## horizontal line

Technical challenge

**─**

Ankit Bagde

Department - Computer Science & Engineering

Course / CGPA - Dual Degree / 7.79

College - IIT Kharagpur

Graduating year - 2022

Phone no. - 7069454463

Email id - ankit2000.bagde@gmail.com

# The challenge

Train a very large neural network, then make it very small through weight pruning and unit/neuron pruning.

# Goals

1. To study the effect of pruning on the accuracy of the neural network graphically.
2. Compare two types of pruning namely weight pruning and unit/neuron pruning

# Theory

The two types of pruning to be studied are discussed below:

## Weight Pruning

Basically, the weights whose absolute value lie close to zero can be pruned away resulting in a ‘sparse’ graph. It, in turn, affects the accuracy of the neural networks, which is discussed in the observation section.

To achieve %K sparsity, we need to sparse k% of total weights (excluding the last layer weights) which is achieved in the following way theoretically:

* Sort all the weights as per their absolute values.
* Set the least K% of the total weights to zero.

## Unit/Neuron Pruning

In the above method, weights were set as zero (or technically, links in neural networks were removed) whereas in the neuron pruning as the name suggests, the neuron is pruned away resulting in a neural network with a fewer number of neurons.

Eg - In hidden layer 1, there were 1000 neurons, after pruning it became 985.

In this case, %k sparsity means %k of total neurons are deleted (excluding the last layer, as it’s in the form of the output) which is achieved in the following way theoretically :

* In the weight matrix, each column corresponds to a neuron.
* Sort all the columns of the weight matrix as per their L2-norm.
* Set %k columns to zero, resulting in the pruning of %k neurons.

# My Approach and observations

I have used keras to solve the challenge. After training the neural network, I achieved an accuracy of 97.81% on the test set after 5 epochs

## Weight pruning

The following steps are followed :

* model.get\_weights() is used to extract all the weights of the layers in the form of the numpy array (WEIGHT\_MATRIX) in the following way:
  + WEIGHT\_MATRIX [0] - contains a weight matrix of the input layer x hidden\_layer 1.
  + WEIGHT\_MATRIX [1] - contains a ‘biases’ row vector of hidden layer 1
  + Similarly, to access weight matrix in WEIGHT\_MATRIX [N], N = 0,2,4,6,8 and to access biases vector N=1,3,5,7
* A dict namely COMPLETE\_MATRIX is created in which key is (N, row, column), where N is the indices of the WEIGHT\_MATRIX, row x column corresponds to the position of a particular weight.
* Dict is sorted and is stored in **‘sort’,** which is an array, description of which is given below :

sort will be of the form ((a,b,c), weight) in ascending order of weights where,

a ---> layer i.e. 0,2,4,6,8

b ---> row

c ---> column

weight ---> weight of the corresponding neuron

sort can be accesssed in the following ways :

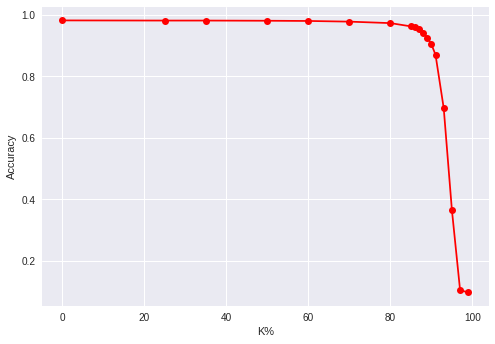
Eg. if a = [((3, 190, 156), 4.8154054), ((2, 190, 156), 4.8154054)]

a[0] ---------> ((3, 190, 156), 4.8154054)

a[0][1] ------> 4.8154054

a[0][0] ------> (3, 190, 156)

a[0][0][1] ---> 190

* .set\_weights() is used to set the weights from a given list of numpy arrays to same shape as of obtained from .get\_weights()
* k = [0, 25, 35, 50, 60, 70, 80, 85, 86, 87, 88, 89, 90, 91, 93, 95, 97, 99] is taken in our case, for each case, least k% values in WEIGHT\_MATRIX is set to zero and accuracy is calculated and the graph obtained is shown below:

**Figure - 1**

## Unit/ Neuron pruning

The same process is followed in Neuron pruning, with a slight difference while creating the dict COMPLETE\_MATRIX.

* In this case, the key value is (N, column) where N is the indices of the WEIGHT\_MATRIX, a column corresponds to the column vector index of a particular neuron
* Dict is sorted and is stored in **‘sort’,** which is an array, description of which is given below :

sort will be of the form ((a,b), c) in ascending order of weights where,

a ---> layer i.e. 0,2,4,6,8

b ---> column

c ---> L2-norm of the column vector

sort can be accessed in the following ways :

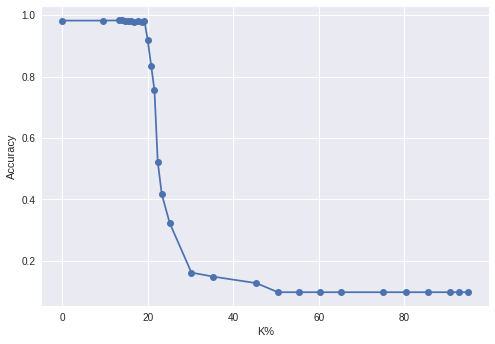
Eg. if a = [((3, 190), 4.8154054), ((2, 190), 4.8154054)]

a[0] ---------> ((3, 190), 4.8154054)

a[0][1] ------> 4.8154054

a[0][0] ------> (3, 190)

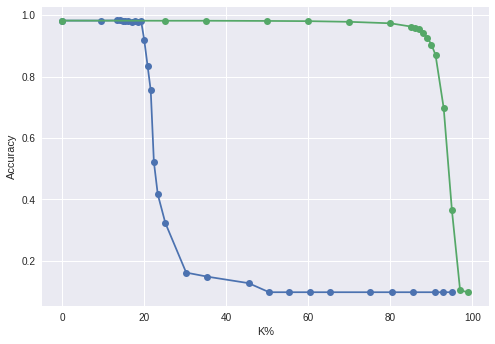
* .set\_weights() is used to set the weights from a given list of numpy arrays to same shape as of obtained from .get\_weights()
* k\_for\_neurons = [0, 10, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 35, 40, 50, 55, 60, 65, 70, 80, 85, 90, 95, 97, 99] is taken in our case, for each case, k% columns with leasr L2-norm in WEIGHT\_MATRIX is set to zero and accuracy is calculated and the graph obtained is shown below:



**Figure - 2**

### Weight Pruning vs Unit/Neuron Pruning

In the unit pruning, the k\_for\_neurons represents the % of the neurons not the weights. The corresponding\_wt array contains the %weights corresponding to the % neurons and the following plot is obtained against the accuracy :



**Figure -3**

**Note - In the above graph K represents the % weights, whereas the graph under the Unit/Neuron pruning heading K% represents the % neurons pruned.**

Now, its quite obvious that accuracy will decrease earlier for unit pruning as compared to weight pruning because, in neuron pruning %20 means 20% of neurons will be pruned, and each neuron corresponds to weights equal to the number of neurons in that particular layer.

But with the above argument, the graph with **Figure - 2** should resemble with **Figure - 1,** but I think the reason behind their non-resemblance is the process for both is quite different. In the weight pruning, the weights with least absolute value is set to zero whereas in the unit pruning L2 - norm is used as a factor to prune the neurons.



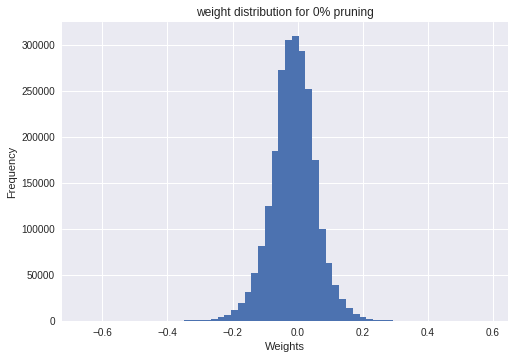


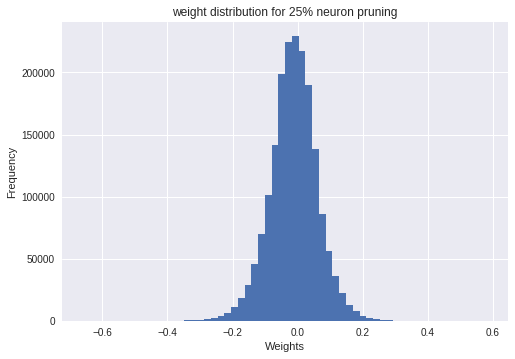
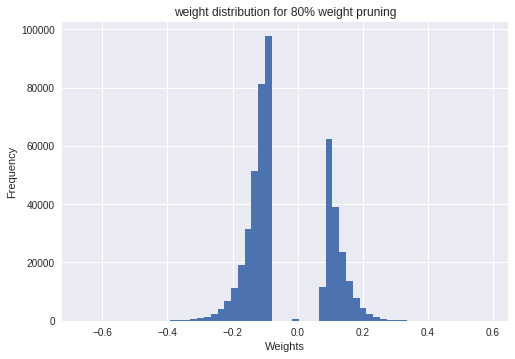
For the L2 norm is given by thus here lies the difference

between the two processes.

### Graphical understanding of two processes :

Firstly, I observed the graph of weight distribution of the complete neural network, which is shown below :



Then, I plotted two graphs each for weight pruning and neuron pruning, shown as follows:

Observations:

* In case of wt pruning, wts near abs value zero are pruned
* In case of neuron pruning, the difference in frequency can be observed as compared to the graph given above with 0% pruning.

The difference between the two types of pruning can be thus clearly observed through the graphs.