Multi-layer Neural Networks to classify CIFAR-100 Dataset

Nishanth Rachakonda
Department of Computer Science
University of California, San Diego
nrachakonda@ucsd.edu

Soumya Ganguly
Department of Mathematics
University of California, San Diego
s1gangul@ucsd.edu

Abstract

Multilayered neural networks are implemented to classify images from CIFAR-100 dataset. These multilayered networks compute the gradients of the parameters effectively using backpropagation. Techniques such as momentum, regularization are used for training the models with a test accuracy of 28.38% and 29.98%. We also use various activation functions like 'sigmoid' and 'ReLU' with test accuracies of 24.73% and 33.86%. Furthermore, we test the model with various topologies by halving and doubling the number of hidden units and increasing the number of hidden layers with test accuracies of 33.24%, 35.50% and 31.17% respectively. We obtain a test accuracy of 21.78% when trained on 100 classes.

1 Introduction

Deep learning models have proven to be effective in modeling complex real-world data and have found widespread use in a variety of applications. However, to model complex data effectively, these learning models require multiple hidden layers, which can result in a large number of parameters to optimize. The backpropagation algorithm computes gradients for these parameters efficiently using the chain rule. The algorithm starts with the final layer, computing the gradients for each parameter, then uses these gradients to compute the gradients for the previous layer until the gradients for all layers have been computed. This makes backpropagation an effective algorithm. In this project, we calculated gradients using backprop and verified using a numerical approximation of gradients, that these values agree with each other up to $O(\epsilon^2)$ if ϵ is the infinitesimal parameter used to calculate the numerical gradient. This shows a high degree of accuracy for the backprop model.

Our model is trained to classify CIFAR-100 dataset, a huge dataset with images of everyday objects. Many methods were used to get a reasonably efficient image classifier that classifies our dataset into 20 'superclasses' or '100 classes'. We started with a 1-hidden layer neural network and used gradient descent with Momentum so that the model is resistant to noise and oscillations while optimizing the parameters. We got 28.38% test accuracy after performing tests in this method, with hyper tuned parameters. It is evident that a large number of trainable parameters will cause the model to overfit on the training dataset. Hence, we performed Regularization that constrains the model parameters to avoid this problem of overfitting. This also enables the model to not learn from the noise in the train data. We got the best regularization constant λ to be 10^{-2} with the test accuracy of 29.98%. We trained the hidden layer of our model with different activation functions like 'tanh', 'ReLU' and 'sigmoid'. We found there that 'ReLU' worked the best for our classification problem considered here, giving us a test accuracy of 33.86%. We also changed the number of hidden units a compare the efficiencies of different neural network models. We got the best test accuracies with 256 hidden units in the hidden layer. We saw that adding an additional hidden layer while keeping the number of parameters the same, drops the test accuracy of the 'ReLU' system before to 31.17%. In the end, we classified our image dataset into 100 classes with our 'ReLU' classifier and got 21.78% test accuracy.

2 Related Work

In the lecture slides of CSE 251B by Cottrell [1], the theory behind backpropagation is discussed which provides us with the key formulas being used in this project for computing gradients. In [2], the same author has introduced concept of regularization. The theory and techniques for using Momentum in gradient descent, batch normalization, Shuffling are discussed in [3].

3 Data loading

3.1 A brief description of data

The CIFAR-100 dataset (Canadian Institute for Advanced Research, 100 classes) is a an image dataset consisting of 5000 train and 1000 test images, each of 32 X 32 X 3 size. This is a popular dataset used frequently for training multiple deep learning models. Each image comes with a ”fine” label (the class to which it belongs) and a ”coarse” label (the superclass to which it belongs). There are 100 classes in the CIFAR-100 that further belong to 20 superclasses.

Some randomly sampled images of CIFAR-100 dataset are shown in Figure 1.

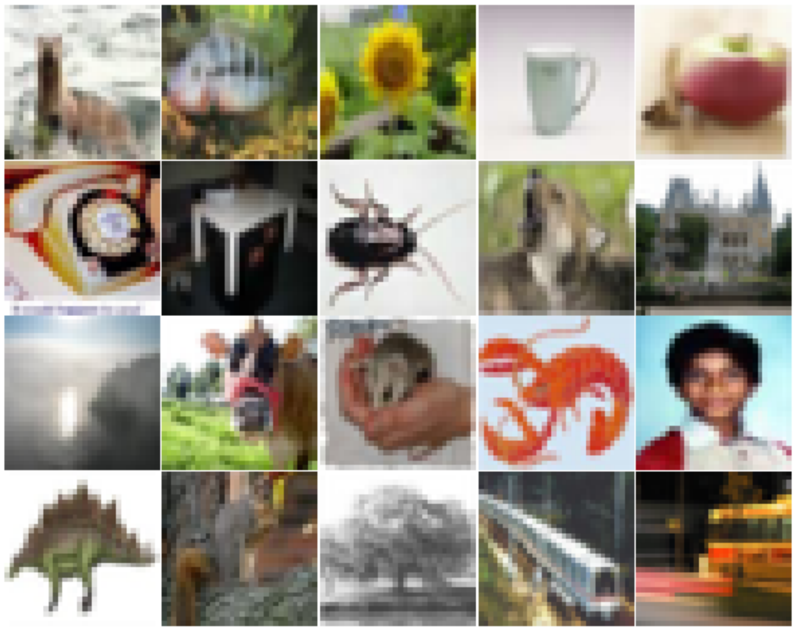

Figure 1: Random examples of different image classes in CIFAR-100.

3.2 Dataset Splitting and Normalization

Training dataset is split into two subsets train set and validation set of sizes in the ratio 80:20. Training dataset is shuffled randomly before making the split.

The dataset is then normalized using z score normalization method on a per channel per image basis. Each sample point is a flattened vector of an image matrix with three channels red, green and blue of size 32 X 32 X 3 = 3072. Let us call a sample image vector X. Then we have:

$$X = [X_1, X_2, X_3]$$

where each X_i represents a channel of the image vector X. We then calculate, for each $i \in \{1, 2, 3\}$:

$$\hat{X}_i = \frac{X_i - \mu_i}{\sigma_i}$$

where μ_i and σ_i are the mean and standard deviation of the vector X_i respectively. We then restack these normalized channels to form our normalized image per channel as $\hat{X} = [\hat{X_1}, \hat{X_2}, \hat{X_3}]$.

In the following table 1 the means and standard deviations of each of the channels of a sample training image can be found.

Table 1: Channel wise Mean and standard deviation of a training sample

Image Channel	**Mean**	**Standard Deviation**
Red	129.19	68.13
Green	124.09	65.35
Blue	112.48	70.42

4 Numerical Approximation of Gradients

4.1 Method

The backpropagation method is used for efficiently computing the gradients used in the weight update rule. First recall, for gradient descent's update rule, we have to calculate the gradient of the cost function with respect to the weights. This can be simplified using the notation below:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}}$$
$$\frac{\partial E}{\partial a_j} = -\delta_j, \frac{\partial a_j}{\partial w_{ij}} = z_i \text{ (say)}$$

Backpropagation now is given by the following set of equations, involving the δ_js:

$$\delta_j = \begin{cases} t_j - y_j, & \text{For output layer} \\ g'(a_j)\sum_k \delta_k w_{jk}, & \text{For } j^{th} \text{ hidden layer, where } k \text{ is the immediate next outer layer} \end{cases}$$

The gradient can be calculated using an approximation from the Taylor expansion of the cost function. The numerical approximation of the gradient can be given by the following equation.

$$\frac{\partial E^n(w)}{\partial w} \approx \frac{E^n(w+\epsilon) - E^n(w-\epsilon)}{2\epsilon}$$

This gradient is within $O(\epsilon^2)$ error. We use this numerical approximation method to verify the gradients calculated using the backpropagation method.

4.2 Results

We compare the gradients obtained from the backpropagation formula above and the gradient approximation rule, in table 2. We use $\epsilon = 10^{-2}$ for computing the numerically approximated gradient.

4.3 Discussion of Results

From the table above we can see that the absolute difference between numerically approximated gradients and the gradients obtained from back-propagation are at least $O(\epsilon^2)$, which indicates a great level of accuracy of gradients that are obtained using back-propagation.

Table 2: Table for Comparison of Gradients

TYPE OF WEIGHT	NUMERICAL GRADIENT	BACKPROP. GRADIENT	ABSOLUTE DIFFERENCE
Hidden to output-1	0.051357009026120615	0.05136377125292402 5	6.762×10^{-06}
Hidden to output-2	0.0066053585788816704	0.006605356943060802	1.636×10^{-09}
Input to hidden-1	-9.47392×10^{-06}	$-9.472398 \times 10^{-06}$	1.5198×10^{-09}
Input to hidden-2	-1.90081×10^{-09}	-1.90079×10^{-09}	1.8127×10^{-14}
Hidden bias	9.750867×10^{-10}	$9.7501865 \times 10^{-10}$	6.8024×10^{-14}
Output bias	0.0509703	0.0509695	7.24×10^{-07}

5 Neural Network with Momentum

5.1 Method

We use a special update rule for gradient descent which uses momentum. The momentum method damps oscillations of the gradient update rule, in directions of high curvature by combining gradients with opposite signs. It leads to the convergence of gradient descent in a gentle, consistent way. The momentum update steps can be described as:

$$v_{t+1} \leftarrow \gamma v_t - \frac{\alpha}{N}\frac{\partial E}{\partial w_t}$$
$$w_{t+1} \leftarrow w_t + v_{t+1}$$

where $v_0 = 0$. Here α is the learning rate, N is the mini-batch size and γ is the momentum parameter. In these experiments, we set $\gamma = 0.9$.

5.2 Training and Results

We use a 1-hidden layer neural network with 128 hidden units. We used the validation set for the early stopping of our training, i.e. we stopped training when the error on the validation set goes up for some "patience" number of epochs. After we stop training, we save the weights which resulted in a minimum validation error. The patience parameter could be 5, but we treated it as a hyperparameter and we got the best result for the patience number of epochs=10.

For hyperparameter tuning we first chose the hyperparameters in a particular order. Then we kept on tuning our model with one hyperparameter at a time, keeping the others constant. We keep momentum parameter γ constant at 0.9. We first tuned the learning rate, then batch size and in the end, we tuned the "patience" no. of epochs for early stopping. We chose the best set of hyperparameters for each one, by looking at the best validation accuracy. We chose the learning rates as $[10^{-2}, 10^{-3}, 10^{-4}]$, keeping our batch size=128, "patience" number of epochs=5. We got that for learning rate 10^{-3} we get the best validation accuracy. So then keeping our learning rate as 10^{-3} from now on, we start tuning our batch-size from the list $[64, 128, 256]$. We got the best result for batch-size 128. Then finally we ran our code for "patience" no. of epochs as 5 and 10 and got the best result for 10.

Some of the choices of hyperparameters are shown in the following table along with the best validation accuracy they gave.

Based on the experiments we conducted by tuning the hyperparameters we found that for 'tanh' activation function in the hidden layer, in our 1 hidden layer model, we get the best set of hyperparameters as :

learning rate=0.001, mini batch-size=128, and "patience" number of epochs in early epoch=10. In the following, we give loss and accuracy plots for the model with these tuned sets of hyperparameters.

The test accuracy we obtained from this tuned set of hyperparameters is = 28.38%

Table 3: Table for comparing model accuracy for various hyperparameters

LEARNING RATE	BATCH SIZE	PATIENCE	BEST VALIDATION ACCURACY(%)
10^{-3}	128	5	28.56
10^{-2}	128	5	27.61
10^{-4}	128	5	16.17
10^{-3}	64	5	28.56
10^{-3}	256	5	28.53
10^{-3}	128	10	28.60

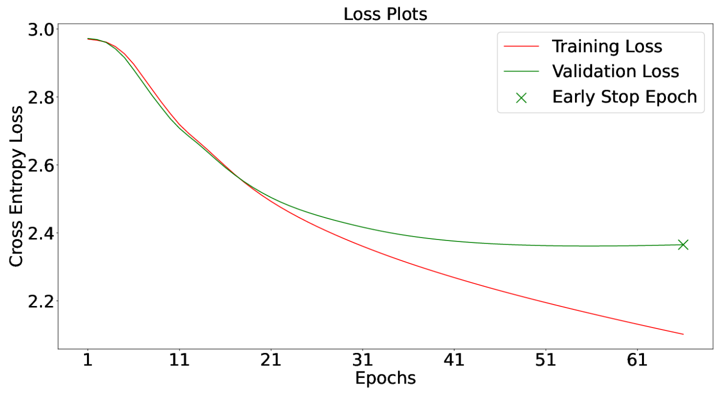

Figure 2: Training and Validation loss plots up to early stopping for a. learning rate=0.001, b. mini-batch size=128, c. ”patience” number of epochs=10, with momentum parameter $\gamma = 0.9$ and without regularization

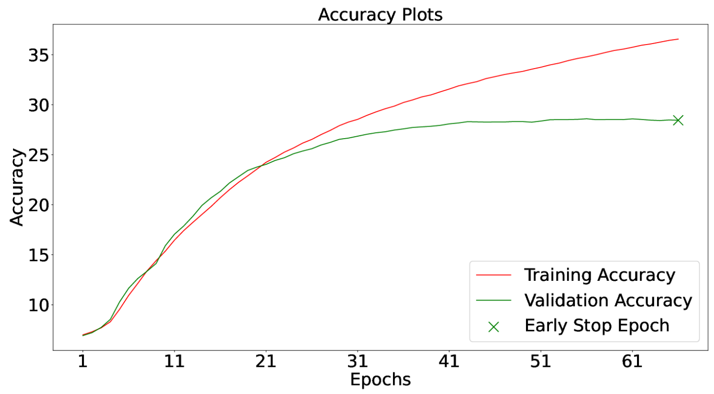

Figure 3: Training and Validation accuracy plots up to early stopping for a. learning rate=0.001, b. mini-batch size=128, c. ”patience” number of epochs=10, with momentum parameter $\gamma = 0.9$ and without regularization

5.3 Discussion of Results

From 3, it can be observed that the optimal learning rate is 10^{-3}, when the learning rate is increased to 10^{-2} the model does not converge to the optimal parameters. However, when the learning rate is decreased to 10^{-4} the model converges slowly to the optimal parameters. It can be observed that the best validation accuracy is approximately the same for different batch sizes. However, we found that batch size of 128 provides the best validation accuracy. It can be observed that tuning the "patience" number of epochs generates best result at 10 epochs. However, on increasing this hyperparameter further does not improve the validation accuracy.

6 Regularization Experiments

6.1 Method

Regularization of the weights is a technique in machine learning that discourages learning a more complex or flexible model by controlling the magnitude of the weights, so as to avoid the risk of overfitting. We use L_2 regularization here which penalizes higher weights more. We keep our loss function as the standard cross entropy loss only, so that regularization only affects the update step of the weights. The update rule with both momentum and L_2 regularization looks like the following:

$$v_{t+1} \leftarrow \gamma v_t - \frac{\alpha}{N}\frac{\partial E}{\partial w_t} - 2\lambda\alpha w_t$$

$$w_{t+1} \leftarrow w_t + v_{t+1}$$

where $v_0 = 0, \alpha, \gamma, N$ are same as in 5.1. The parameter λ decides the amount of regularization and is treated as a hyperparameter in the experiments shown below.

6.2 Results

In section 5.1, we did experiments with the maximum number of epochs as 100. In this section, we do experiments with 10% more epochs than the last one i.e. we train the network with 110 maximum number of epochs. However, early stopping is still present. Keeping the optimal parameters from the last section the same, we chose different values of λ to pick the best or the most optimal one based on the validation accuracies. We write the observations of our experiments in a tabular format as below:

Table 4: Table for comparing the effect of choosing different values of λ on validation accuracy

CHOICE(s) of λ	**BEST VALIDATION ACCURACY($\%$)**
10^{-2}	31.02
10^{-3}	30.12
10^{-4}	29.03

From the table above we see for $\lambda = 0.01$ we get the best validation accuracy. Hence we plot the training and validation loss, and training and validation accuracy for this λ, in 9.2:

The test accuracy we obtained for $\lambda = 0.01$, learning rate=0.001, mini-batch size=128, "patience" number of epochs=10, with momentum parameter $\gamma = 0.9$ is : 29.98%.

6.3 Discussion of Results

We see from the results above that as λ decreases, the validation accuracies decrease as well. This is because as λ is too low, our model becomes more complex and we run into the risk of overfitting the data. $\lambda = 10^{-2}$ seems to give us a balanced model that learns 'enough' (without over/underfitting the data) from the training data.

One important observation here is: unlike 5.1, there is no early stopping happening in this case, when we have regularization. This is because early stopping essentially avoids model overfitting, which is something done by regularization itself. Hence early stopping is not required here. Moreover, compared with the last section, where we did not have regularization, we can clearly see an improvement in test accuracy. This means without regularization, our model was overfitting and now our model has become better at predicting labels for test data.

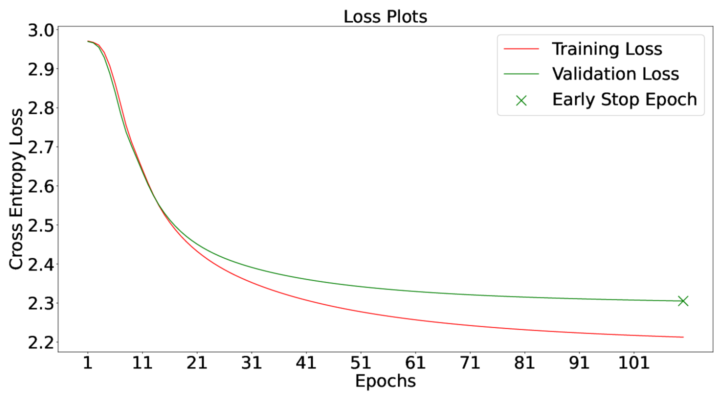

Figure 4: Training and Validation loss plots up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. ”patience” number of epochs=10, with momentum parameter $\gamma = 0.9$ L_2 regularization with penalty (λ)=0.01

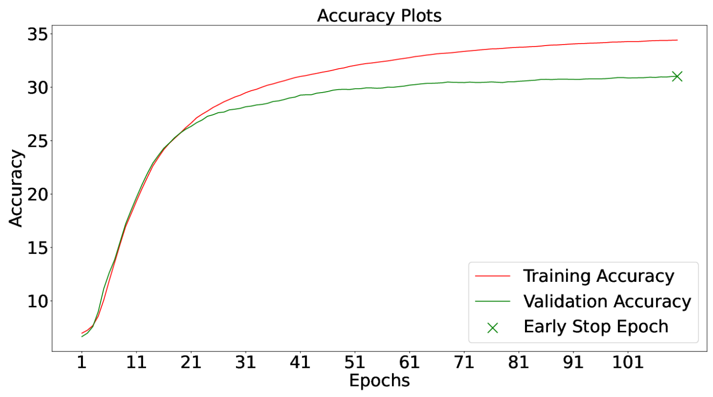

Figure 5: Training and Validation accuracy plots up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. ”patience” number of epochs=10, with momentum parameter $\gamma = 0.9$ and L_2 regularization with penalty (λ)=0.01

7 Activation Experiments

7.1 Method

In deep learning, the three most popular non-linear functions to be used are defined in the following. We mention their derivative also those derivatives are used during the backpropagation step. They are defined in the following:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \text{ with } \tanh'(z) = 1 - \tanh(z)^2$$
$$sigmoid(z) = \sigma(z) = \frac{1}{1+e^{-z}} \text{ with } \sigma'(z) = \sigma(z)(1-\sigma(z))$$
$$ReLU(z) = max(0, z) \text{ with } ReLU'(z) = 1 \text{ for } z > 0 \text{ and } = 0 \text{ otherwise}$$

In the experiments above, we worked with the activation function ’tanh’ in the hidden layer. But now, to compare the efficiency of different activation functions in this model, we use ’sigmoid’ and ’ReLU’ activation functions in the hidden layer.

7.2 Results

We keep all the parameters (learning rate, momentum, regularization, mini-batch size, ”patience” number of epochs, L_2 penalty, and the number of epochs) the same as in the experiments conducted in the section 6 above except for the activation functions in the hidden layer.

We plot the losses and accuracies as before for the activation functions ’sigmoid’ and ’ReLU’ in the plots 7.2.

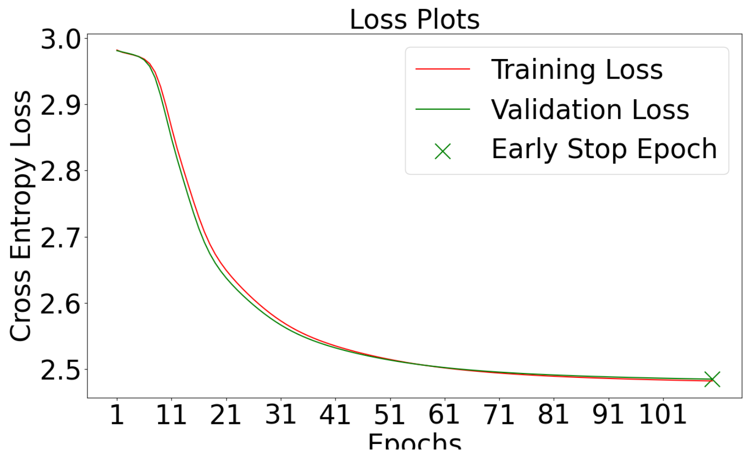

Figure 6: Training and Validation loss plots with ’sigmoid’ hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. ”patience” number of epochs=10, with momentum parameter $\gamma = 0.9$ and L_2 regularization with penalty (λ)=0.01

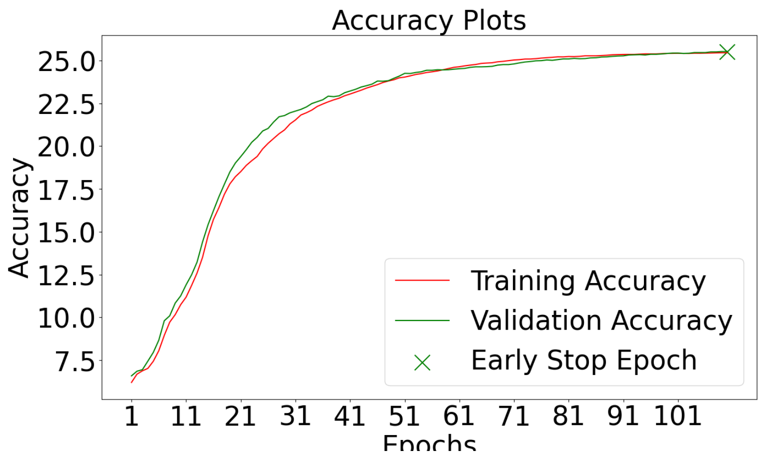

Figure 7: Training and Validation accuracy plots with ’sigmoid’ hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. ”patience” number of epochs=10, with momentum parameter $\gamma = 0.9$ and L_2 regularization with penalty (λ)=0.01

In the table below, we compare the test accuracies for different activation functions and we observe that test accuracy is highest for ReLU activation function.

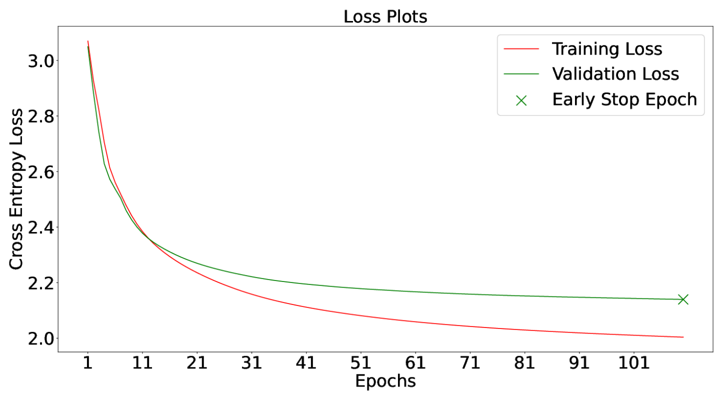

Figure 8: Training and Validation loss plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. "patience" number of epochs=10, with momentum parameter $\gamma = 0.9$ and L_2 regularization with penalty (λ)=0.01

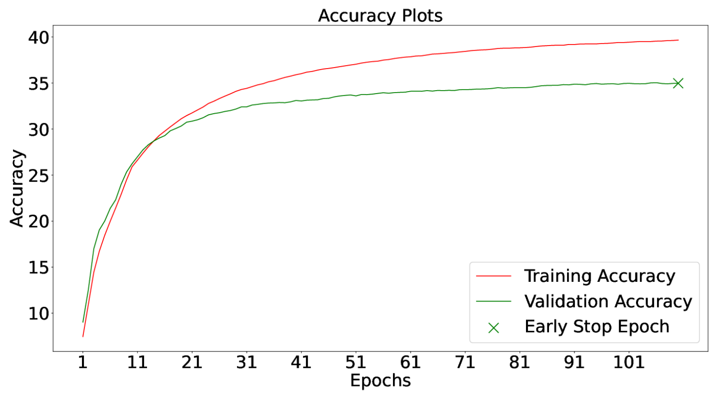

Figure 9: Training and Validation accuracy plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. "patience" number of epochs=10, with momentum parameter $\gamma = 0.9$ and L_2 regularization with penalty (λ)=0.01

Table 5: Table for comparing activation functions

ACTIVATION FUNCTION	**TEST ACCURACY(%)**
tanh	29.98
sigmoid	24.73
ReLU	33.86

7.3 Discussion of Results

As mentioned above, we see the best final test accuracy is obtained for the "ReLU" activation function. The worst test accuracy is for the activation function "sigmoid".

In many practical scenarios, 'sigmoid' is seen to be performing worse than 'tanh'. One of the reasons for that is, for 'sigmoid', the maximum value of the derivative is 0.25 but for 'tanh' it is 1-which means the update of weights will be much smaller for 'sigmoid', thus leading to slower convergence. On the other hand, for 'ReLU' the simplicity of the function and the fact that on the positive side of the function, the gradient never gets saturated- make it one of the best-performing hidden layer activation functions.

8 Network Topology Experiments

8.1 Changing Number of hidden units

8.1.1 Description

In the previous sections, we considered the number of hidden units to be 128. Here we keep the best network as obtained in the section 7 and only change the number of hidden units. We consider 64 and 256 hidden units respectively. These changes consequently change the dimension of weight matrices also.

8.1.2 Results

In the following, we give the plots for losses and accuracies as before, while only changing the number of hidden layer units to 64 and 256 respectively. We keep the other parameters of the network the same as the optimal one, obtained in the last section (which means we use the hidden activation function 'ReLU')

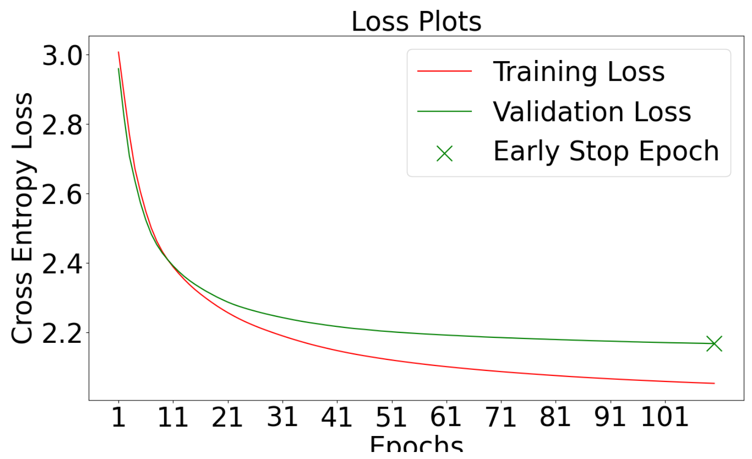

Figure 10: Training and Validation loss plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. "patience" number of epochs=10, d. momentum parameter $\gamma = 0.9$, e.L_2 regularization with penalty (λ)=0.01, f. No. of hidden units=64 -for classifying into 20 classes in CIFAR 100

We also report the final test accuracies obtained when we change the number of hidden units, in the following table. For completion, we write the result we obtained in the last section, with 128 hidden units also.

Table 6: Table for comparing the effect of changing the number of hidden units

NO. OF HIDDEN UNITS	**TEST ACCURACY (%)**
64	33.24
128 (original)	33.86
256	35.50

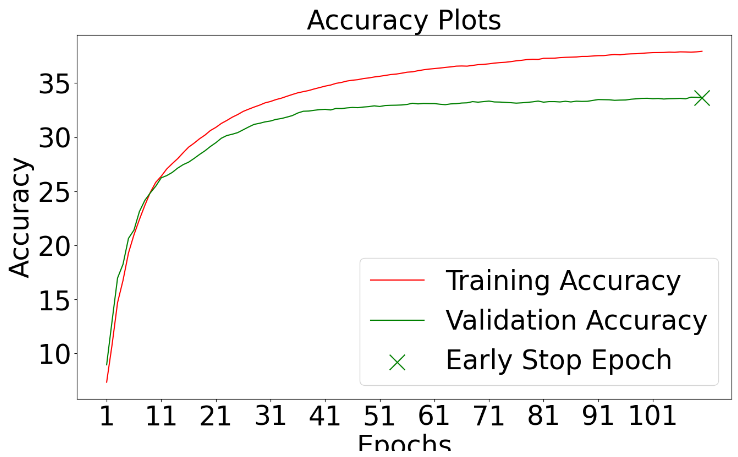

Figure 11: Training and Validation accuracy plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. "patience" number of epochs=10, d. momentum parameter $\gamma = 0.9$, e.L_2 regularization with penalty (λ)=0.01, f. No. of hidden units=64 -for classifying into 20 classes in CIFAR 100

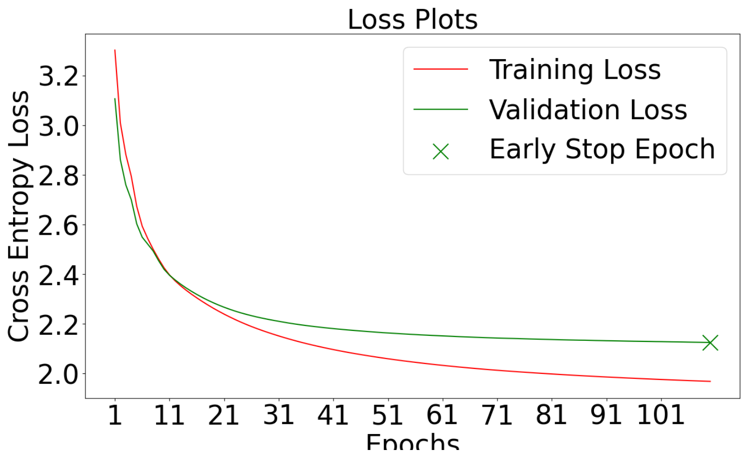

Figure 12: Training and Validation loss plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. "patience" number of epochs=10, d. momentum parameter $\gamma = 0.9$, e.L_2 regularization with penalty (λ)=0.001, f. No. of hidden units=256 -for classifying into 20 classes in CIFAR 100

8.1.3 Discussion of Results

We see when we had the number of hidden units =256, our performance was optimal. As we decrease the number of hidden units by a factor of two, our test accuracy from the model kept on decreasing. It is happening because a large dataset can be well represented with more parameters. We can expect a decline in this nature if we keep on increasing the number of hidden units as we will not have enough data to train this increased number of parameters.

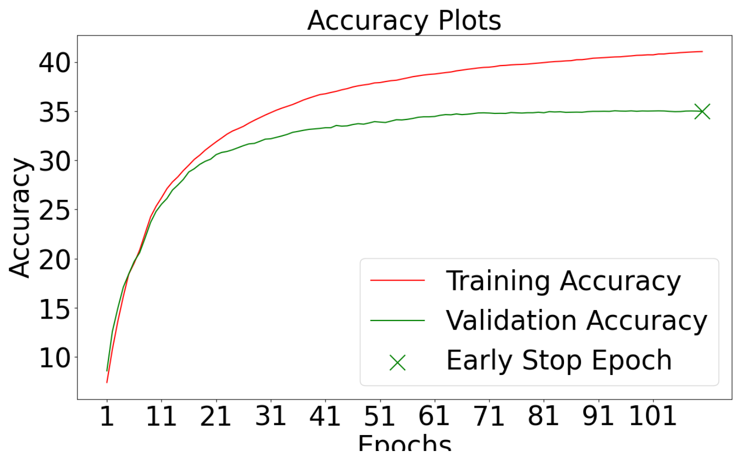

Figure 13: Training and Validation accuracy plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.01, b. mini-batch size=128, c. "patience" number of epochs=10, d. momentum parameter $\gamma = 0.9$, e.L_2 regularization with penalty (λ)=0.001, f. No. of hidden units=256 -for classifying into 20 classes in CIFAR 100

8.2 Changing number of hidden layers

8.2.1 Description

In the previous sections, we have constructed a model with one hidden layer of size 128. We add an additional hidden layer now such that the size of the two hidden layers is the same (x, say) and the number of parameters (i.e weights and the biases) is the same for this network, and the best configuration of the network obtained in the last section. The previous models have two matrix parameters to learn, input to the hidden layer (3073 X 128) and hidden layer to output layer (129 X 20). However, on adding an additional layer we have three matrix parameters to learn, input to hidden layer-1 (3073 X x), hidden layer-1 to hidden layer-2 ($x + 1$ X x), and hidden layer-2 to the output layer ($x + 1$ X 20). Equating these total number of parameters gives us:

$$\begin{aligned}
&\implies 3073 * 128 + 129 * 20 = 3073 * x + (x + 1) * x + (x + 1) * 20 \\
&\implies x = 123.0625 \\
&\implies x \approx 124
\end{aligned}$$

So we see if we don't want to change the number of parameters of the optimal model of the last section then we must make the two hidden layers in our new network with 124 units each.

8.2.2 Results

On adding two hidden layers of 124 units each and keeping all the other test parameters the same as in section 7, we get the plots below for losses and accuracies.

The test accuracy with this model is 31.17%.

8.2.3 Discussion of Results

It can be observed that making the model deeper does not improve the test accuracy. This could be attributed to the fact that the numerical value of the derivative of layers decreases significantly with the increase in the depth of the model. Hence, the weight parameters are updated very slowly in deeper networks. Moreover, the model is learning a representation of the data with almost the same

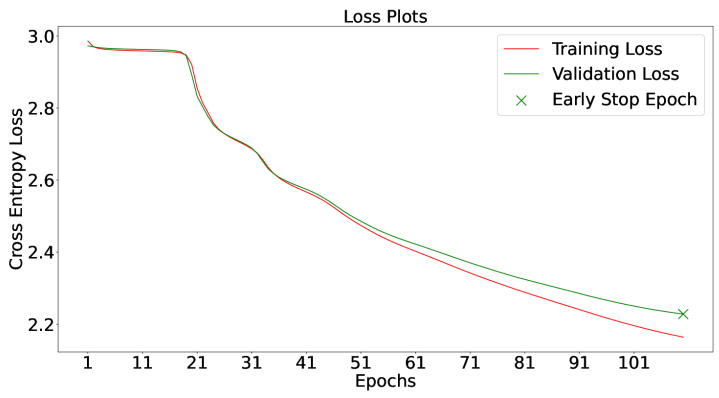

Figure 14: Training and Validation loss plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. "patience" number of epochs=10, d. momentum parameter $\gamma = 0.9$, e.L_2 regularization with penalty (λ)=0.001, f. Two hidden layers with 124 units each -for classifying into 20 classes in CIFAR 100

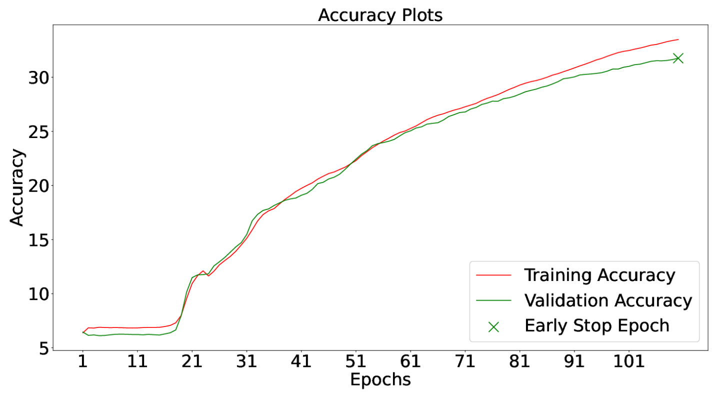

Figure 15: Training and Validation accuracy plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.01, b. mini-batch size=128, c. "patience" number of epochs=10, d. momentum parameter $\gamma = 0.9$, e.L_2 regularization with penalty (λ)=0.001, f. Two hidden layers with 124 units each -for classifying into 20 classes in CIFAR 100

number of parameters. Hence, it can be understood that the model test accuracy is almost the same as the best model from section 7.

9 Experiment with 100 classes

9.1 Method

As mentioned in section 3, CIFAR-100 has two sets of labels, coarse and fine labels. Coarse labels classify the dataset into 20 broader classes like aquatic mammals, fish, flowers, etc. where as the finer labels provide a narrower class definition such as beaver, dolphin, otter (for aquatic mammals), etc. For predicting the 100 fine labels on CIFAR-100 dataset, we use the best hyperparameters from section 7 with an output layer of size 100.

9.2 Results

The following two plots give us loss and validation curves as before, keeping the network parameters the same as the model from section 7.

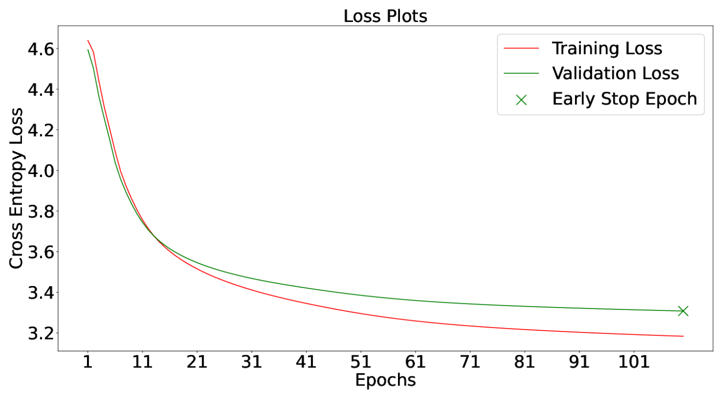

Figure 16: Training and Validation loss plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. "patience" number of epochs=10, d. momentum parameter $\gamma = 0.9$, e.L_2 regularization with penalty (λ)=0.01 -for classifying into 100 classes in CIFAR 100

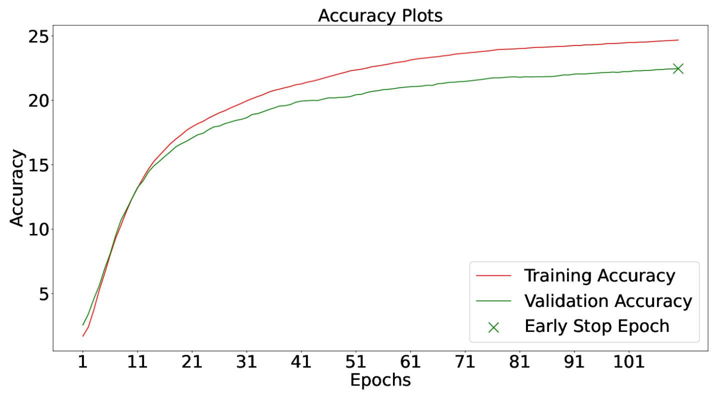

Figure 17: Training and Validation accuracy plots with 'ReLU' hidden activation, up to early stopping (or 110 epochs) for a. learning rate=0.001, b. mini-batch size=128, c. "patience" number of epochs=10, d. momentum parameter $\gamma = 0.9$, e.L_2 regularization with penalty (λ)=0.01 -for classifying into 100 classes in CIFAR 100

The test accuracy on CIFAR-100 for classifying into 100 classes, with our current model is 21.78%.

9.3 Discussion of Results

It can be observed the test accuracy dropped a lot when we tried to classify our dataset into 100 classes instead of 20 classes. This is because the number of parameters to be trained increases in the hidden to the output layer. However, the number of train samples do not increase in proportion. In order to improve the test accuracy, we can perform data augmentation using techniques like rotation, cropping, etc. These methods will increase the amount of data that can be used for training a large number of parameters.

10 Team Contributions

The project is a result of a team effort where both of us contributed as a group. Both ran the codes and provided necessary data for plotting and making tables in this report and both actively participated in hyperparameter tuning for the momentum method and finding optimal early stop epoch. However, roughly the contribution of each team member is listed below:

- Nishanth primarily worked on implementing codes for backpropagation, training loop, the general forward pass, integrating different parts of the code in main.py, checking backprop gradients with numerically approximated gradients, and L_2 regularization. He ran the experiments on checking gradients, creating a separate hidden layer, and classifying samples into 100 classes in CIFAR 100. In the project report, he wrote the abstract, introduction, 'Numerical Approximation of Gradients', 'Network Topology' and 'Experiment with 100 classes'.
- Soumya primarily worked on implementing the codes for implementing the activation functions' forward pass and gradients (in neuralnet.py), and data pre-processing (data loading, normalizing in util.py). He also implemented the algorithm for the momentum method for gradient descent. He ran the experiments for L_2 regularization, doubling and halving the number of hidden units, comparing the performance of different activation functions. He was also in charge of writing about the sections 'Dataset splitting and Normalization', 'Neural Network with Momentum', 'Regularization Experiments', and 'Activation Experiments' in the project report.

11 References

[1] Cottrell, G.W. (2023) Backprop: Representations, Representations,Representations *CSE 251B lecture slides*: 1-30.

[2] Cottrell, G.W. (2023) A Few Notes of Improving Generalization. *CSE 251B lecture slides*: 1-4.

[3] Cottrell, G.W (2023) PA1: Trick of the trade *CSE 251B discussion notes*: 11-64.