CSE343/ ECE363/ ECE563: Machine Learning W2021
Assignment-2

Navneet Agarwal
2018348

May 1, 2021

Programming Questions

1) Backpropagation

5.

(a)

Accuracy are reported in Table 1

(b)

The loss vs epochs plots are in the Figure 1 and 2.

(c)

In every case the activation function of the output layer was Softmax for every case. Since we are using cross-entropy loss as our loss function, we need our Neural Network to predict probabilities. The probabilities could also have been calculated directly by normalization but that does not give correct interpretation. eg. if predicted values for two samples were {1,2,3} and {101,102,103} the normalization would give probabilities as {0.166, 0.333, 0.5} and {0.33, 0.33, 0.33} respectively. We can see that both the samples had same difference between the predicted values but the predicted probabilities does not interpret this information correctly for the second sample. However in the case of softmax function both the samples will have same predicted probabilities conveying the correct information. Another point to consider Softmax function is because of the derivative of cross entropy wrt softmax function,i.e. y_pred - y_orig.

(d)

Total Number of layers = 5
Total Number of Hidden layers = 3

(e)

The best results came out for ReLU activation function.The tSNE plot for the last hidden layer can be seen in Figure 3.

6.

The results are in Table 1. There are very minute differences between the results of MLP classifier and my neural network implementation.

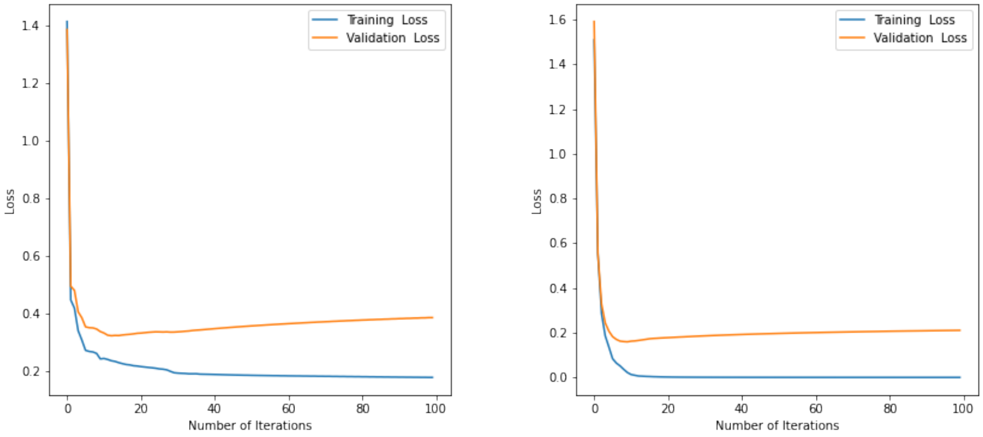


Figure 1: My NN :Linear(left) and Tanh(right) Activation : Loss vs epoch plots

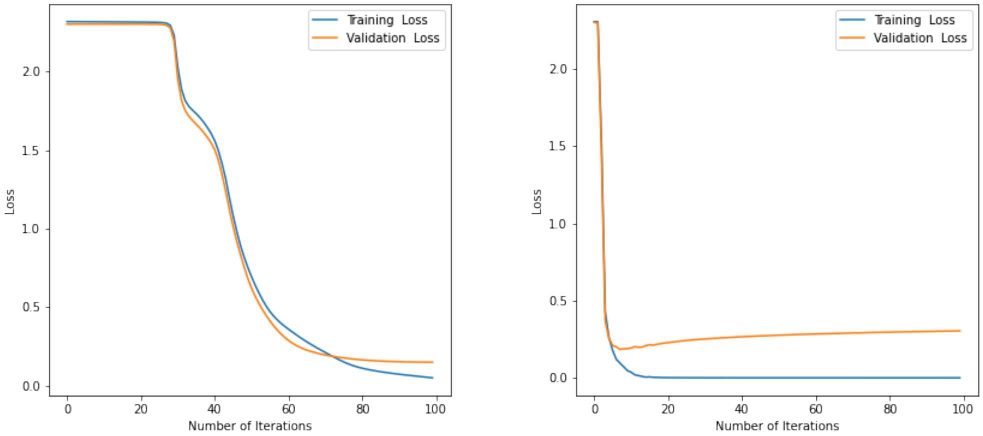


Figure 2: My NN :Sigmoid(left) and Relu(right) Activation : Loss vs epoch plots

Model	**Activation**	**Training Accuracy**	**Validation Accuracy**	**Testing Accuracy**
My NN	Linear	93.29	90.74	91.05
MLP	Linear	93.80	91.06	91.97
My NN	Tanh	100	96.41	96.75
MLP	Tanh	99.99	96.65	96.72
My NN	Sigmoid	99.75	95.96	95.95
MLP	Sigmoid	99.99	96.76	96.66
My NN	Relu	100	96.79	97.05
MLP	Relu	100	96.91	97.33

Table 1: Results for MLP and My NN

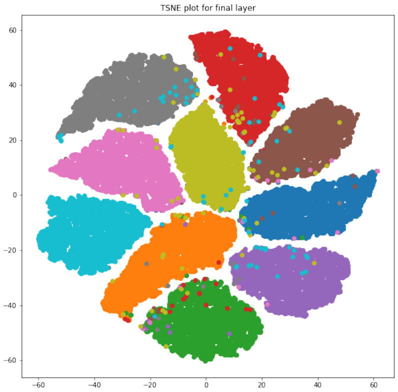


Figure 3: My NN : tSNE plot

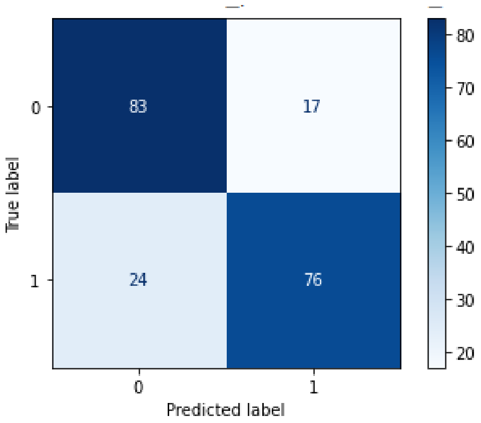


Figure 4: Confusion Matrix for text Classification

3) Text Classification

The python notebook for this question is in Code folder with name **Q3.ipynb**
Apart from the preprocessing steps mentioned in the questions , Porter stemmer was also applied on all the text data.The Grid search results are attached and can be found in the results folder with name ”ques3_grid.csv”. Many grid searches were done, one of them is in the file mentioned above.
The best parameters were : **kernel** = sigmoid , ”C” = 5 and **gamma** = auto. For the best hyper-parameters the model achieved an **Accuracy** of 79.5% and **F1 score** of 78.76% . The Confusion matrix is given in this figure 4

Class	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Accuracy(%)	88.5	96.9	83.9	67.9	84.9	83.7	93.4	86.2	91.3	90.4

Table 2: VGG16: Class wise Accuracy

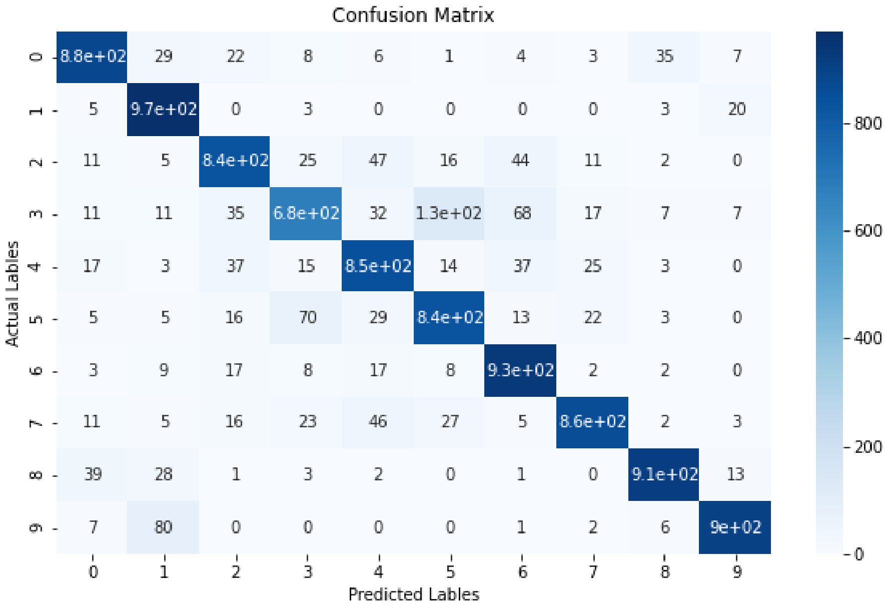


Figure 5: Confusion Matrix for CIFAR-10

4) Extra Credit Question

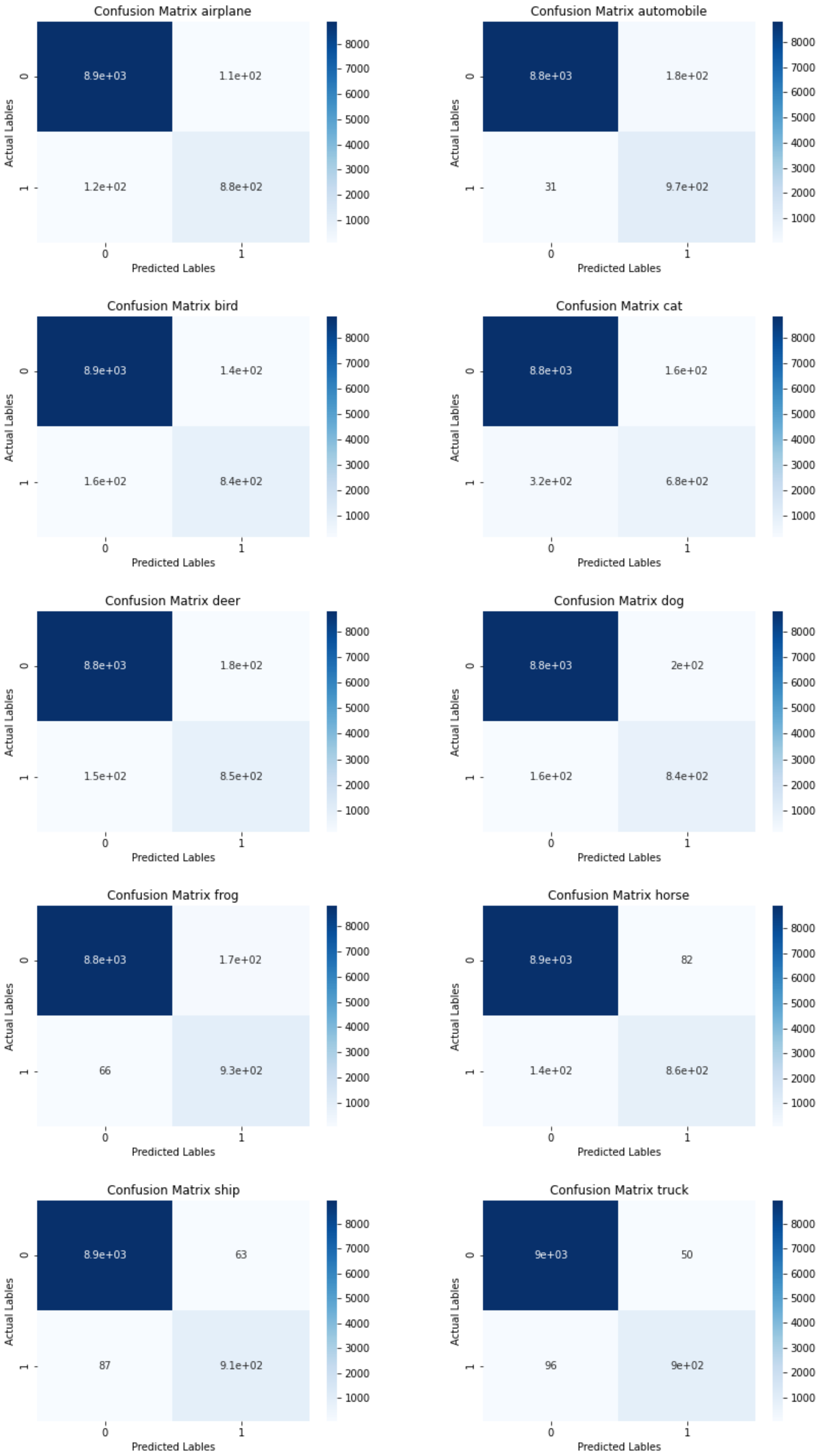
The python notebook for this question is in Code folder with name **Q4.ipynb**The fine-tuned model achieved an **Accuracy** of 86.71% on test data . The Confusion matrix is given in this figure 5

The class wise accuracy are given in Table 2.The confusion matrix for every class is given in Figure 6 On observing the class-wise accuracy and the confusion matrix we infer that the Class Cat is the most mis-classified class. This might be happening due to either due to unfortunate train-valid split or the model is not robust enough to classify any minor changes in the cats images. Rest all the classes have a decent performance.

I stopped training my model as soon as validation loss started increasing and the validation loss continued dipping to prevent over-fitting.The validation and training loss can be sen in figure 7.The saved model has size more than 500 MB, so the google drive to access it is this. Due to computation limitations on google collab I set the batch size to 128 and learning rate to 0.0001. The loss function was chosen to be cross-entropy loss and the optimizer was chosen to be Adam as they are the most popular choices for training models.

The weights of all the lower convolutional layers were frozen. In the original model the classifier had following layers:

1. Linear Layer(input_features=25088, output_features=4096)
2. ReLU Layer
3. Dropout Layer with probability = 0.5
4. Linear Layer (input_features=4096, output_features=4096)
5. ReLU Layer
6. Dropout Layer with probability = 0.5
7. Linear Layer (input_features=4096, output_features=1000)



Confusion Matrix airplane

Confusion Matrix automobile

Confusion Matrix bird

Confusion Matrix cat

Confusion Matrix deer

Confusion Matrix dog

Confusion Matrix frog

Confusion Matrix horse

Confusion Matrix ship

Confusion Matrix truck

Figure 6: Class wise Confusion Matrix

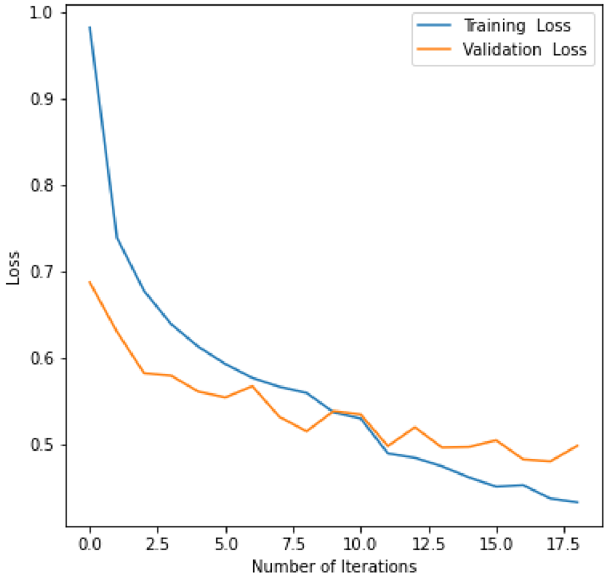


Figure 7: VGG16 : Loss vs epoch graph

The final layer should have output features = 10 . So I modeified the classifier layers as follows:

1. Linear Layer(input_features=25088, output_features=4096)
2. ReLU Layer
3. Dropout Layer with probability = 0.5
4. Linear Layer (input_features=4096, output_features=4096)
5. ReLU Layer
6. Dropout Layer with probability = 0.5
7. Linear Layer (input_features=4096, output_features=1024)
8. ReLU Layer
9. Dropout Layer with probability = 0.5
10. Linear Layer (input_features=1024, output_features=10)

The layer 7 was modified to have output features as 1024 and then ReLU and Dropout layers were applied and finally a Linear layer to reduce the output_features to 10. The Layer 7 was modified to have output features as 1024 in order to reduce the number of features to have better predictions when finally getting 10 output features.

Theory Questions

1.

The cross-entropy loss for multiclass classification is given as : $-\Sigma_z P(z)log(Q(z))$ where P(z) is the original probability distribution (one-hot vector of the inputs) and Q(z) is the predicted probability distribution Now the KL-divergence is given as :

$$KL(P|Q) = \Sigma_z P(z)log\frac{P(z)}{Q(z)}$$

$$KL(P|Q) = \Sigma_z\{P(z)logP(z) - P(z)logQ(z)\}$$

$$KL(P|Q) = \Sigma_z\{P(z)logP(z)\} - \Sigma_z\{P(z)logQ(z)\}$$

Now we know that for the multi-class classification problem, P(z) has only one value as 1 and all the remaining are zero. Let the z=c has $P(z=c) = 1$ then,

$$KL(P|Q) = \{0 + 0 + ... + P(z=c)logP(z=c) + ... + 0 + 0\} - \Sigma_z\{P(z)logQ(z)\}$$

$$KL(P|Q) = \{0 + 0 + ... + 1log(1) + ... + 0 + 0\} - \Sigma_z\{P(z)logQ(z)\}$$

$$KL(P|Q) = \{1 * 0\} - \Sigma_z\{P(z)logQ(z)\}$$

$$KL(P|Q) = -\Sigma_z\{P(z)logQ(z)\}$$

$$KL(P|Q) = \text{Cross Entropy Loss}$$

The underlying assumption here is in the probability distribution of the true labels, for a particular sample only one class has posterior probability = 1 and rest all the classes have posterior probability = 0 or we can say P(z) is a one hot encoded vector of the true labels.

2.

a)

Activation function for neurons **N1 and N2** will be A2 and for **N3** will be A1. Explanation and derivation for the same is given in part b.

b)

Output for Neuron N1 is given as:

$$a_1 = c(w_1x_1 + w_3x_2)$$

$$a_1 = cw_1x_1 + cw_3x_2$$

Output for Neuron N2 is given as:

$$a_2 = c(w_2x_1 + w_4x_2)$$

$$a_2 = cw_2x_1 + cw_4x_2$$

Output for Neuron N3 is given as:

$$y = sign[\sigma(w_5a_1 + w_6a_2) - 0.5]$$

Now, $w_5a_1 + w_6a_2 = w_5(cw_1x_1 + cw_3x_2) + w_6(cw_2x_1 + cw_4x_2)$

$$w_5a_1 + w_6a_2 = cw_1w_5x_1 + cw_3w_5x_2 + cw_2w_6x_1 + cw_4w_6x_2$$

$$w_5a_1 + w_6a_2 = (cw_1w_5 + cw_2w_6)x_1 + (cw_3w_5 + cw_4w_6)x_2$$

Let $b_1 = (cw_1w_5 + cw_2w_6)$ **and** $b_2 = (cw_3w_5 + cw_4w_6)$

$$w_5a_1 + w_6a_2 = b_1x_1 + b_2x_2$$

So,

$$y = sign[\sigma(b_1x_1 + b_2x_2) - 0.5]$$

$$y = sign[\frac{1}{1 + e^{-(b_1x_1+b_2x_2)}} - 0.5]$$

$$y = sign[\frac{e^{b_1x_1+b_2x_2}}{e^{b_1x_1+b_2x_2} + 1} - 0.5]$$

Now, y will have value 1 when ,

$$\frac{e^{b_1x_1+b_2x_2}}{1 + e^{b_1x_1+b_2x_2}} > 0.5 \tag{1}$$

From the question, y will have value 1 when

$$P(Y = 1|X) > P(Y = -1|X)$$

$$P(Y = 1|X) > 1 - P(Y = 1|X)$$

$$2 * P(Y = 1|X) > 1$$

$$P(Y = 1|X) > \frac{1}{2}$$

$$P(Y = 1|X) > 0.5$$

Taking P(Y=1—X) value from the question

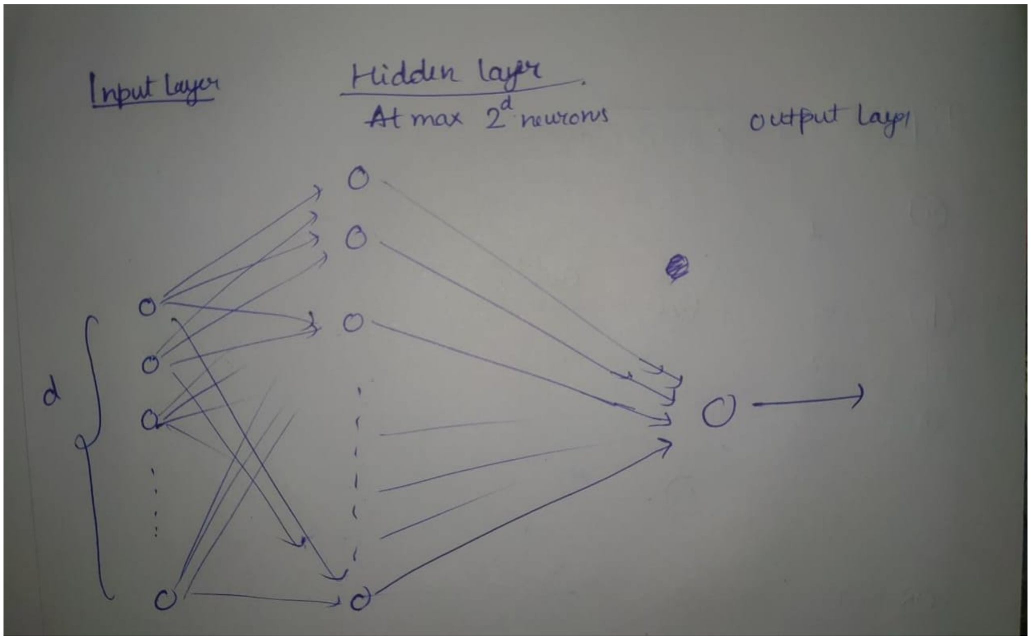
$$\frac{e^{\beta_1x_1+\beta_2x_2}}{1 + e^{\beta_1x_1+\beta_2x_2}} > 0.5 \tag{2}$$

On comparing 1 and 2 we see that we have essentially the same equations, thus $\beta_1 = b_1$ and $\beta_2 = b_2$ Hence , $\beta_1 = cw_1w_5 + cw_2w_6$ and $\beta_2 = cw_3w_5 + cw_4w_6$

3.

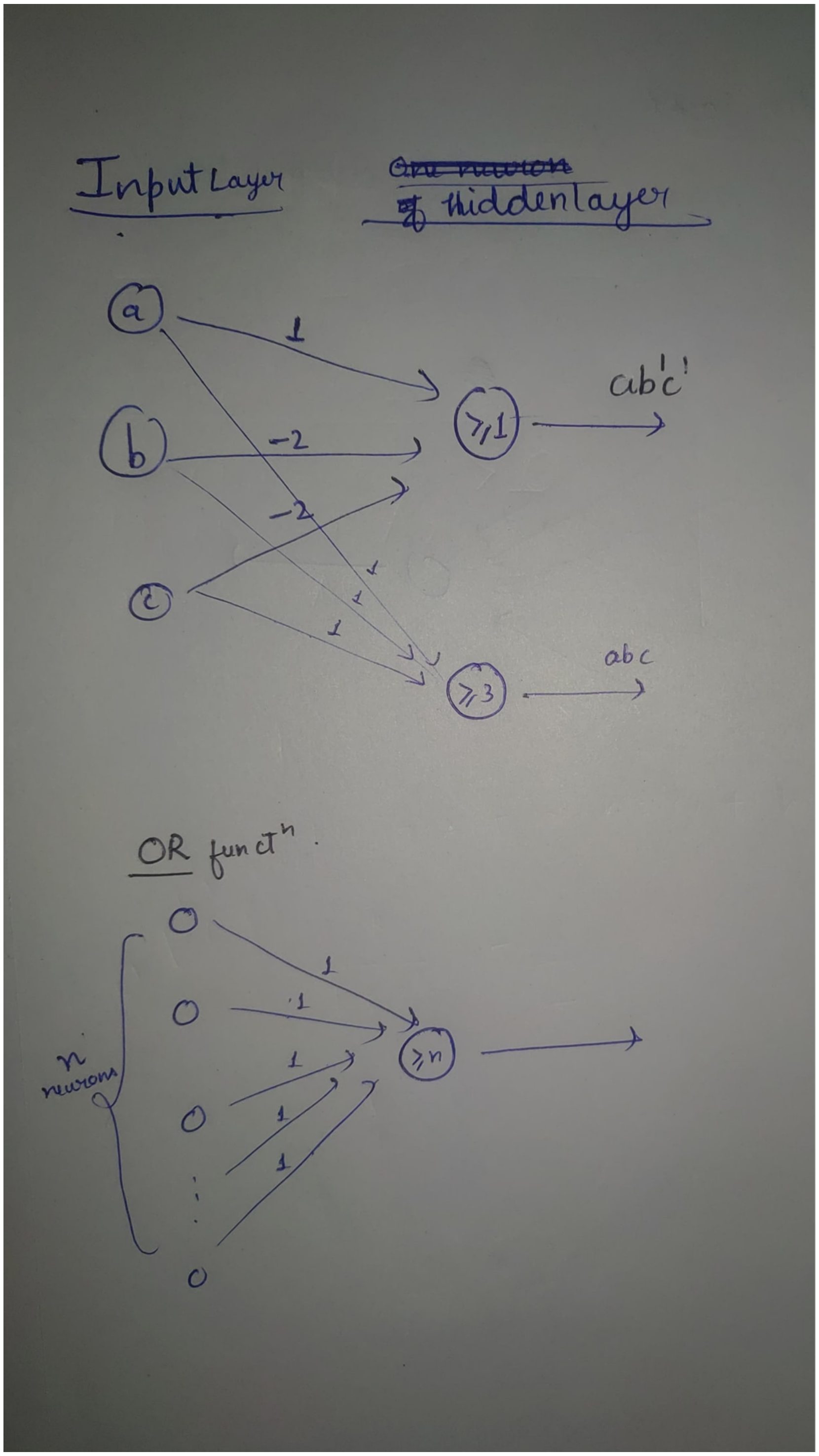
Any binary function with any number of Boolean variables giving a Boolean output can be written either in the form of **Sum of Products (SOP)** form or **Product of Sums (POS)** form from their respective truth table or by simply reducing them into one.

Let us consider SOP form . In the first layer (input layer) we will have d Boolean variables , in the second layer (hidden layer) we will have all the product terms of the SOP terms and in the third layer (output layer) we will only one neuron. All the input variables from the first layer will be mapped to a particular neuron in the second layer. The weights of these input neurons and the threshold of the output neurons can be decided manually by doing some calculation. These product terms can be at max 2^d. From the hidden layer to output layer we can use a simple OR function with each weights as 1 for each neuron of hidden layer going to output layer neuron.The threshold for output layer neuron will be the number of neurons in the hidden layer. The representation of the same is given below.



The weights from hidden layer to output layer are define and shown in the figure below for a simple OR function with n input neurons . For going from input layer to the hidden layer lets consider an example with three Boolean variables a,b,c w.l.o.g and consider two product terms $ab'c'$ and abc of a general function.

We can see in the diagram below that we can easily compute these two using only one map operation (going from input layer to hidden layer) for each of the terms and we already know that the sum of these product terms (neurons of hidden layer) can be easily computed. Thus this idea can be generalised to a model with d input boolean variables and only one hidden layer is required to compute such function.


Input Layer
~~One neuron~~
~~of~~ Hidden layer
a
b
c
1
-2
-2
1
1
1
≥1
ab'c'
≥3
abc
OR functⁿ
1
1
1
1
1
n neurons
≥n