Deep Learning: Assignment 1

by

Palaskar Adarsh Mahesh

B20EE087

January 29, 2023

1 Question 1

In this question we have implemented an Artificial Neural Network(ANN) for the CIFAR10 dataset. This dataset has 50000 training samples, which we have divided into 45000 for training set and 5000 for validation set. It has 10000 samples in the testing split.

1.1 Performance Comparison of Sigmoid, Tanh, and ReLU activation functions:

Various activation functions are used for different applications and each one of them has its own strengths and weaknesses.

1.1.1 Sigmoid:

The sigmoid activation function maps its inputs to values between 0 and 1, making it useful for binary classification problems where the output must be in the range of 0 and 1. However, sigmoid has a slow convergence and can also result in vanishing gradients, which can slow down the training process or may even ultimately stop it.

The following results were obtained after training the ANN for 40 epochs using sigmoid:

Parameter	**Value**
Train Loss	1.172
Train Accuracy	57.769
Validation Loss	1.487
Validation Accuracy	48.720
Test Loss	1.462
Test Accuracy	49.430

1.1.2 Tanh:

The Tanh activation function maps its inputs to values between -1 and 1. It has a larger range compared to sigmoid and this makes it useful for capturing non-linear relationships, but it also has a slow convergence and can also result in vanishing gradients.

The following results were obtained after training the ANN for 40 epochs using tanh:

Parameter	Value
Train Loss	1.351
Train Accuracy	51.540
Validation Loss	1.535
Validation Accuracy	46.300
Test Loss	1.502
Test Accuracy	46.210

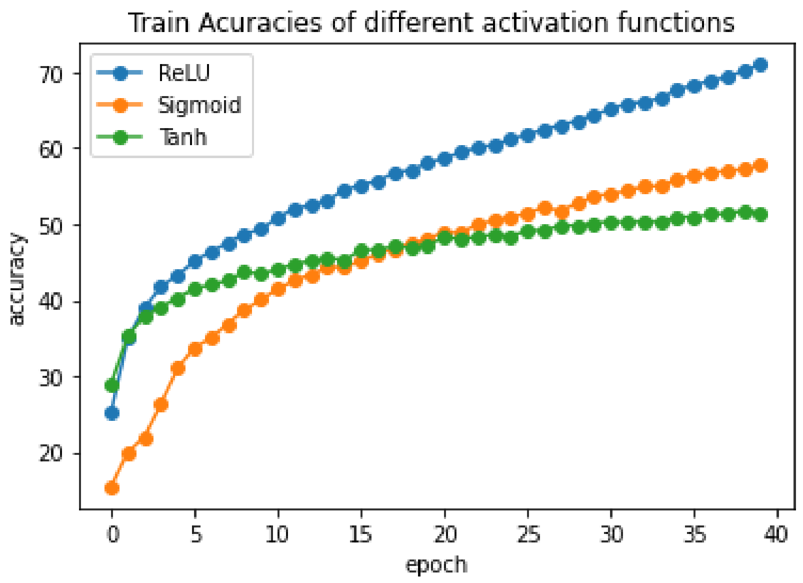
1.1.3 ReLU:

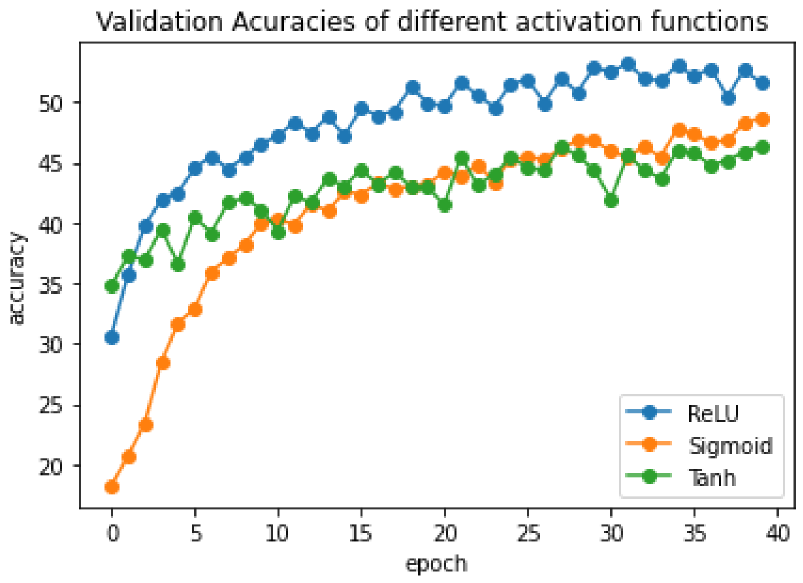
The ReLU activation function maps all negative values to zero and leaves all positive values unchanged. This makes it computationally efficient and less prone to overfitting than the sigmoid and tanh functions. However, ReLU has the disadvantage of the "dying ReLU" problem, where a neuron may stop producing any output during training since it gives a zero gradient value for inputs less than zero, effectively reducing the network's ability to learn.

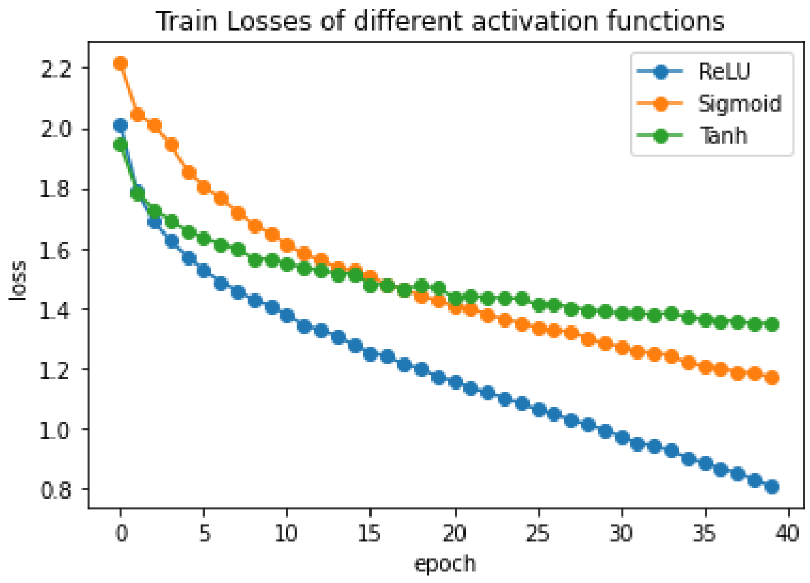
The following results were obtained after training the ANN for 40 epochs using ReLU:

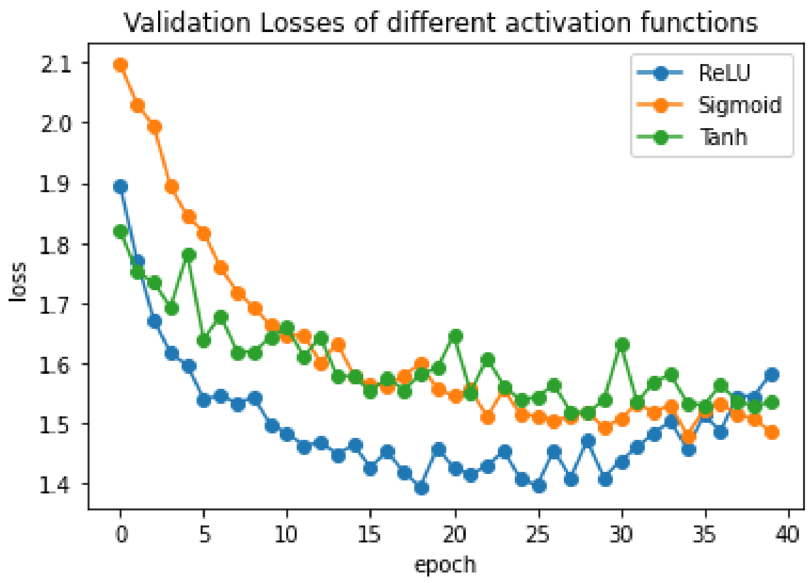
Parameter	Value
Train Loss	0.809
Train Accuracy	71.049
Validation Loss	1.582
Validation Accuracy	51.620
Test Loss	1.511
Test Accuracy	51.920

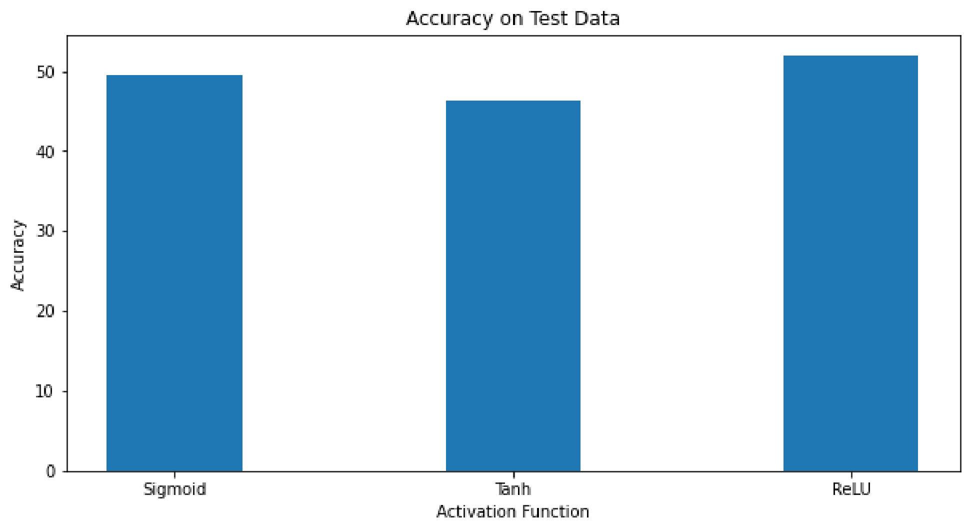
1.1.4 Comparison of Results:






Train Losses of different activation functions
ReLU
Sigmoid
Tanh
loss
epoch
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0
5
10
15
20
25
30
35
40


Validation Losses of different activation functions
ReLU
Sigmoid
Tanh
loss
epoch
2.1
2.0
1.9
1.8
1.7
1.6
1.5
1.4
0
5
10
15
20
25
30
35
40



It is observed that ReLU gives the highest accuracy on the dataset for this ANN architecture, but the observed difference in accuracies is not a lot. From the above graphs, it is observed that Training accuracy for ReLU is very high compared to validation and testing accuracies, implying that the model is over-fitting. This difference is less in case of sigmoid and ReLU and therefore these models are less prone to over-fitting.

1.2 Detecting Vanishing Gradient

The vanishing gradient problem occurs in deep neural networks when the gradients become very small during the backpropagation step of training, making it difficult for the model to learn. This can occur when using activation functions such as sigmoid or tanh, which have a saturation region where the derivative is close to zero, resulting in a small gradient. The small gradient means that the model parameters do not change much during training, leading to slow convergence or even failure to converge. Vanishing Gradient is detected using the following methods:

1.2.1 Identifying stagnant loss during training:

When there is no change in the loss during training after each epoch, we can infer that out neural network is no longer learning and a reason of this failure could be the vanishing gradient problem.

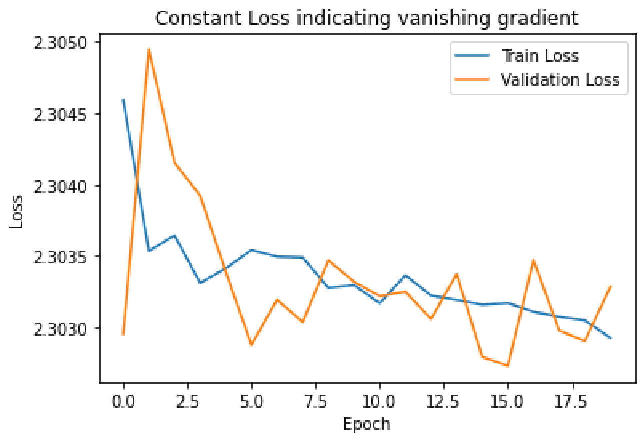


Figure 1: Loss comparison after each epoch

1.2.2 Comparing mean and standard deviation of weights at every epoch at every layer:

After comparing mean and standard deviation of weights at every epoch at every layer it is observed that there is little to no variation(less than 0.003) in the mean and standard deviation of the weights. This behaviour implies that weights do not change much during training of the ANN.

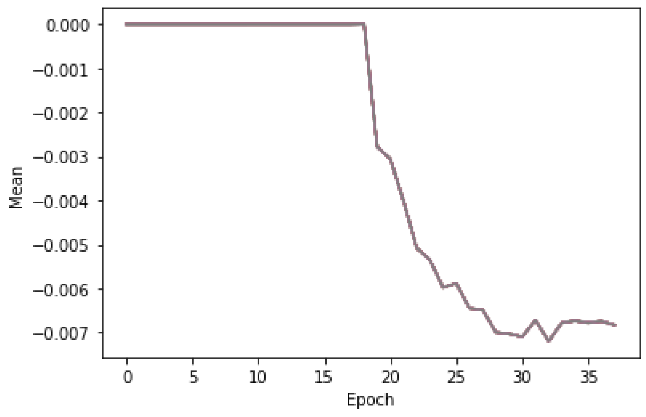


Figure 2: Comparing mean at every epoch

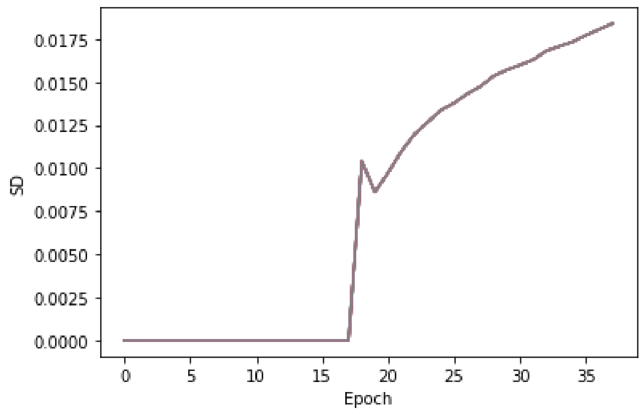


Figure 3: Comparing standard deviation at every epoch

1.2.3 Comparing Weights of a particular layer at the first and last epochs:

In the deep ANN architecture, we compared the weights corresponding to the first hidden layer on Epoch 1 and Epoch 20. Very little to no difference in values was observed, indicating that the weight values did not change during the training process and the gradient values were reduced to zero.

1.3 Methods to overcome the Vanishing Gradient Problem:

1.3.1 Using variants of ReLU(LeakyReLU, ReLU, etc.)

Activation functions such as ReLU and its variants (Leaky ReLU, etc.) do not have a saturation region and result in non-zero gradients, making them less prone to the vanishing gradient problem.

Using ReLU on the deep NN, the vanishing gradient problem was resolved and the following results were obtained:

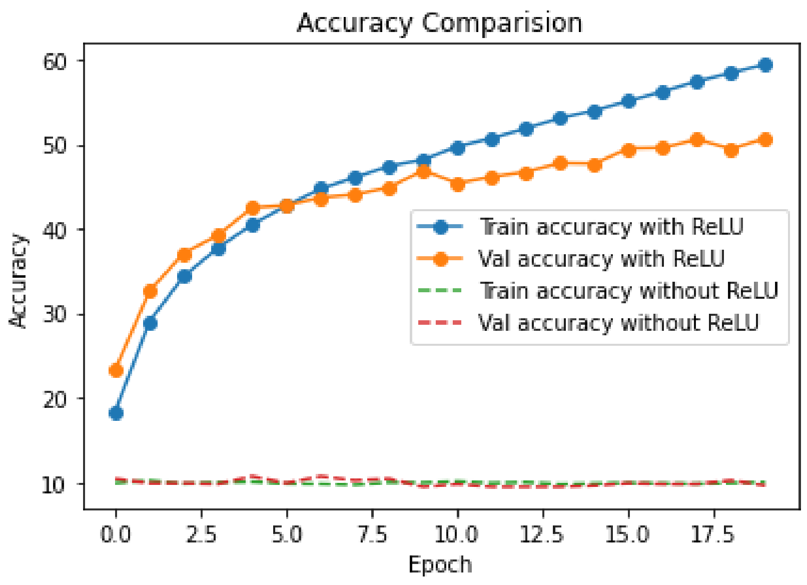


Figure 4: Comparing standard deviation at every epoch

1.3.2 Using Batch Normalization:

Normalization techniques such as batch normalization can help stabilize the distribution of activations and gradients, reducing the likelihood of the vanishing gradient problem.

1.3.3 Using Weight Initialization Techniques:

Proper weight initialization can help the model converge faster and more reliably to a good solution, can reduce the likelihood of overfitting, where the model performs well on the training data but poorly on new, unseen data and is powerful than a random initialization, since random initialization can also initialize zero values which can affect the overall network.

After applying the above techniques of Weight initialization and Batch Normalization, significant increase in accuracy of test data as well as less overfitting was observed.

2 Question 2

In this question we have implemented an Artificial Neural Network(ANN) for the Gurmukhi dataset. This dataset contains 10 classes for corresponding to 10 numbers, which our network is supposed to identify.

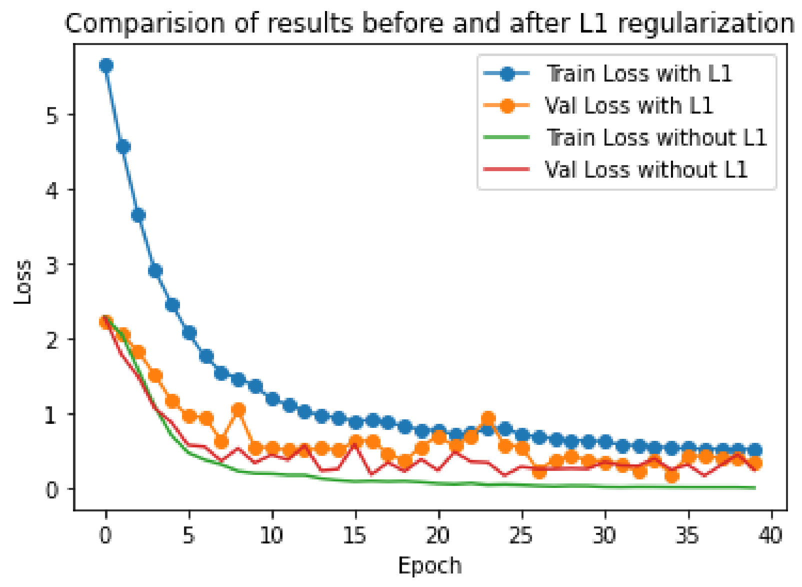
2.1 Implementing various Regularization techniques and comparing them:

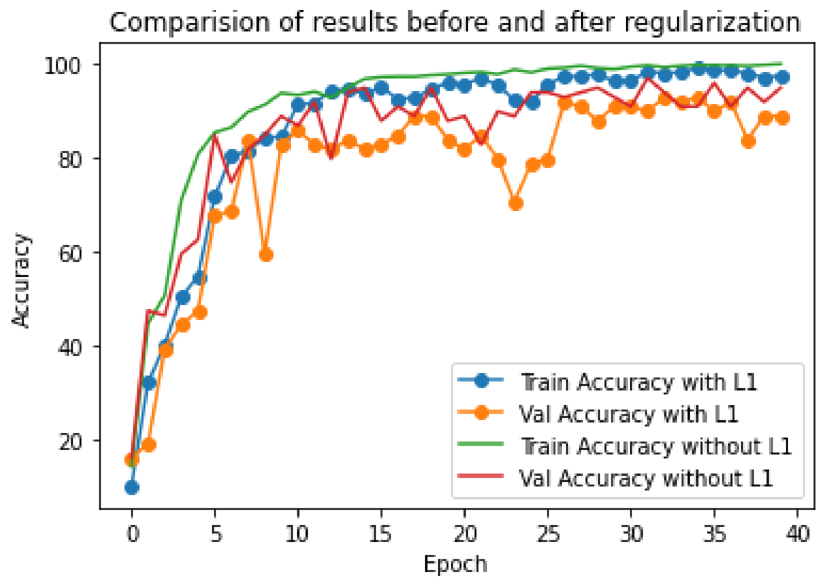
Regularization is to prevent overfitting and improve the generalization ability of a model. Overfitting occurs when a model performs well on the training data but poorly on new, unseen data. This is often due to the model having learned too much detail from the training data and being unable to generalize to new examples.

Regularization adds a penalty term to the loss function during training, which discourages the model from having large weights. This can reduce the complexity of the model and prevent overfitting. The most commonly used regularization techniques in deep learning are L1 regularization, L2 regularization, and dropout.

2.1.1 L1 regularization:

In L1 regularization, a penalty term proportional to the absolute value of the weights is added to the loss function during training. This has the effect of shrinking the magnitude of the weights towards zero, encouraging sparse solutions where some weights are exactly zero. As a result, L1 regularization can be used for feature selection, as it can lead to sparse models where only a subset of the features are used. The L1 regularization term is typically controlled by a hyperparameter, which determines the strength of the regularization. If the hyperparameter is set to a high value, the regularization term will be strong and many weights will be shrunk towards zero. If the hyperparameter is set to a low value, the regularization term will be weak and the model will be less likely to be regularized. The following results are obtained after applying L1 regularization:

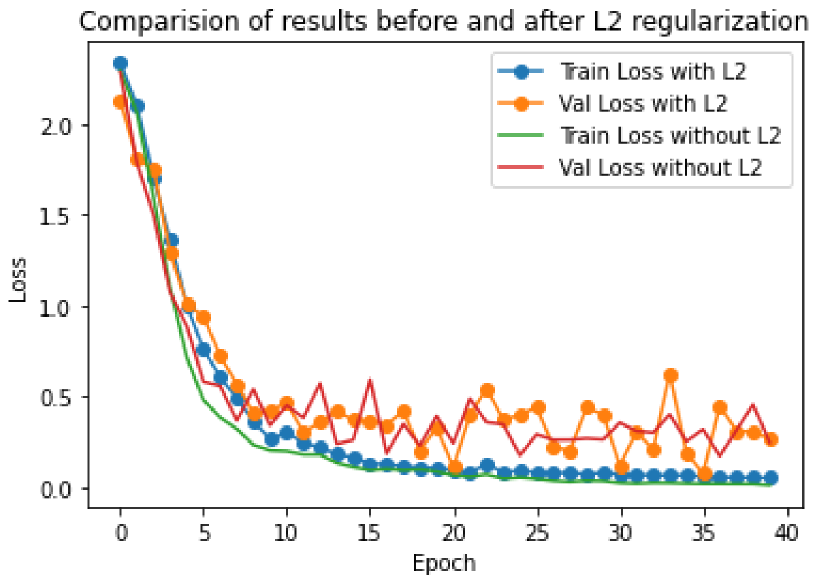


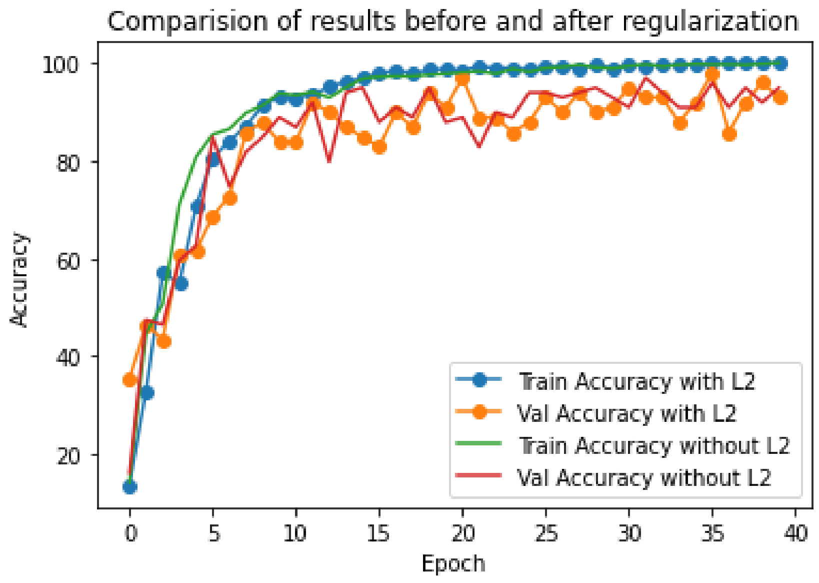


2.1.2 L2 Regularization:

In L2 regularization, a penalty term proportional to the square of the weights is added to the loss function during training. This has the effect of shrinking the magnitude of the weights towards zero, encouraging solutions with small weights. Unlike L1 regularization, L2 regularization does not encourage sparse solutions where some weights are exactly zero.

The L2 regularization term is typically controlled by a hyperparameter, which determines the strength of the regularization. If the hyperparameter is set to a high value, the regularization term will be strong and the weights will be shrunk towards zero. If the hyperparameter is set to a low value, the regularization term will be weak and the model will be less likely to be regularized.

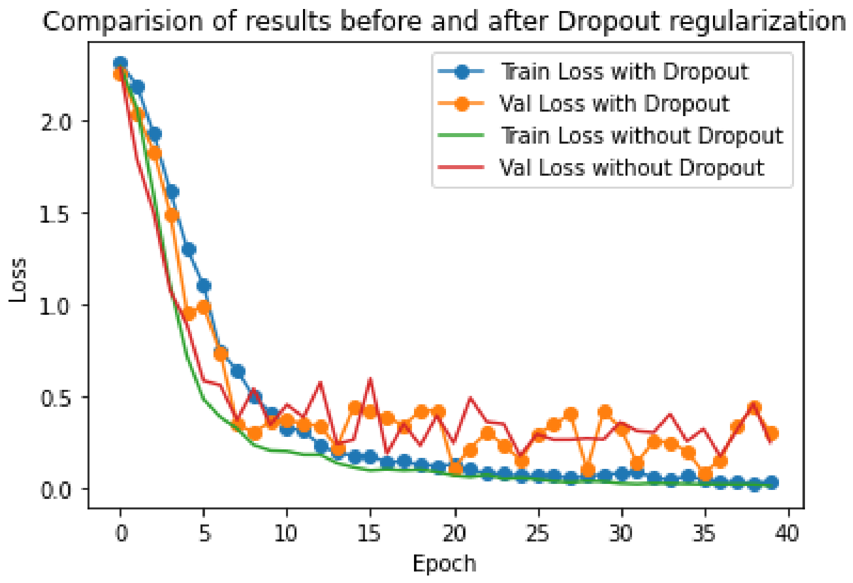


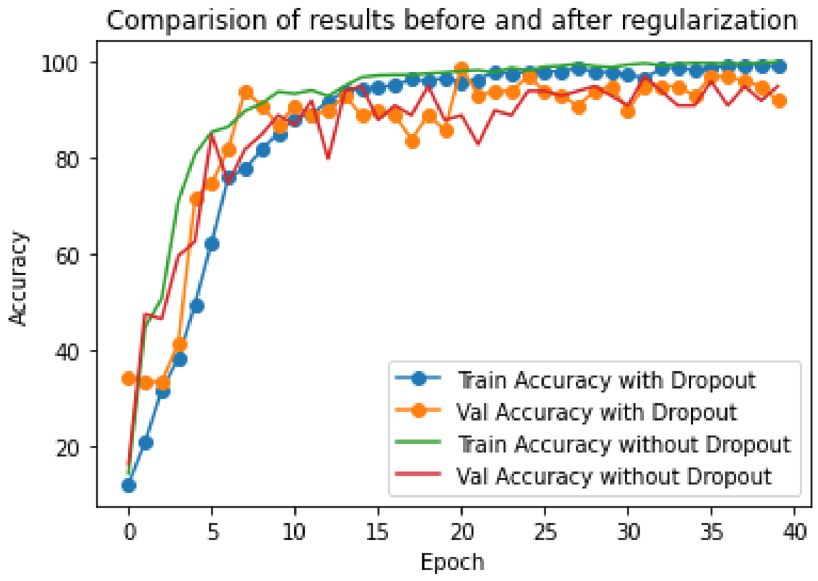


2.1.3 Dropout Regularization:

In dropout, during training, some neurons in the network are randomly "dropped out" or disabled, meaning their activations are set to zero. The probability of dropping out a neuron is controlled by a hyperparameter, which determines the rate at which neurons are dropped out. The dropped-out neurons are not used in the forward pass of the model, reducing the complexity of the model and adding some degree of randomness to the training process.

Dropout is usually applied to fully connected layers, where each neuron in the layer is connected to every neuron in the previous layer. Dropout is applied at each training iteration, and at test time, all neurons are used, so the model effectively has an exponential number of possible configurations. This has the effect of averaging over many different models, reducing the risk of overfitting.





2.1.4 Observed Results and comparison of various regularization techniques:

It observed that in case of all the regularization techniques, better test accuracies are obtained. L2 regularization gives the best accuracy, followed by dropout and L1 regularization. This can be explained by the property of L1 regularization to reduce weights towards zero, which can also make them equal to zero thus eliminating its significance. This is not observed in L2 regularization since it always maintains a value that is not equal to zero. Also during testing since dropout uses all the neurons, it essentially uses various different models thus leading to less complexity and reducing the risk of underfitting.

2.2 Implementing Gradient Checking:

Gradient checking is a debugging technique used to check the correctness of the gradient computation in deep learning models. It involves numerically estimating the gradient of the loss function and comparing it to the gradient computed by backpropagation. If the two gradients are significantly different, it indicates a bug in the implementation or the gradient computation, and the model may not be learning correctly. Gradient checking is typically used early in the development process to verify the correctness of the model and its gradient computation.

Using the formula for approximating gradients, we have assigned epsilon to a very small value greater than zero and then calculated the gradients using the first rule of differentiation. This calculation has been performed for the weights of layer linear5 in the deep neural network. These gradients have then been compared with the actual gradients calculated by the model and if the difference is found to be very small, we can infer that the gradients calcualted by the model are correct.