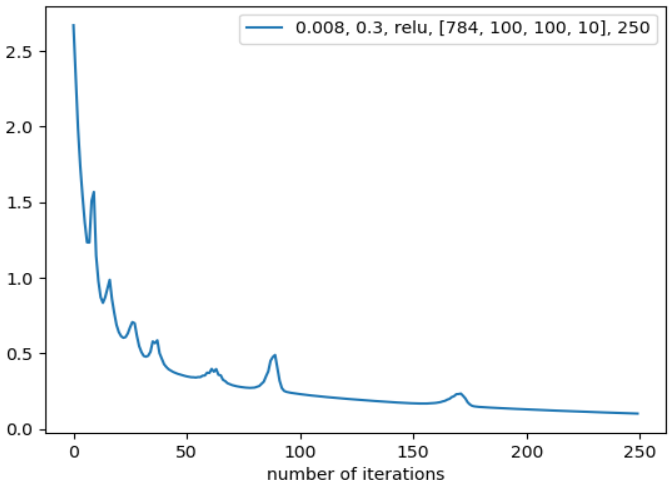
acc on cross validation set= 0.926



rate=0.08, Lambda = 0.1

acc on training set= 0.9765

acc on cross validation set= 0.942



rate=0.08, Lambda = 0.3

acc on training set= 0.9738333333333333

acc on cross validation set= 0.938

Since, the sweet spot occurred at lambda = 0.1 and rate =0.08, we try to run it for 300 iterations in a hope for better performance.

acc on training set= 0.9833333333333333

acc on cross validation set= 0.942

Hence, we see that the cross validation set accuracy has saturated, this gives us a view that our current model has "probably" with a good confidence, reached it's capacity, hence we again increase model complexity by adding one more layer with 100 neurons.

The following were done with 3 hidden layers of 100 neurons each.

rate=0.001, Lambda = 0

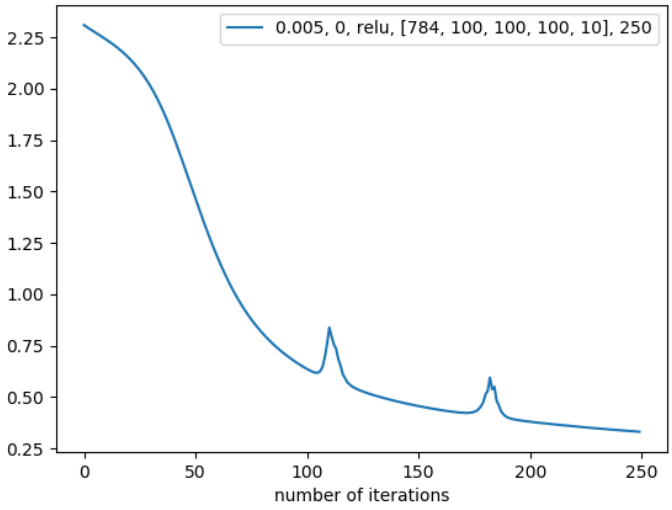
acc on training set= 0.6183333333333333

acc on cross validation set= 0.623

rate=0.005, Lambda = 0

acc on training set= 0.9076666666666666

acc on cross validation set= 0.895



rate=0.01, Lambda = 0

acc on training set= 0.9448333333333333

acc on cross validation set= 0.92

rate=0.05, Lambda = 0

acc on training set= 0.8496666666666667

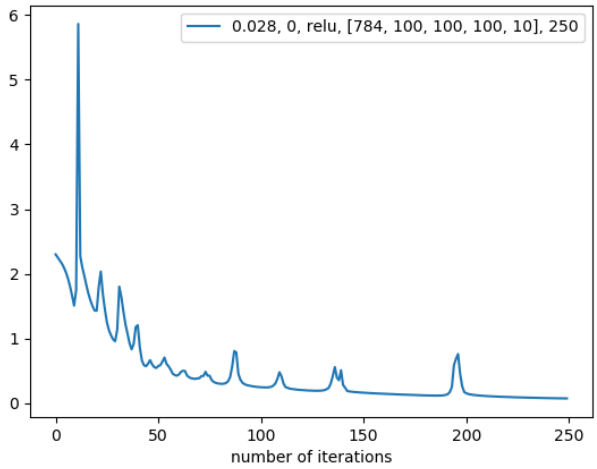
acc on cross validation set= 0.831

Now reduce learning rate from 0.05 to 0.01

rate=0.04, Lambda = 0

acc on training set= 0.969

acc on cross validation set= 0.935



rate=0.03, Lambda = 0

acc on training set= 0.9703333333333334

acc on cross validation set= 0.938

rate=0.02, Lambda = 0

acc on training set= 0.9691666666666666

acc on cross validation set= 0.935

Now increase learning rate from 0.02 to 0.03

rate=0.023, Lambda = 0

acc on training set= 0.9745

acc on cross validation set= 0.946

rate=0.026, Lambda = 0

acc on training set= 0.974333333333333

acc on cross validation set= 0.944

rate=0.028, Lambda = 0

acc on training set= 0.9808333333333333

acc on cross validation set= 0.942

Hence, we conclude that at rate = 0.023, we get best accuracy, now we try to prevent overfitting as we increase number of iterations from 250 to 300.

rate=0.023, Lambda = 0.05, num_iter=250

acc on training set= 0.9743333333333334

acc on cross validation set= 0.939

rate=0.023, Lambda = 0.05, num_iter=300

acc on training set= 0.9885

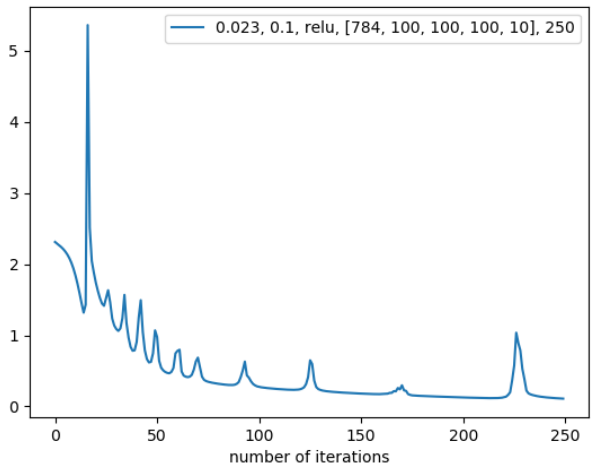
acc on cross validation set= 0.939

model acc doesn't increase on increasing num_iter ➔ Increase lambda

rate=0.023, Lambda = 0.1, num_iter=250

acc on training set= 0.9708333333333333

acc on cross validation set= 0.937



Looking at the graph, we understand that the learning rate maybe a little too high, so we decrease the learning rate to avoid the spikes.

rate=0.02, Lambda = 0.05, num_iter=250

acc on training set= 0.9745

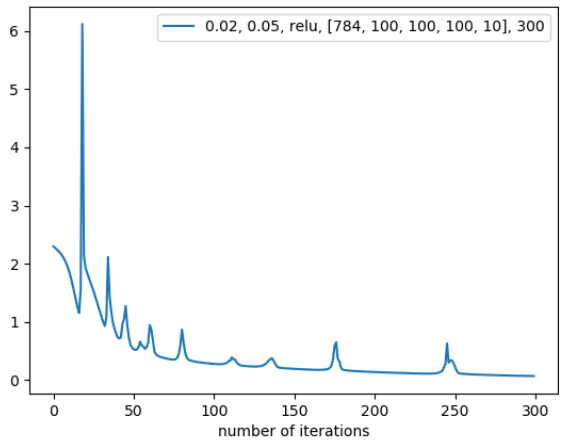
acc on cross validation set= 0.941

Accuracy on cross validation ➔ try increasing the number of iterations

rate=0.02, Lambda = 0.05, num_iter=300

acc on training set= 0.9815

acc on cross validation set= 0.94



Again increase lambda

rate=0.02, Lambda = 0.08, num_iter=300

acc on training set= 0.9808333333333333

acc on cross validation set= 0.945

Decrease Lambda

rate=0.02, Lambda = 0.07, num_iter=300

acc on training set= 0.9838333333333333

acc on cross validation set= 0.948

rate=0.02, Lambda = 0.06, num_iter=300

acc on training set= 0.98

acc on cross validation set= 0.936

Hence, we see that the sweet spot for this network occurs at rate=0.02, Lambda = 0.7 with a cross validation set accuracy of about 95%.

Till now we have been using full batch gradient descent, we now see the effect of batch size on the model performance in terms of training time and accuracy

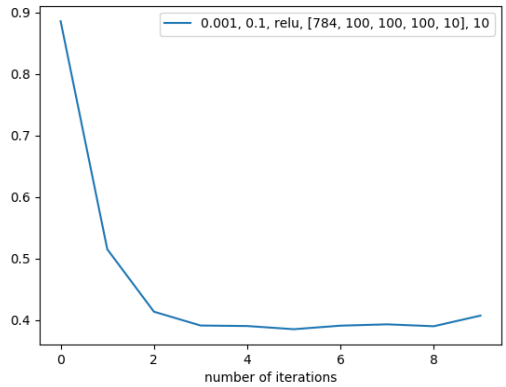
We do the following for three hidden layers of 100 neurons each :

Batch_size=1, rate= 0.001, epoch=10

acc on training set= 1.0

acc on cross validation set= 0.904

We have overfit the data here. As we have run it effectively for 6000*10 iterations of weight updates which is very large, we have kept learning rate low as otherwise the update gets very irregular as we are making updates after each data point.



Batch_size=50, rate= 0.001, epoch=10

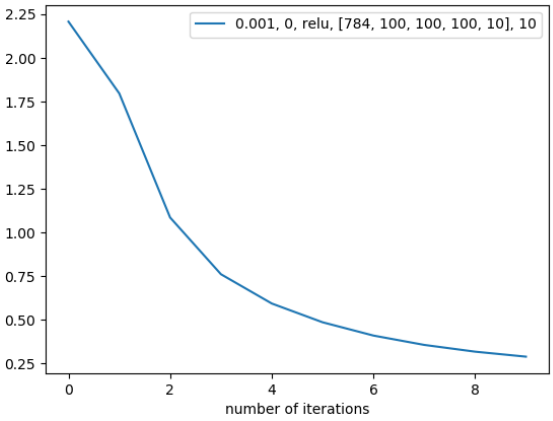
acc on training set= 0.92

acc on cross validation set= 0.891

Batch_size=50, rate= 0.005, epoch=10

acc on training set= 0.98

acc on cross validation set= 0.931



Since, the updates are made in large batches now, so the updates are made for lesser number of times, hence we don't end up overfitting the training set. So, we try to increase the learning rate and we see that the accuracy on both the sets as the updates are made in large batches.

We now try to see learning with learning rate of 0.001 and 10 epochs

Batch_size=100, rate= 0.001, epoch=10

acc on training set= 0.87

acc on cross validation set= 0.837

Batch_size=200, rate= 0.001, epoch=10

acc on training set= 0.65

acc on cross validation set= 0.704

Batch_size=500, rate= 0.001, epoch=10

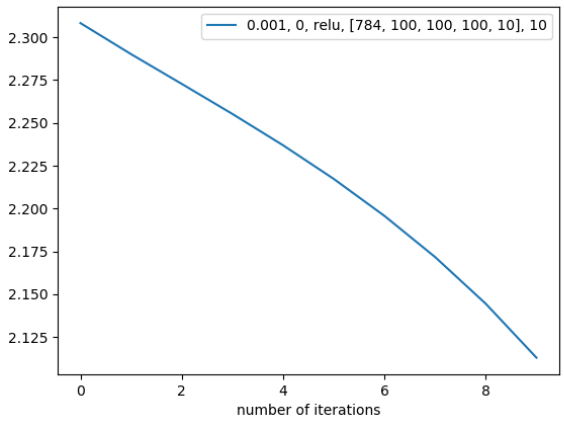
acc on training set= 0.41

acc on cross validation set= 0.411

Batch_size=1000, rate= 0.001, epoch=10

acc on training set= 0.269

acc on cross validation set= 0.275



We see here, that for a given learning rate, and epochs, as we increase the batch size, the training and cross validation set accuracy decreases. This is because the weight updates happen for a larger number of times. Also, we see that for very small batch size we end up overfitting the data while at very large batch size accuracy is very low to increase which we require more epochs and hence more training time. So there exist a sweet spot somewhere in between which allows us good accuracy in less running time.

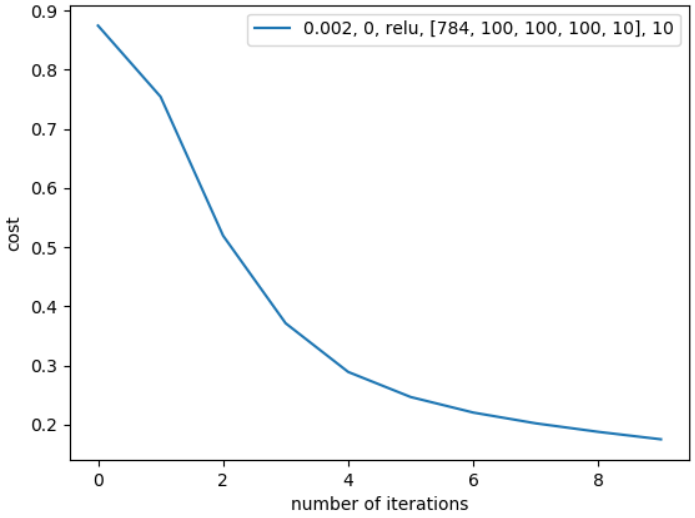
We now compare the performance if our model with cross entropy v.s. MSE loss functions

We prefer cross entropy loss for the multi-class classification problem, as it tries to minimize the Shannon-entropy, which penalizes misclassification highly, hence is suitable for classification problems. However the MSE error is preferred for regression purposes as close approximation or neighborhood of the exact value is also accepted which is not so in classification. We demonstrate this below.

MSE loss, 3 hidden layer 100 neurons each, batch_size=100, epoch=10

acc on training set= 0.88

acc on cross validation set= 0.877



Cross Entropy loss, 3 hidden layer 100 neurons each, batch_size=100, epoch=10

acc on training set= 0.92

acc on cross validation set= 0.906