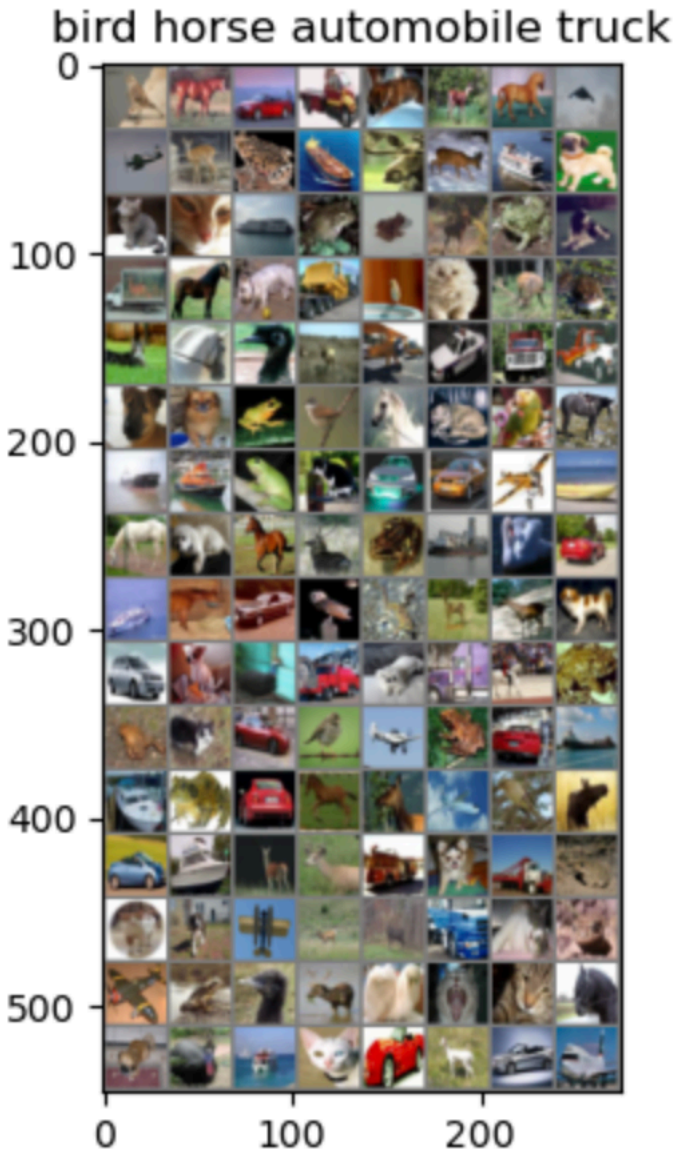
HW3 Q1: Producing Vanishing gradients and resolving them on CIFAR-10 Dataset

By Sundar Swaminathan

Dataset

The data set that will be used for this assignment is the CIFAR dataset. We will be using the CIFAR-10 dataset provided by the PyTorch library(https://pytorch.org/vision/main/generated/torchvision.datasets.CIFAR10.html). This is a dataset that contains 60,000 images of size 32*32 images of everyday objects, categorised in 10 categories of everyday objects for 6,000 images in each category.



Sample Images from the dataset

Preprocessing

I have preprocessed the data in different steps as follows:

- **Conversion:** The data is converted from PIL format(0-255) into tensor format(0-1) by dividing pixel values by 255. The shape changes from HWC(Height, Width, Channels) into CHW.
- **Normalise:** Normalises each pixel to fir (0.5, 0.5, 0.5) format by independently applying it to R, G, B colours and the pixel values converted earlier(0-1) are scaled to (-1, 1).
- **Precision:** The data is initially in an int based structured array, this is converted to float 32 format to improve precision and avoid overflow issues
- **Dataloader:** Wraps the dataset into a data loader for efficient batch processing, I have used a batch size of 128 and set shuffle to true
- **Train-Test Split:** I have used a training and split ratio of 90%-10% which is the default used in CIFAR datasets.

Architecture

Input

Setup: Fully connected neural network
Layer 1: Receives images with 32 * 32 * 3 = 3,072 features
Layer 2: Hidden Layers with varying sizes for different purposes with a sequence of 512, 256, 128, 64, 32, 16 neurons with activation function sigmoid, tanh and relu
Layer 3: Final fully connected layer with 10 neurons to Map to the 10 output classes

---

Output

10 neuron output layer with no explicit activation function, instead CrossEntropyLoss, which uses softmax internally is used.

Optimizers

Training 1: Adam with Default learning rate(0.001) and validation split of 0.2

Epochs

The model is trained for 10 epochs

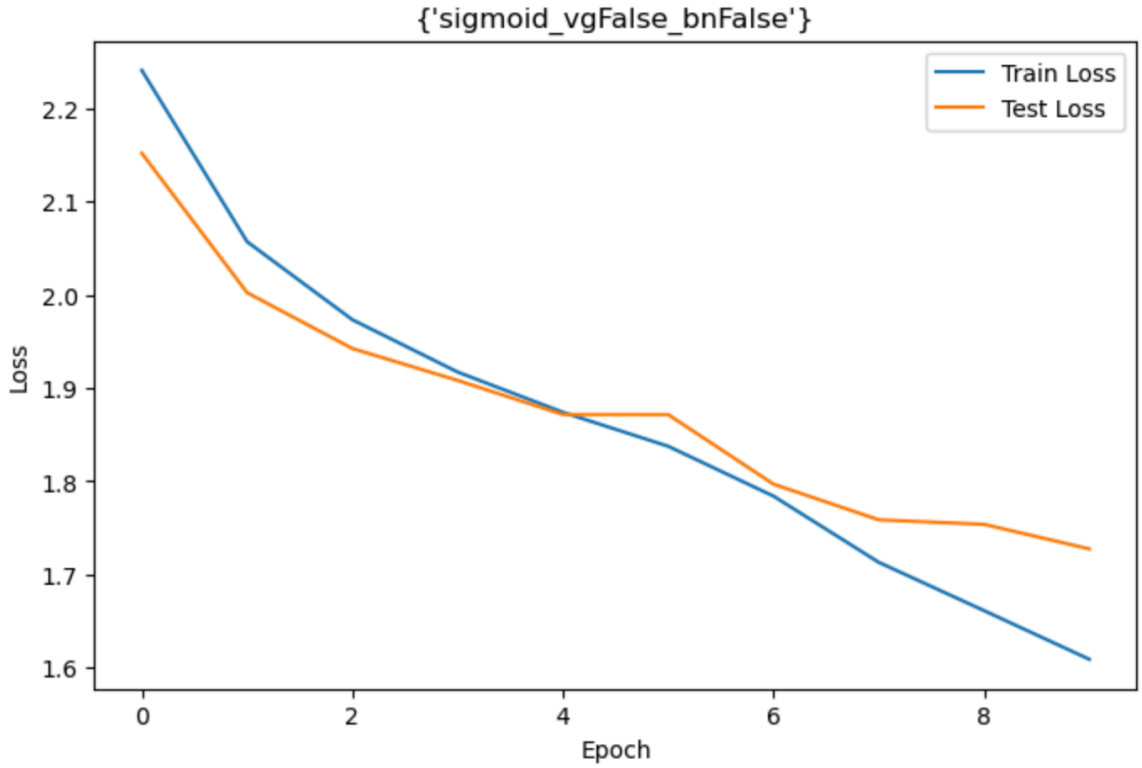
Training

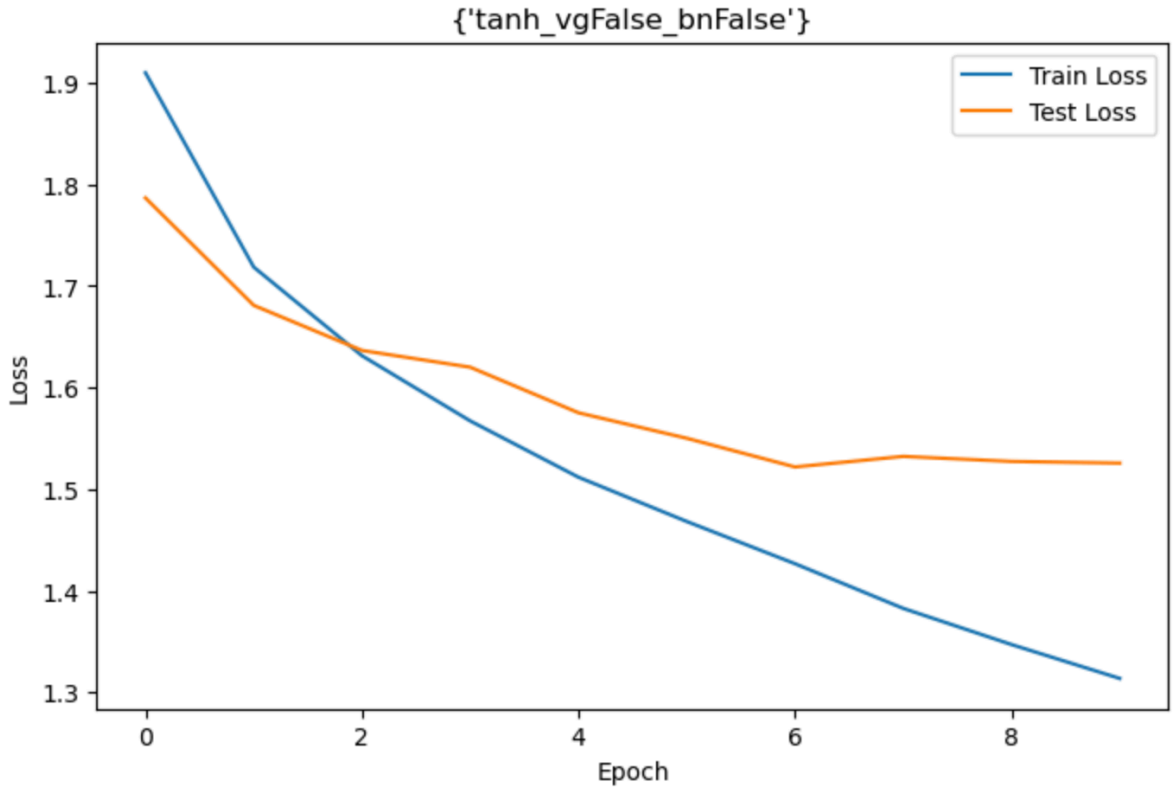
For training the model, I used 3 different experiments for each activation function, plain, with vanishing gradient and with fixes for the vanishing gradient for a total of 9 analyses. As expected ReLU was best suited for this use case. With regular training, the model produced a steadily declining gradient but when more and more layers were added the gradients became less pronounced and almost reached 0 effective change. In order to combat this, I added He initialisation for ReLU and Xavier initialisation for sigmoid and tanh activation functions. I plotted gradient graphs for all these changes to effectively log and understand these hyper parameter tuning effects.

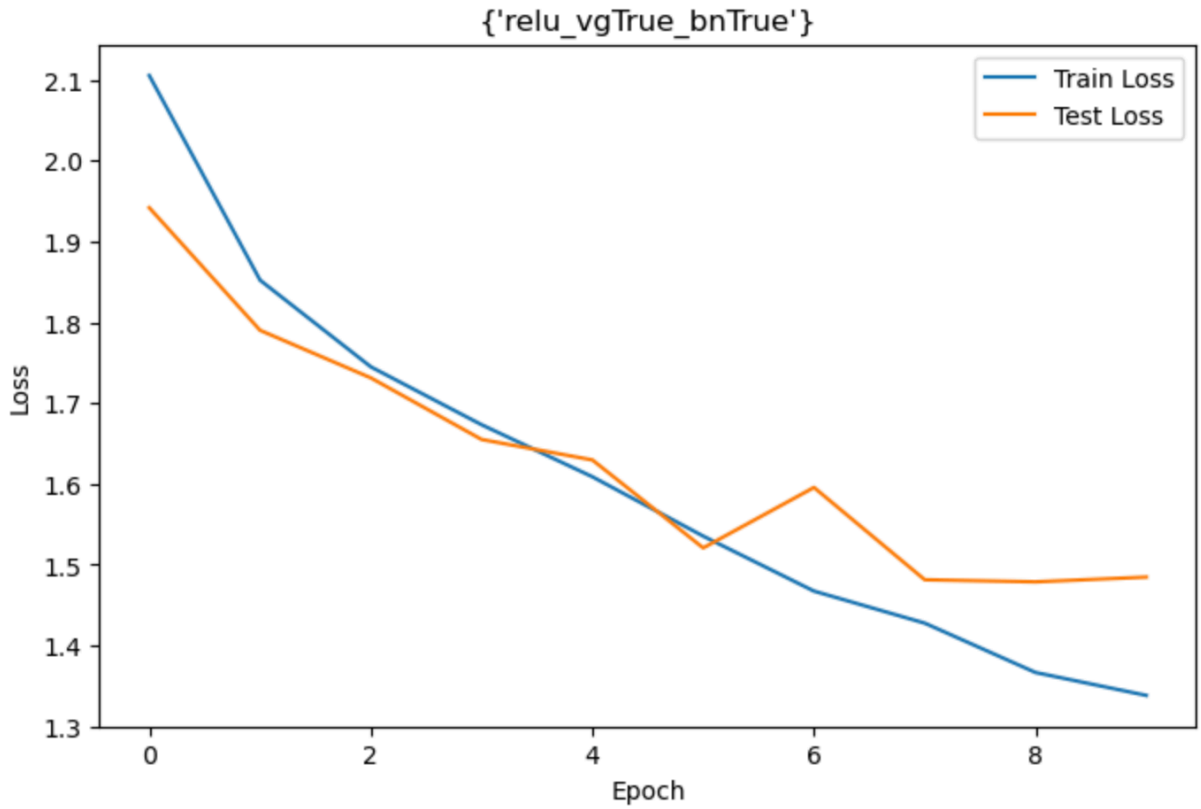
Impact of Activation Functions

I have used Adam optimiser to minimise the effects of optimisers and focus on the impacts of activation functions and adding and reducing layers. From the observations, I came to the conclusion that ReLU performs the best for image classification tasks with significantly low loss and high accuracy compared to sigmoid and tanh.

Here's a comparison of the different train and test loss for the 3 activation functions:



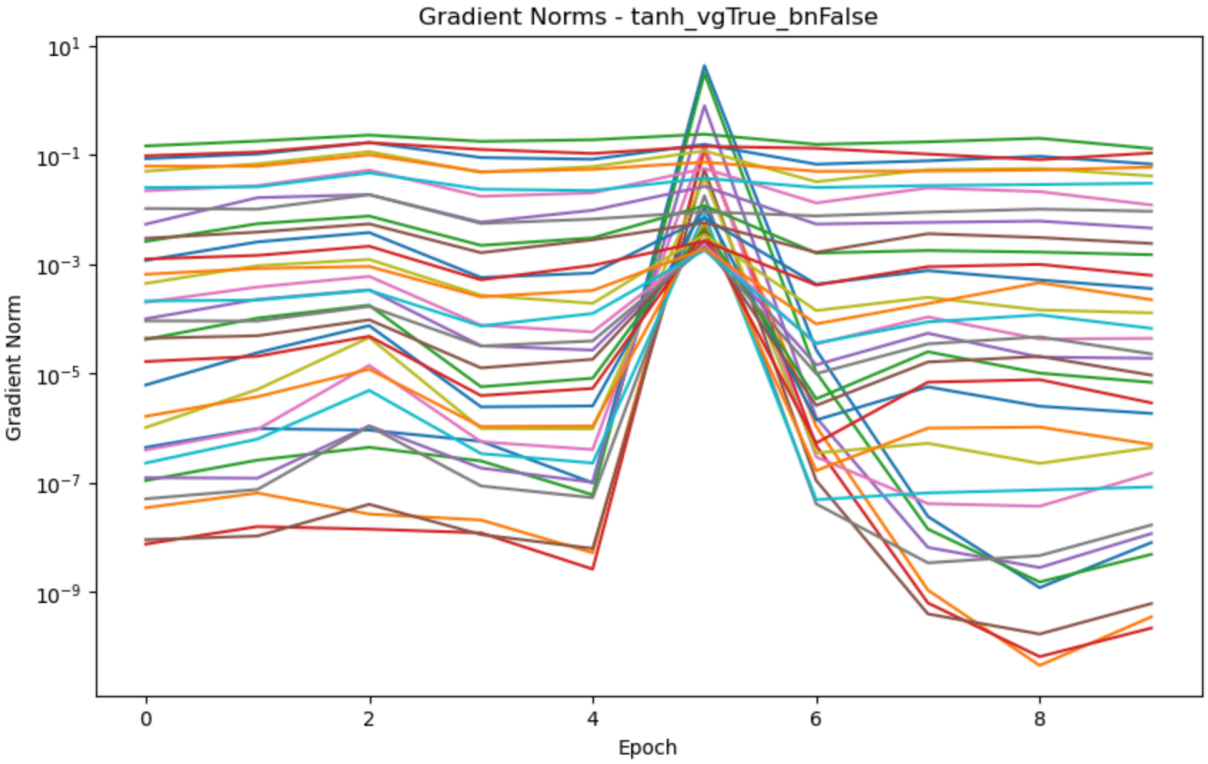


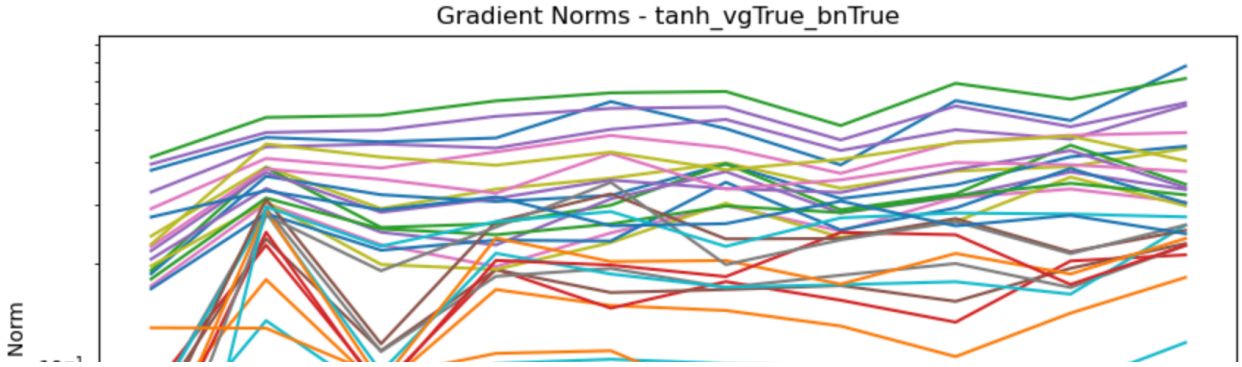


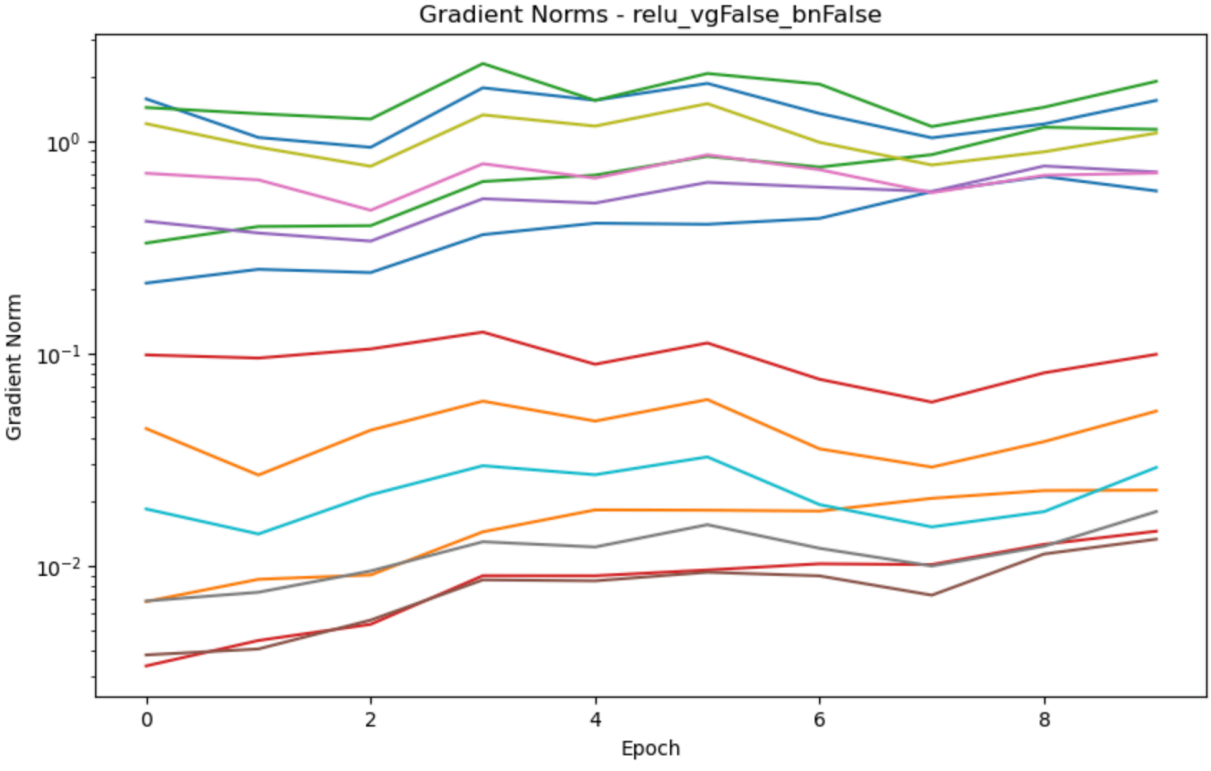
Vanishing Gradient Cause and Fix

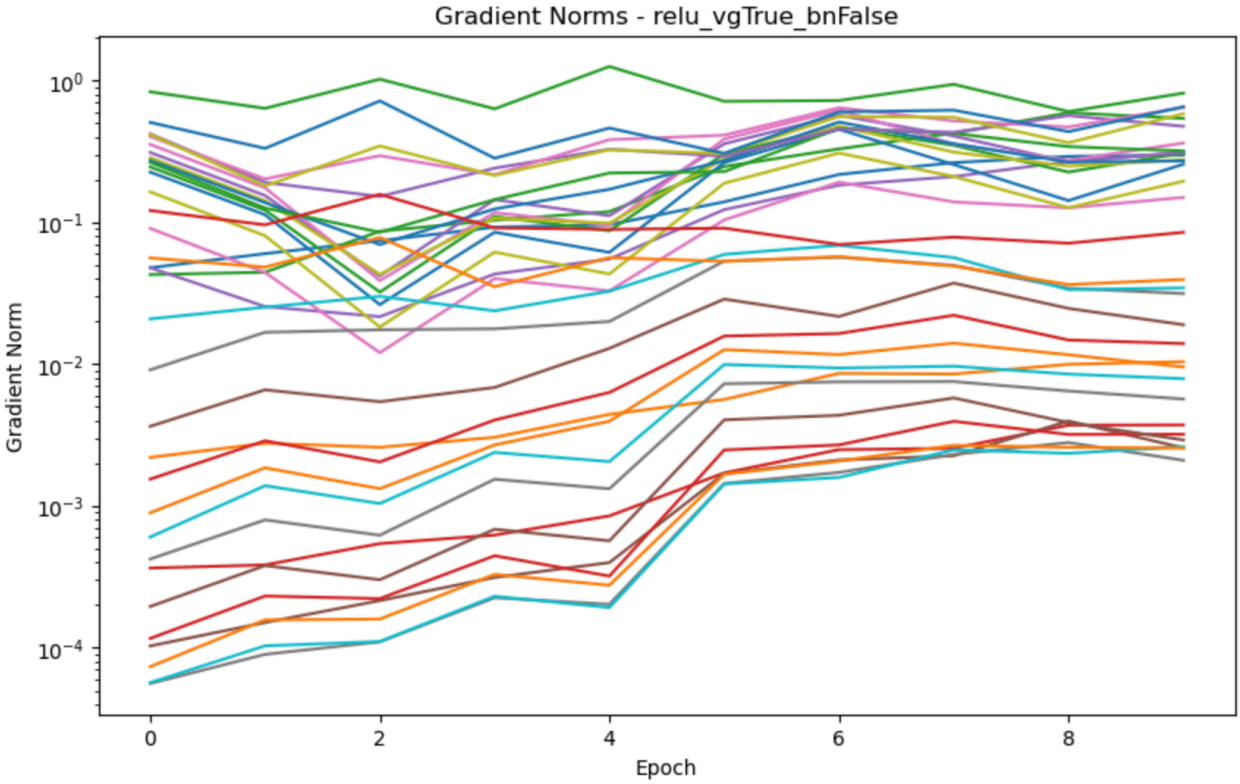
As the next step of the experiment, I added 10 layers with fixed size(16, 16) to artificially induce a vanishing gradient. As expected, the more layers I added the less and less the gradient became, indicating a severe vanishing gradient problem. Then as the logical next step, I added weight initialization and batch normalization to fix the vanishing gradient problem which pulled the gradients back into an acceptable range. The presence of vanishing gradients can be identified from the continuous reduction of gradients to the point it is insignificant. In this case, it reached 10^-14. All the graphs can be found below:

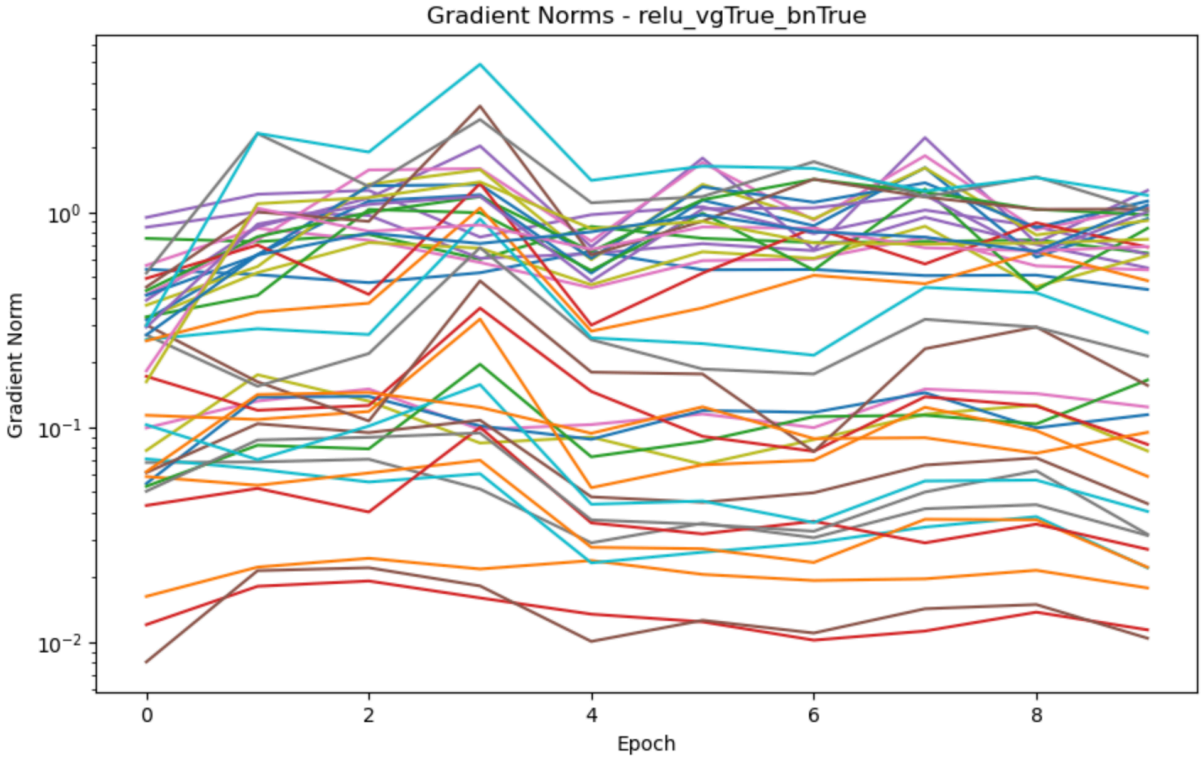
Sigmoid/Tanh/Relu = activation function
Vg = Vanishing Gradient introduced
Bn = Batch normalization / weight initialization applied


Gradient Norms - tanh_vgTrue_bnFalse
Gradient Norm
Epoch


Gradient Norms - tanh_vgTrue_bnTrue
Norm


Gradient Norms - relu_vgFalse_bnFalse
Gradient Norm
Epoch


Gradient Norms - relu_vgTrue_bnFalse
Gradient Norm
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
2
4
6
8
Epoch


Gradient Norms - relu_vgTrue_bnTrue
Gradient Norm
10^{0}
10^{-1}
10^{-2}
0
2
4
6
8
Epoch

Conclusion

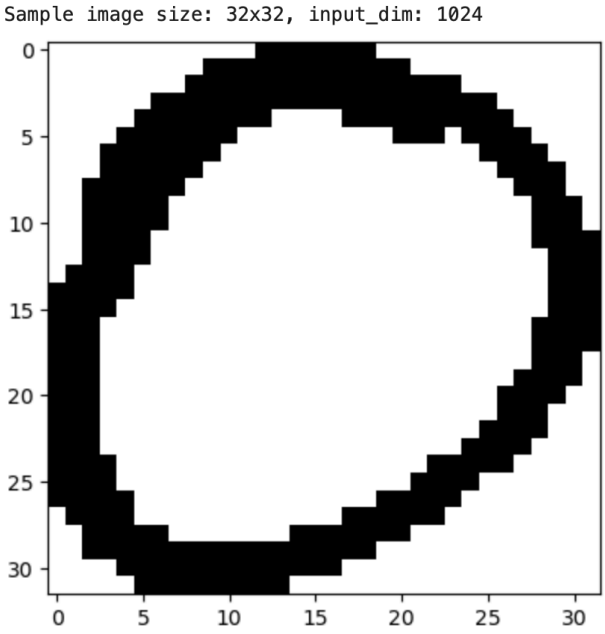
After the experiment, I feel that for image classification tasks ReLU performs best and produced the most accuracy with least epochs. Tanh performs slightly better than sigmoid but both of them are much worse than ReLU. The starting accuracy for sigmoid and tanh was around 80% while ReLU started with around 88%. Coming to the vanishing gradient, I was able to successfully produce a vanishing gradient by adding 10 layers with fixed size(16, 16). Then, I added weight initialisation and batch normalization to fix the vanishing gradient introduced and plotted the difference in the gradients in a graph for analysis.

HW3 Q2: Implementing Neural Network for Gurmukhi Dataset and implementing various regularisation techniques from scratch

By Sundar Swaminathan

Dataset

The data set that will be used for this question is the Gurmukhi numerical dataset. We will be implementing a neural network to train on this dataset and then using a number of tests to check the accuracy and plot the gradients. We will also be tuning the hyper parameters to study the imapact. We will be implementing L1, L2 and Dropout regularisations from check and also implement an in-house gradient checking to plot the changes.



Preprocessing

I have preprocessed the data in different steps as follows:

- **Conversion:** The data is converted from PIL format with pixel values in the range(0-255) into tensor format(0-1) using .ToTensor() method. The shape changes from HWC(Height, Width, Channels) into CHW.
- **Normalise:** Normalises each pixel to fir (0.5, 0.5, 0.5) format by independently applying it to R, G, B colours and the pixel values converted earlier(0-1) are scaled to (-1, 1). This helps with stability and faster convergence.
- **Precision:** The data is initially in an int based structured array, this is converted to float 32 format implicitly
- **Dataloader:** Wraps the dataset into a data loader for efficient batch processing, I have used a batch size of 64 since the dataset size here is smaller. Shuffling is enabled for training to enable randomisation while it is disabled in testing to maintain order during evaluation.
- **Train-Test Split:** Separate folders are used for training and testing(train/, val/) thereby not needing to split the data in the code.

Architecture

FULLY CONNECTED NEURAL NETWORK FOR DIGIT CLASSIFICATION

Input

Layer 1: Receives images with 1 * 32 * 32 = 1,024 features when flattened
Layer 2: Hidden Layers with progressively smaller sizeswith a sequence of 512, 256, 128, 64, 16 neurons with activation function relu
Layer 3: Final fully connected layer with 10 neurons to Map to the 10 output classes(digits 0-9)

Output

10 neuron output layer with no explicit activation function, instead CrossEntropyLoss, which uses softmax internally is used.

Optimizers

Training 1: Adam with Default learning rate(0.001)

Epochs

The model is trained for 10 epochs

Training

For training the model, I used Relu, because it is best for image classification tasks. I added Dropout during the training phase for all experiments. Along with that, I did 3 different experiments and mapped them in plots to understand the impact of each and understand the gradient descent: L1 regularization, L2 regularization and L1 and L2 regularization. Each of these produced different impacts and I studied the 3 plots to come up with the conclusions.

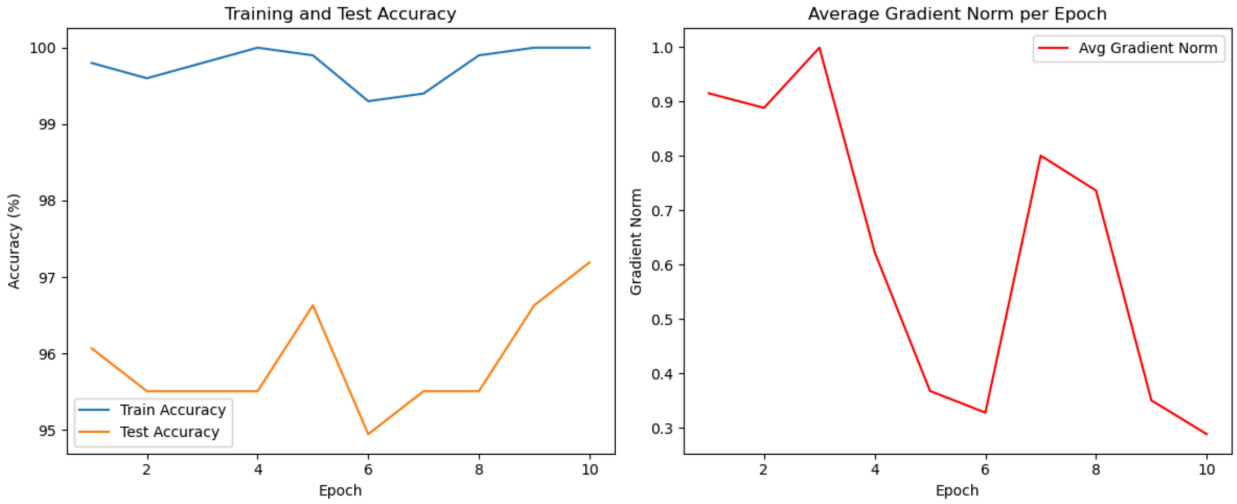
Impact of Regularization Functions

I found that the L2 regularization combined with Dropout gave the best results, this is consistent with expectations and L1 regularization combined with Dropout gave the result. The gains provided by L2 are offset by the loss produced by L1 in the combination of L1, L2 and Dropout regularization functions.

Here's a comparison of the different accuracy and gradient norm changes:

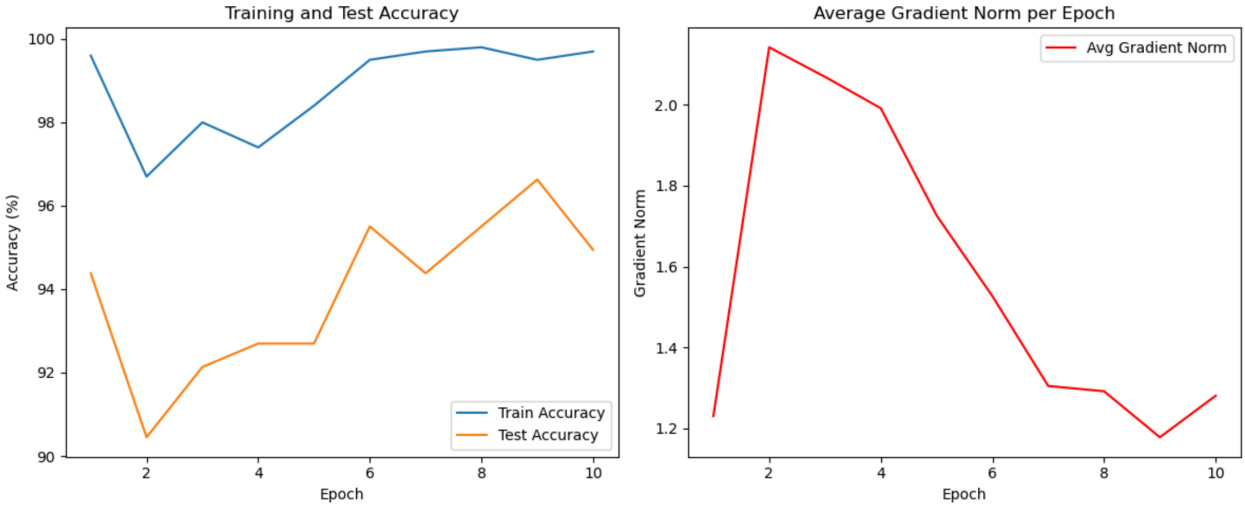
Only Dropout Regularization:

```
Training with Dropout regularization
Epoch [1/10] - Loss: 0.0856, Train Acc: 99.80%, Test Acc: 96.07%, Grad Norm: 0.9148
Epoch [2/10] - Loss: 0.0559, Train Acc: 99.60%, Test Acc: 95.51%, Grad Norm: 0.8881
Epoch [3/10] - Loss: 0.0478, Train Acc: 99.80%, Test Acc: 95.51%, Grad Norm: 0.9987
Epoch [4/10] - Loss: 0.0334, Train Acc: 100.00%, Test Acc: 95.51%, Grad Norm: 0.6225
Epoch [5/10] - Loss: 0.0130, Train Acc: 99.90%, Test Acc: 96.63%, Grad Norm: 0.3676
Epoch [6/10] - Loss: 0.0137, Train Acc: 99.30%, Test Acc: 94.94%, Grad Norm: 0.3278
Epoch [7/10] - Loss: 0.0251, Train Acc: 99.40%, Test Acc: 95.51%, Grad Norm: 0.8003
Epoch [8/10] - Loss: 0.0227, Train Acc: 99.90%, Test Acc: 95.51%, Grad Norm: 0.7361
Epoch [9/10] - Loss: 0.0131, Train Acc: 100.00%, Test Acc: 96.63%, Grad Norm: 0.3504
Epoch [10/10] - Loss: 0.0105, Train Acc: 100.00%, Test Acc: 97.19%, Grad Norm: 0.2882
```



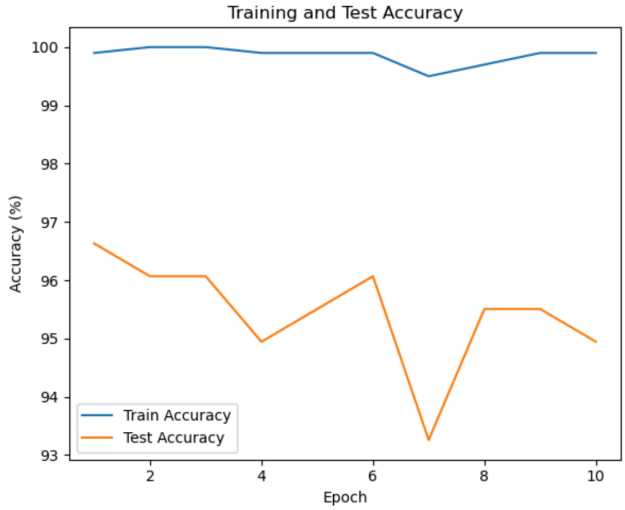
L1 Regularisation and Dropout:

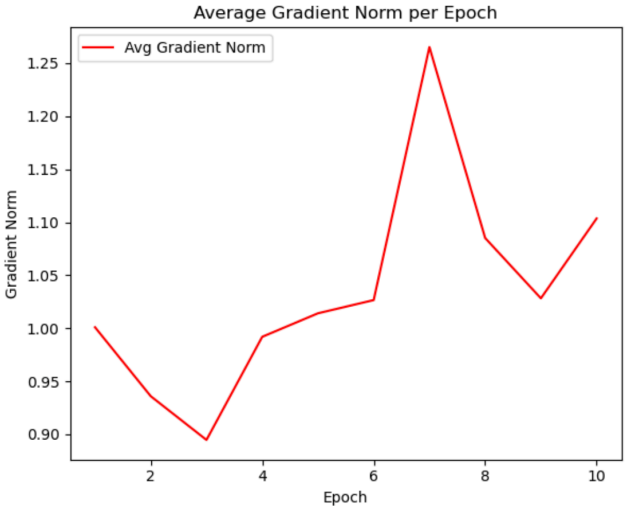
```
Training with L1 regularization (λ1=0.001)
Epoch [1/10] - Loss: 6.4010, Train Acc: 99.60%, Test Acc: 94.38%, Grad Norm: 1.2302
Epoch [2/10] - Loss: 3.6390, Train Acc: 96.70%, Test Acc: 90.45%, Grad Norm: 2.1425
Epoch [3/10] - Loss: 2.8268, Train Acc: 98.00%, Test Acc: 92.13%, Grad Norm: 2.0695
Epoch [4/10] - Loss: 2.3520, Train Acc: 97.40%, Test Acc: 92.70%, Grad Norm: 1.9917
Epoch [5/10] - Loss: 1.9723, Train Acc: 98.40%, Test Acc: 92.70%, Grad Norm: 1.7265
Epoch [6/10] - Loss: 1.7031, Train Acc: 99.50%, Test Acc: 95.51%, Grad Norm: 1.5268
Epoch [7/10] - Loss: 1.4994, Train Acc: 99.70%, Test Acc: 94.38%, Grad Norm: 1.3047
Epoch [8/10] - Loss: 1.3742, Train Acc: 99.80%, Test Acc: 95.51%, Grad Norm: 1.2916
Epoch [9/10] - Loss: 1.2817, Train Acc: 99.50%, Test Acc: 96.63%, Grad Norm: 1.1778
Epoch [10/10] - Loss: 1.2009, Train Acc: 99.70%, Test Acc: 94.94%, Grad Norm: 1.2806
```



L1 Regularization, L2 Regularization and Dropout:

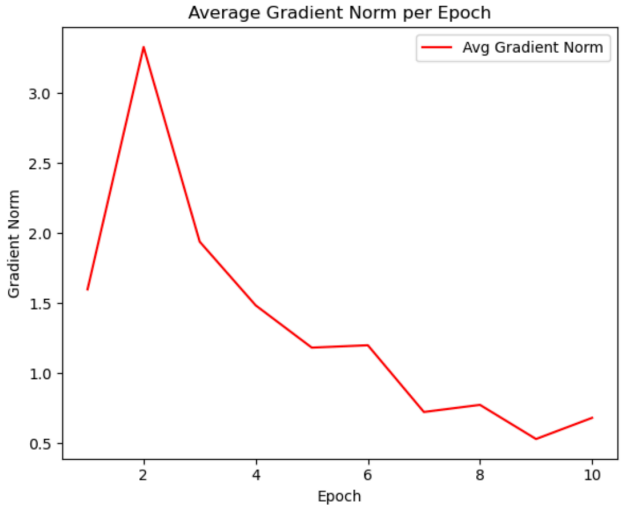
```
Training with L1, L2, Dropout regularization (λ2=0.001)
Epoch [1/10] - Loss: 1.7688, Train Acc: 99.90%, Test Acc: 96.63%, Grad Norm: 1.0010
Epoch [2/10] - Loss: 1.3596, Train Acc: 100.00%, Test Acc: 96.07%, Grad Norm: 0.9359
Epoch [3/10] - Loss: 1.1612, Train Acc: 100.00%, Test Acc: 96.07%, Grad Norm: 0.8947
Epoch [4/10] - Loss: 1.0747, Train Acc: 99.90%, Test Acc: 94.94%, Grad Norm: 0.9919
Epoch [5/10] - Loss: 1.0301, Train Acc: 99.90%, Test Acc: 95.51%, Grad Norm: 1.0141
Epoch [6/10] - Loss: 0.9938, Train Acc: 99.90%, Test Acc: 96.07%, Grad Norm: 1.0266
Epoch [7/10] - Loss: 0.9832, Train Acc: 99.50%, Test Acc: 93.26%, Grad Norm: 1.2652
Epoch [8/10] - Loss: 0.9520, Train Acc: 99.70%, Test Acc: 95.51%, Grad Norm: 1.0851
Epoch [9/10] - Loss: 0.9190, Train Acc: 99.90%, Test Acc: 95.51%, Grad Norm: 1.0282
Epoch [10/10] - Loss: 0.9035, Train Acc: 99.90%, Test Acc: 94.94%, Grad Norm: 1.1036
```





L2 Regularization and Dropout:

```
Training with L2 regularization (λ2=0.001)
Epoch [1/10] - Loss: 1.9639, Train Acc: 86.40%, Test Acc: 74.72%, Grad Norm: 1.5984
Epoch [2/10] - Loss: 0.8282, Train Acc: 93.30%, Test Acc: 88.20%, Grad Norm: 3.3273
Epoch [3/10] - Loss: 0.5213, Train Acc: 97.90%, Test Acc: 95.51%, Grad Norm: 1.9389
Epoch [4/10] - Loss: 0.4151, Train Acc: 98.90%, Test Acc: 97.19%, Grad Norm: 1.4841
Epoch [5/10] - Loss: 0.3505, Train Acc: 99.30%, Test Acc: 95.51%, Grad Norm: 1.1828
Epoch [6/10] - Loss: 0.3262, Train Acc: 99.70%, Test Acc: 96.63%, Grad Norm: 1.1998
Epoch [7/10] - Loss: 0.2878, Train Acc: 100.00%, Test Acc: 96.07%, Grad Norm: 0.7238
Epoch [8/10] - Loss: 0.2723, Train Acc: 99.90%, Test Acc: 96.07%, Grad Norm: 0.7746
Epoch [9/10] - Loss: 0.2530, Train Acc: 99.80%, Test Acc: 95.51%, Grad Norm: 0.5310
Epoch [10/10] - Loss: 0.2428, Train Acc: 99.70%, Test Acc: 96.07%, Grad Norm: 0.6821
```



Conclusion

After the experiment, I came to the conclusion that L2 regularization along with Dropout works best to maximize the accuracy and minimize the loss. Introducing L1 to the network introduces significant loss that offsets the gains achieved by having L2 and dropout in the network architecture.