Report for Assignment – 2

Sonu Dixit

SR – 14432

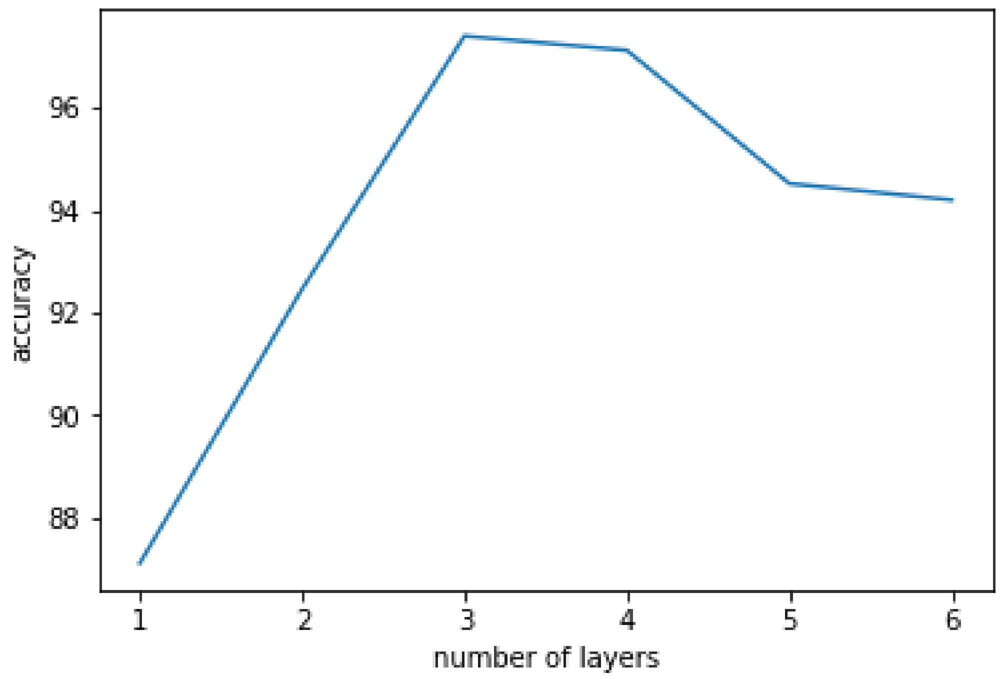
Python 3

Part 1: Implementation of MLNN and Experimentation on MNIST

Implementation Details:

1. Batch Stochastic Gradient for optimization
2. Every layer can have different learning rate
3. Use of momentum for gradient smoothening
4. Gradient clipping
5. Supported initialization: zeros, normal, Xavier(glorot normal), uniform
6. Supported activations: sigmoid, tanh, softplus, swish, relu, softmax(only at the end layer)

TASK 1 (experiment with different number of layers):



I have experimented with (1,2,3,4,5,6) hidden layers for MNIST dataset. In each layer I have tried (20/30) nodes. Best accuracy I got at 2 and 3 hidden layers. After 3 hidden layer there is no increase in accuracy. Model is overfitting. Training loss decreases but validation loss increases.

TASK 2 (various configuration for number of hidden nodes):

I found 3hidden layer with each having 30 hidden nodes as best performing.

TASK 3 (specific configuration(sigmoid,tanh,relu,swish)):

Not any improvement, but rate of decrease of loss is less as compared to tanh,relu,relu,relu. I think this is because tanh, swish are at the final layers, and for these the gradient value is less as compared to relu. I think, there is hindrance in gradient flow. Here I need 25 epochs with learning rate 0.001 to get 94.85 accuracy. Whereas in the case of relu, only 10 epochs is good enough to get 95% accuracy.

Task 4(initialization techniques):

I have found Xavier initialization(glorot normal in keras) as the best performing initialization. Its picking up weights from truncated normal distribution where std depends on number of weights to be initialized. Since in standard normal, std is always fixed, so Xavier performs better than stand. Normal.

Task5(Keras Library Comparision):

1. It requires very less time as compared to my implementation. They have implemented in terms of whole batch and parallelized the processing in different batches. I am doing sequential calculation, summing up gradients of a batch and then update. The keras implementation is more vectorized and optimized than mine and hence requires less time.
2. With keras I got 98% accuracy with 2 hidden layers having 30 nodes each with activation (relu, relu). In my implementation for 2 hidden layers I have got ~92% accuracy.

PART TWO CAT_DOG datset:

for cat and dog data set multilayer network is not at all performing. its performance is just as good as the random classifier that will predict either 0 or 1 randomly or putting up everyone as 1 or 0. From my implementation, I am getting all the results as either 1 or zero. Accuracy is around 50%. For my implementation I reduced the image size from 200 to 50X50 using cv2 library, since 40000 dimensional vector processing was taking a lot of time.

When I tried with Keras implementation, there also, the accuracy remains stuck at 49~51%. I have tried with full 200 x 200, as well as 100x100(using cv2 library to resize),80x80, 50x50x32x32…nothing improved the results.

The reason that MLNN is unable to perform well on this cat and dog data set is because cat and dog dataset is image dataset and the simple neural network is unable to understand the spatial structure in the image. Classifying image requires spatial structural information. Networks needs to be understood neighboring properties of pixel to group them together as a characteristic of cat or dog. Simple NN is unable to capture that. Although CONV nets are a special kind of MLNN with shared weights and linear(at conv layers) activation, they are able to extract this spatial information in the image. Because of shared weights.

MLNN has been able to perform well on MNIST, because here the data is digits, which are nopt as complex as cat or dog image. Digits are made up of simpe continuous kinda of sketch. So MLNN is able to perform well.

PART #3: performance on dolphins, pubmed, twitter dataset

Dolphins dataset:

No improvement in result by MLNN. With NN classifier I could get 100 % accuracy. Here with 2 layer NN(30,4) I could get 95% accuracy. Reason I think, 62 samples are not large enough data to train a neural net.

PUBMED dataset:

NO significant improvement. Bayes classifier was giving 43% accuracy. Here with 2 layer network I am getting similar ~44 % accuracy. Here also I think for a 128 dimensional data 12k data points are not good enough. I tried with PCA as well but I have not observed any improvement. Deeper Network overfits, training loss decreases but test accuracy decreases as well.

TWITTER dataset:

No significant improvement.

Its 2845(preprocessed using Bag of words model) dimensional dataset. Total data points are only 6000. I tried with PCA by reducing data to 100 dimension, but still accuracy is around 56%. Bayes classifier gives 57% accuracy for this dataset in assignment 1.

I think using word embeddings in this case should increase the accuracy.