CS 260 HW3

Shruti Sharan UID:405228029

March 15th, 2019

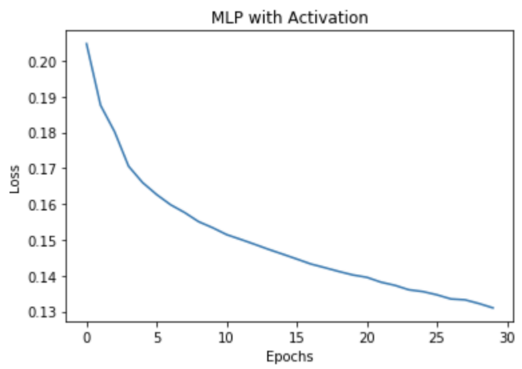
We use CIFAR10 classification dataset to perform multi class classification of objects.

1 Fully Connected Layer

I implemented a 7 layers fully-connected neural network with ReLu activation function. To achieve high accuracy I decreased the number of channels in each layer, going from 32*32*3= 3072 down to 10(No of output classes). The network architecture is as follows:

Layer	Input Dimension	Output Dimension
Fully Connected 1	3072	‘1024
Fully Connected 2	1024	512
Fully Connected 3	512	256
Fully Connected 4	256	128
Fully Connected 5	128	64
Fully Connected 6	64	32
Fully Connected 7	32	10

I used a learning rate of 1e=4 with the Adam Optimizer and the Cross Entropy Loss function to compute the loss. Plotting the average loss per Epoch over all the epochs.



Running it for 30 epochs, I was able to achieve a test accuracy of **52%**

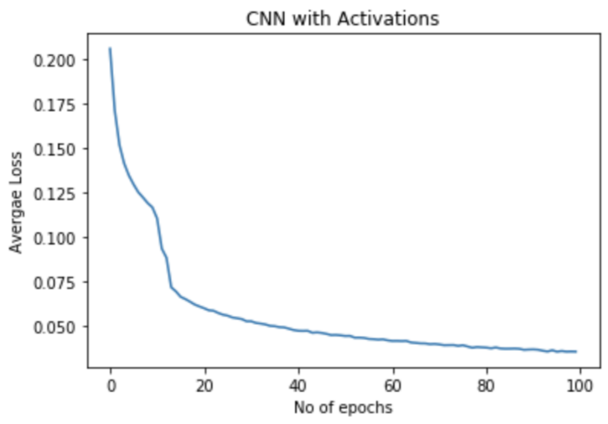
2 Convolution Neural Network

I implemented a 7 layers model with 4 convolution neural networks followed by 3 fully-connected layers, with ReLu activation function. The network architecture is as follows:

Layer	Input Channel	Output Channel	Kernel Size	Stride	padding
Convolution 1	3	64	3	1	1
Convolution 2	64	50	3	1	1
Convolution 3	50	40	5	1	2
Convolution 4	40	16	5	2	2

Layer	Input Dimension	Output Dimension
Fully Connected 1	4096	512
Fully Connected 2	512	256
Fully Connected 3	256	10

The output from the Convolution Neural Network is flattened out before feeding it into the Linear layer.I used a learning rate of 3e-4 with the Adam Optimizer with weight decay= 5e-4 and the Cross Entropy Loss function to compute the loss. GPU training was used with CUDA as back-end.Running it for 100 epochs, I was able to achieve a test accuracy of **85%**
The Average loss per Epoch is plotted across all Epochs.

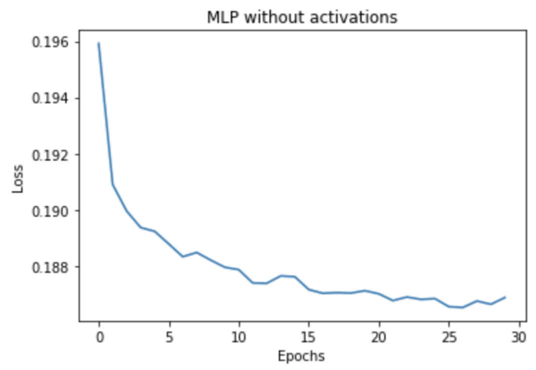


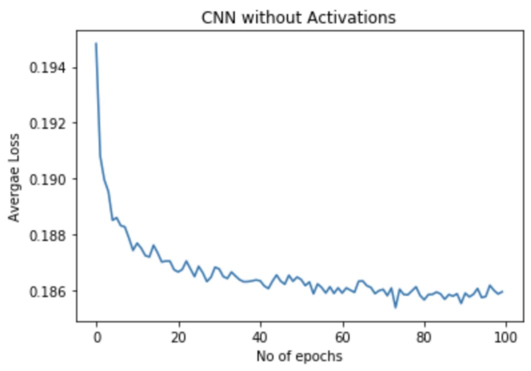
3 Comparison and Experimentation

Comparing the two models we see that the Convolution Neural Network does much better than the simple Multi-layer model. This is because, in fully connected layers, each perceptron is connected with every other perceptron which leads to a very high dimensionality. That is, the number of total parameters can grow to very high number. This is inefficient because there is redundancy in such high dimensions. Another disadvantage is that it disregards spatial information as it takes flattened vectors as inputs. Convolution Neural Networks on the other hand can account for local connectivity (each filter is panned around the entire image according to certain size and stride, allows the filter to find and match patterns no matter where the pattern is located in a given image). It takes matrices as well as vectors as inputs. The layers are sparsely connected or partially connected rather than fully connected. Every node does not connect to every other node. Thus it gives us much better results, as it allows parameter sharing, weight sharing so that the filter looks for a specific pattern, and is location invariant.

Without Non-Linear Activations:

The Loss per epoch for the MLP and CNN without None-Linear Activation are plotted as follows:





We train both the models with the same specifications as mentioned above, without the activation function(Relu)and the accuracy drops considerably to 37% for MLP and 38% for CNN respectively. A non linear Activation is used to generate non-linear mappings from inputs to outputs so that the neural-networks can compute and learn different functions. When we do not have the activation function the weights and bias simply do a linear transformation. A linear equation is simple to solve but is limited in its capacity to solve complex problems like image classification. Hence the accuracies drop.
We report the overall Average loss for each of the models with the corresponding test accuracies as follows.

Model	Average Loss	No of Epochs	Test Accuracy
MLP	0.150	30	52%
MLP without Relu	0.188	30	37%
CNN	0.056	100	85%
CNN without Relu	0.186	100	38%

Experimentation:

For the CNN model I tried out various architectures starting with high number of channels and resizing the dimensions to get an improved accuracy but it kept getting stuck at 74%. I changed the kernel sizes and adjusted the stride and padding to get better results. All of these were giving poor results as I was training on CPU with low number of epochs. After running it on GPU (Google Cloud) for larger number of epochs (100), I was able to achieve the desired accuracy.