Experiment in Multi-Layers Neural Network of CIFAR-10 Dataset

Dongze Li
Department of Cognitive Science
University of California, San Diego
La Jolla, CA 92093
dol073@ucsd.edu

Xiaoyan He
Department of Mathematics
University of California, San Diego
La Jolla, CA 92093
x6he@ucsd.edu

Yunfan Long
Halıcıoğlu Data Science Institute
University of California, San Diego
La Jolla, CA 92093
yulong@ucsd.edu

Abstract

In this project, we explored the multi-layer neural network to classify images from different classes in CIFAR-10 dataset. We used softmax function as the activation function for output layer. In our original model, we use one hidden layer with 128 hidden units, and chose $\tanh$ function as the activation for hidden layer with learning rate as 1×10^{-5}, batch size as 128, momentum gamma as 0.9 with no regularization. We applied stochastic gradient descent in all parts of our experiment. With our original model, we reached the test accuracy of 47.36%. Then, in other parts of experiment, we tried L1 and L2 regularization, and also tried to use other activation function(ReLU & Sigmoid) for hidden units and change the number of hidden units and hidden layers. We achieved 50.77% as our highest test accuracy with ReLU activation for hidden units.

1 Introduction

We used Python to implement the neural network due to its concise grammar and various well-developed libraries. We built our original baseline model (which is shown in section 5) with three different layers, input layer, hidden layer, and output layer. Our goal in this experiment is to classify the 10 different classes of images from CIFAR-10. Since the task is multi-categories classification, we chose softmax fucntion as the activation for output layer. Using hidden layer can help the model to understand more complex data. We also applied momentum to get a more generalized data.

First, in our original model, we set the hidden units in hidden layer as 128, and the parameter for momentum as 0.9. We used mini-batch stochastic gradient descent for training process. We chose $\tanh$ as activation for hidden units. After tuning hyper-parameters, we finally reached test accuracy of 47.36 for learning rate as 1×10^{-5}, batch size as 128, and momentum gamma as 0.9

Then, we did more experiment that tried different types of regularization, activation functions and network topology. From these experiments, we found that L2 regularization has a better performance on the test. Also choosing ReLU as the activation for hidden units gives a more satisfied performance compare to the original model. Moreover, we found that a proper number of hidden units or hidden layers is an important component to deal with poor accuracy or overfitting.

2 Related Work

We acknowledge the great help from our instructor Garrison Cottrell for briefly explained the concept of softmax regression, back propagation, and valuable advice of regularization.[2][3][4][5]. We also thanks for the book "Dive into Deep Learning" [1] which gives a lot of insight on model construction and the vectorized update rule of back propagation. We also appreciate Krizhevsky for the dataset provided in his website [6][7].

[1] A. Zhang, Z.C. Lipton, M. Li, A.J. Smola. Dive into Deep Learning. *arXiv:2106.11342 [cs.LG], Jul 2022*

[2] G. Cottrell. *Lecture 2: Logistic Regression Multinomial Regression*, lecture notes, Department of Computer Science, University of California San Diego, Oct 2022.

[3]G. Cottrell. *Lecture 3: Back Propagation*, lecture notes, Department of Computer Science, University of California San Diego, Oct 2022.

[4]G. Cottrell. *Lecture 4: Improving Generalization*, lecture notes, Department of Computer Science, University of California San Diego, Oct 2022.

[5]G. Cottrell. *Lecture 4A: Tricks of Trade*, lecture notes, Department of Computer Science, University of California San Diego, Oct 2022.

[6] A. Krizhevsky. CIFAR-10 and CIFAR-100 Datasets, http://www.cs.toronto.edu/ kriz/cifar.html.

[7] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, 2009.

3 Dataset

3.1 Dataset Used

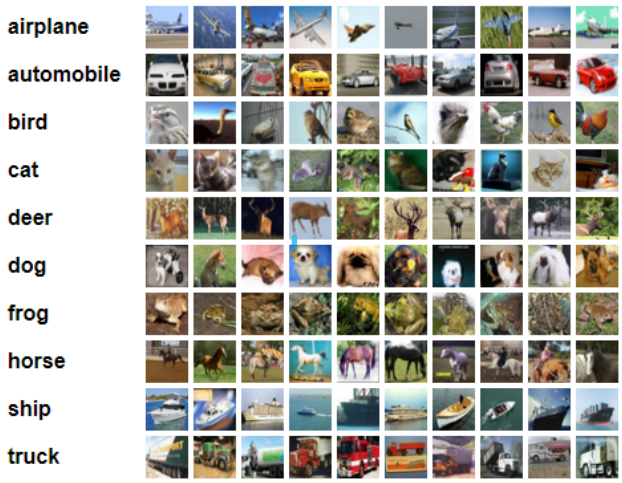


Figure 1: Sample CIFAR-10 data from each class

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. Table 1 below demonstrates the size of training data for the different classes

Table 1: Data statistics

Class	Training Set 1	Training Set 2	Training Set 3	Training Set 4	Training Set 5
0	1005	984	994	1003	1014
1	974	1007	1042	963	1014
2	1032	1010	965	1041	952
3	1016	995	997	976	1016
4	999	1010	990	1004	997
5	937	988	1029	1021	1025
6	1030	1008	978	1004	980
7	1001	1026	1015	981	977
8	1025	987	961	1024	1003
9	981	985	1029	983	1022

3.2 Splitting of Train and Validation Test Set

We perform an 80-20 split of our training data.

Training Set: The data we use to train the model - 80%

Validation Set: The data we use to measure our model's performance on unseen data & hyper-parameter Tuning - 20%

3.3 Normalization procedure

$$\widehat{a_i} = \frac{a_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

Figure 2: Formular for z-score normalization

After transposing data set to a N (the number of examples) $\times\, 3 \times 1024$ matrix, we apply z-score normalization to the dataset by finding the mean and std for each of the three channel for the entire dataset and z-score normalize them to mean 0, unit variance. After that, we concatenate the three different channel together to a $N \times 3072$ matrix.

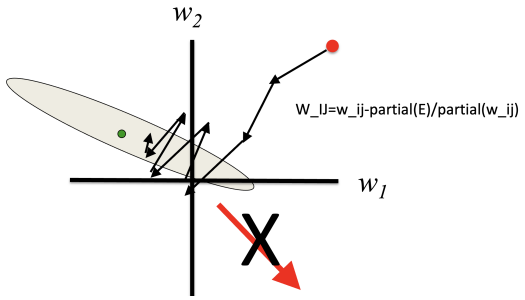


Figure 3: Gradient descent visualization when all inputs are non-negative

This is step is essential because by normalizing our data, we ensure that not all data are positive. In this way, we reduce the oscillation during gradient descent because if all data are positive, the route would be oscillating up and down, which slows down the learning overall.

3.4 Sample Mean & Std for a Training Image

Here, we reported our mean and standard deviation of one random sampled training image in our dataset. The mean of red channel is 111.43457, and standard deviation is 45.75286. The mean of green channel

is 111.20996, and standard deviation is 46.26216. The mean of blue channel is 111.17089, and standard deviation is 46.01742.

4 Numerical Approximation of Gradients

To check if our gradients obtained by back propagation is correct, we compare the gradient calculated by back propagation with numerical approximation of gradient given as following equation.

$$\frac{d}{dw}E^n(w) = \frac{E^n w + \epsilon - E^n w - \epsilon}{2\epsilon} \tag{1}$$

We selected one hidden bias, one output bias, two weights from input layer to hidden layer, and two weights from hidden layer to output layer, then calculated the absolute difference between the numerical approximation and gradient obtained by backprop.(See 2) We use the ϵ of 0.01 in this experiment.

Table 2: Data statistics

Type of Weight	Numerical Approximation	Gradient Obtained by Backprop	Absolute Difference
Output Bias	0.006107154372614332	0.00609510789399266	1.204×10^{-5}
Hidden Bias	0.0027465960634742714	0.0027466678709232363	7.180×10^{-8}
Input to Hidden i	−0.0023176866317697886	−0.0023179609604423294	2.743×10^{-7}
Input to Hidden ii	−0.0020177224840267627	−0.0020177228106122136	3.265×10^{-10}
Hidden to Output i	0.019621765771660193	0.019621753322845232	1.244×10^{-8}
Hidden to Output ii	0.15679421883021405	0.1567941526517481	6.617×10^{-8}

All of the absolute difference yielded by our experiment are with the ϵ^2.

5 Neural Network with Momentum

5.1 Training Procedure

In this section, we will show our work with the original model. We chose to use stochastic gradient descent with momentum to update our weight matrices that connect the layers. Also, we initialize our weight by random in all parts of the experiment. Through back propagation, we use following two updated rules to update the weight. For the weight between input layer and hidden layer, the update rule is

$$w_{ij} = w_{ij} + \alpha \sum_n \delta_j^n z_i^n \delta_j = -\frac{\partial E^n}{\partial a_j^n} = -g'(a_j^n) \sum_{k=1}^{c} \delta_k w_{jk} \tag{2}$$

where α is learning rate, δ_j^n is the delta that maps i-th unit from input to j-th hidden unit for n-th sample.

For the weight between hidden layer and output layer, the update rule is

$$w_{jk} = w_{jk} + \alpha \sum_n \delta_k^n z_j^n \delta_k = -\frac{\partial E^n}{\partial a_k^n} = t_k^n - y_k^n \tag{3}$$

Beyond that, we add momentum in all parts of experiment to against oscillating gradient effect by moving slowly in directions of large inconsistent gradients and quickly in directions of consistent gradients. With momentum added, the change of weight is shown as below

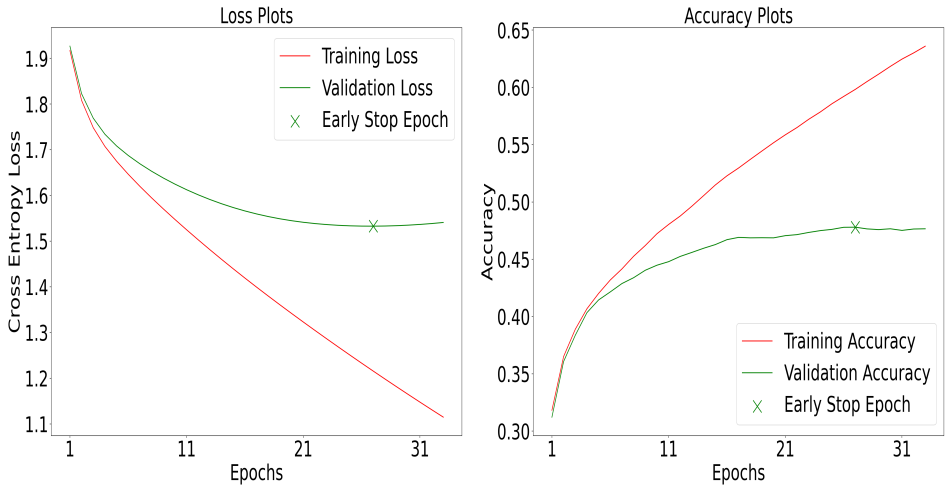
$$\Delta W(t) = \gamma \Delta W(t-1) - \epsilon \frac{\partial E}{\partial W}(t) \tag{4}$$

where $\Delta W(t)$ is current weight change and $\Delta W(t-1)$ is previous weight change.

In this model, we firstly do a forward pass of our samples with the size of minibatch through the network to calculate the loss and record it. Then we use the equations stated above to calculate the gradient and update our weight to optimize the model. We trained our model with activation function as tanh, learning rate as 1×10^{-5}, batch size as 128, momentum parameter $\gamma = 0.9$, no regularization, and a 128 units hidden layer.

5.2 Train vs. Validation

In general, the losses are getting lower, and the train loss is always lower than the validation loss.



(a) Training and validation loss

(b) Training and validation accuracy

Figure 4: Training performance with activation function as tanh, learning rate as 1×10^{-5}, batch size as 128, momentum parameter $\gamma = 0.9$, 128 units hidden layer, no regularization, 100 epochs

5.3 Test Performance

Our model with momentum achieves test accuracy of 47.36% and loss value around 1.555.

5.4 Result

We can observe that with the training accuracy increasing at a decreasing speed, the validation accuracy also keeps rising in a decreasing speed in the first 15 epochs, but after that the accuracy oscillates up and down at around 45% till the end (See figure 4), while the test accuracy reaches 47.36%, which might be even higher than the validation accuracy. Although this result is favorable, the underlying reason can be that we did 80-20 split instead of k-fold validation, and we might have a validation set that is more difficult to classify for our model. For the hyper-parameters, we tried multiple combinations. We first tried the batch size of 64 with the learning rate of 0.01, and we found that the test performance was poor because the network learnt too fast and over-fitted quickly. Then, we tried the batch size of 64 with the learning rate of 0.0001, and we found that the test accuracy improved but still not exactly what we expected. Hence, we increased the batch size to 128 with the learning rate of 0.00001 and re-trained the model. This time the performance was improved and the accuracy was constantly over 47% with reasonable training time. Therefore, we chose the batch size of 128 and the learning rate of 0.00001. We concluded with the hypothesis that the minibatch with size of 128 contains a sufficient number of varied samples that the model could learn. Also, if we want to approach the minimum point as close as we can, we should avoid to set a too large learning rate, like 0.01, since it make the model overfitted and trigger the early stop just after a few epochs and we can't furtherly make the model become more precise in prediction.

6 Regularization Experiments

6.1 Training procedure

In this part, we added the penalty of L1 and L2 regularization. Then our loss function become

$$J = E + \lambda C \tag{5}$$

where C is our measure of regularization.

We update L1 gradient with the following formula:

$$w_{ij} = w_{ij} - \frac{dE}{dw_{ij}} - \lambda sgn(w_{ij}) \tag{6}$$

We update L2 gradient with the following formula:

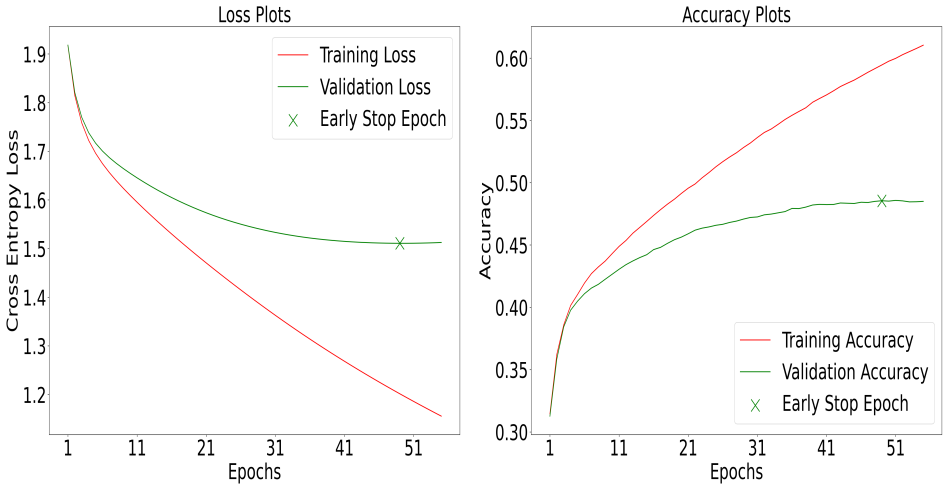
$$w_{ij} = w_{ij} - \frac{dE}{dw_{ij}} - \lambda w_{ij} \tag{7}$$

In theory, L1 regularization penalizes all weight the same, and L2 regularization penalizes big weights more.

We trained the model using both L1 and L2 regularization with $\lambda = 0.01$ and $\lambda = 0.0001$.

And we show the plots and our finding below.

6.2 L1 regularization with $\lambda = 0.01$



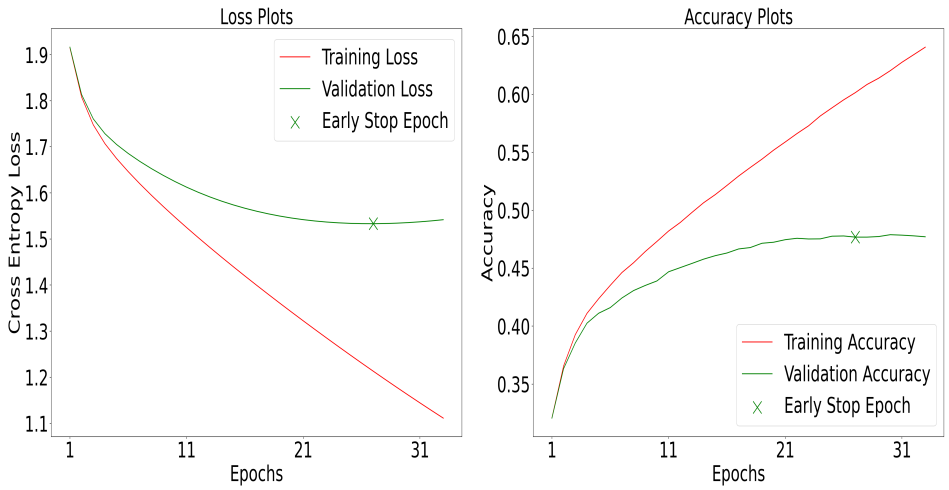
(a) Training and validation loss

(b) Training and validation accuracy

Figure 5: Training performance with learning rate as 1×10^{-5}, batch size as 64, momentum parameter $\gamma = 0.9$, L1 regularization parameter with $\lambda = 0.01$, 128 units hidden layer, 110 epochs

This L1 regularization model with $\lambda = 0.01$ achieves test accuracy of 47.22% and loss value around 1.554.

6.3 L1 regularization with parameter $\lambda = 0.0001$



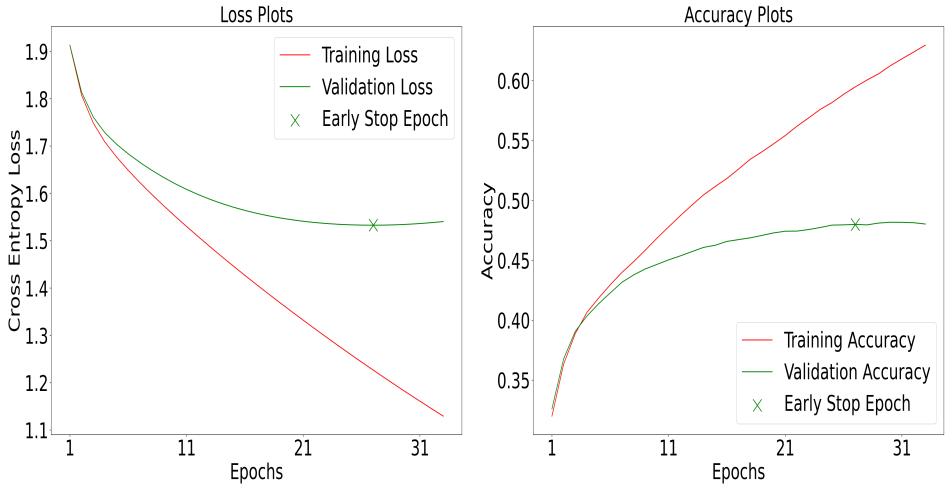
(a) Training and validation loss

(b) Training and validation accuracy

Figure 6: Training performance with learning rate as 1×10^{-5}, batch size as 64, momentum parameter $\gamma = 0.9$, L1 regularization parameter with $\lambda = 0.0001$, 128 units hidden layer, 110 epochs

This L1 regularization model with $\lambda = 0.0001$ achieves test accuracy of 47.40% and loss value around 1.549.

6.4 L2 regularization with parameter $\lambda = 0.01$



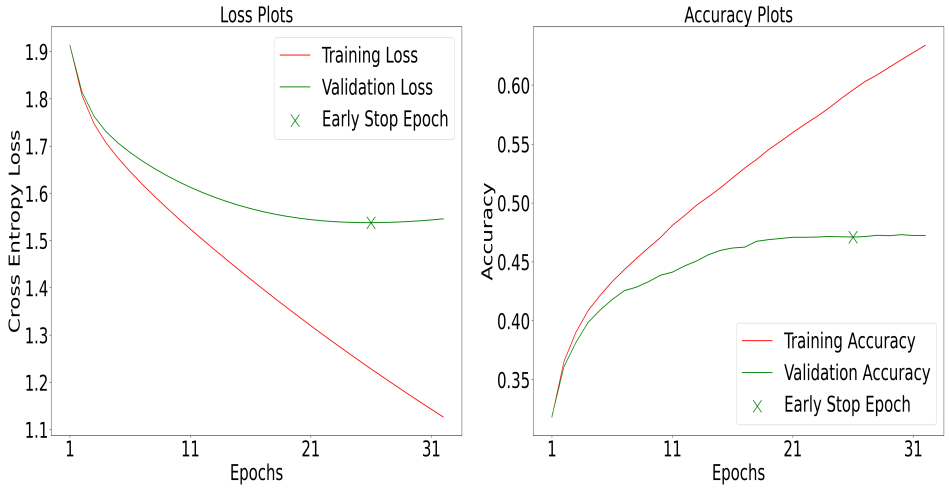
(a) Training and validation loss

(b) Training and validation accuracy

Figure 7: Training performance with learning rate as 1×10^{-5}, batch size as 64, momentum parameter $\gamma = 0.9$, L2 regularization with parameter $\lambda = 0.01$, 128 units hidden layer, 110 epochs

This L2 regularization model with $\lambda = 0.01$ achieves test accuracy of 47.42% and loss value around 1.532.

6.5 L2 regularization with $\lambda = 0.0001$



(a) Training and validation loss

(b) Training and validation accuracy

Figure 8: Training performance with learning rate as 1×10^{-5}, batch size as 64, momentum parameter $\gamma = 0.9$, L2 regularization with parameter $\lambda = 0.0001$, 128 units hidden layer, 110 epochs

This L2 regularization model with $\lambda = 0.0001$ achieves test accuracy of 47.74% and loss value around 1.548.

6.6 Observation: L2 vs. L1 Normalization

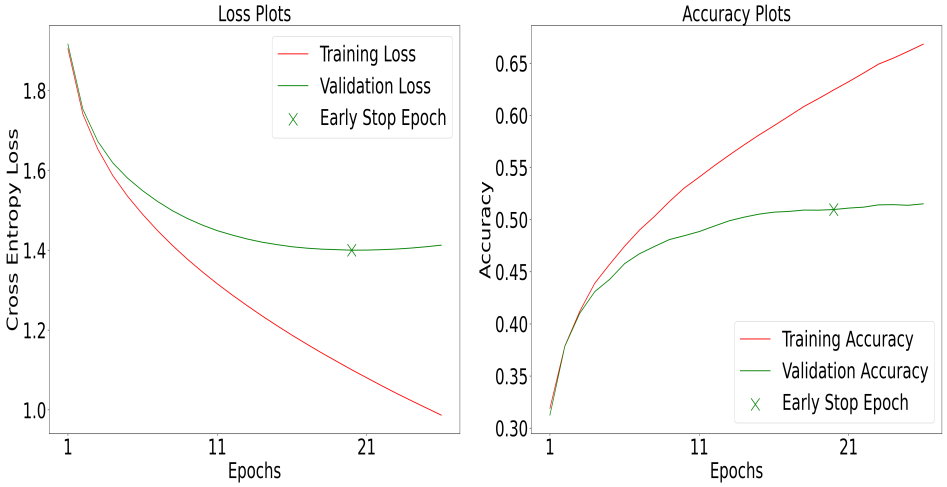
Controlling the regularization rate λ and comparing across the types of regularization, L2 regularization appears to achieve higher accuracy than the L1 regularization, although not by much. When the regularization parameter $\lambda = 0.01$, L2 regularization outperforms L1 regularization by 0.2% (See Figure 5 vs. 7). L2 regularization also beats L1 regularization by 0.34% (See Figure 6 vs. 8). One potential reason that L2 normalization worked better is that it places an outsize penalty on large components of the weight vector. This biases our learning algorithm towards models that distribute weight evenly across a larger number of features. By contrast, L1 regularization penalizes the same amount to all components of the features. This advantage of L2 regularization might make our model more robust to anomalies in the pictures, which gives us a slightly better accuracy.

6.7 Observation: 0.01 vs. 0.0001 Learning Rate

Controlling the type of regularization and comparing the regularization rate λ, both L1 and L2 regularization achieves a higher accuracy with 0.0001 instead of 0.01. L1 regularization with regularization rate of 0.0001 outperforms that with regularization rate of 0.01 by 0.18% (See Figure 5 vs. 6). L2 regularization with regularization rate of 0.0001 outperforms that with regularization rate of 0.01 by 0.32% (See Figure 7 vs. 8). Similar to the problem of too large learning rate, this makes sense because by using a lower regularization rate to, our gradient will bounce back an force in a smaller step around the local minima. Namely, a model with lower regularization rate makes our model converge in a place with lower loss and thus higher accuracy and setting the regularization rate λ too high might also cause the underfit of the model.

7 Activation Experiments

7.1 ReLu - Train vs. Validation



(a) Training and validation loss

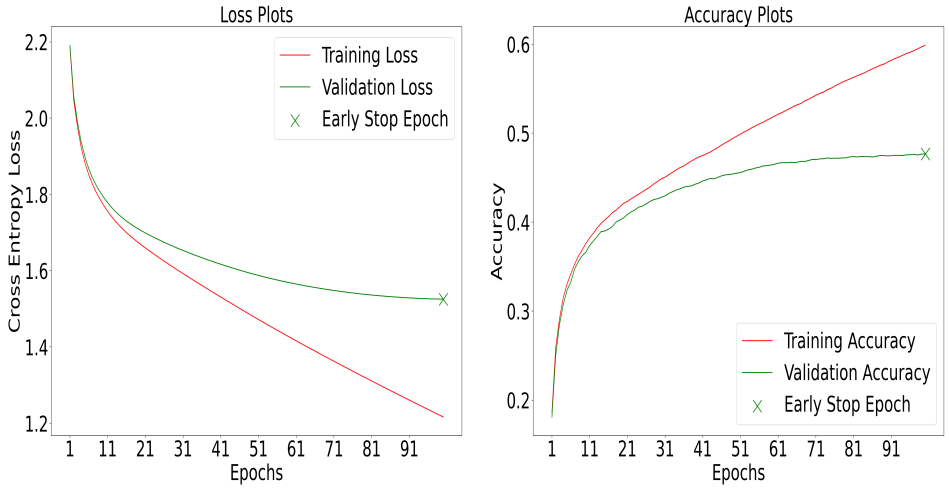
(b) Training and validation accuracy

Figure 9: Training performance with ReLu as activation function, learning rate as 1×10^{-5}, batch size as 64, momentum parameter $\gamma = 0.9$, 128 units hidden layer, 100 epochs

7.2 ReLu Model Final Test Accuracy & Comment

The final test result indicates that our Relu model has a loss of 1.411 and accuracy of 50.77% on test set. This is an about 3% increase in accuracy compare to previous tests. One possible reason for such result is that the ReLu activation function, unlike tanh, has direvative of either 1 or 0. In this case, it avoids the vanishing gradient problem, enables our network to learn at a much faster pace, and hence help our network reach higher accuracy. The relu model, compare to the previous tanh model with L2 regularization (See Figure 8 vs. 9), achieves more than 45% of validation accuracy in 5 epochs, while it takes our tanh model more than 11 epochs to reach 45%. What's more, after the first 20 epochs, the validation accuracy in the Relu model keeps increasing at a decreasing rate until it reaches 50%, while the tanh model stays around 45%.

7.3 Sigmoid - Train Accuracy vs. Validation Accuracy



(a) Training and validation loss

(b) Training and validation accuracy

Figure 10: Training performance with sigmoid as activation function, learning rate as 1×10^{-5}, batch size as 64, momentum parameter $\gamma = 0.9$, 128 units hidden layer, 100 epochs

7.4 Sigmoid Model Final Test Accuracy & Comment

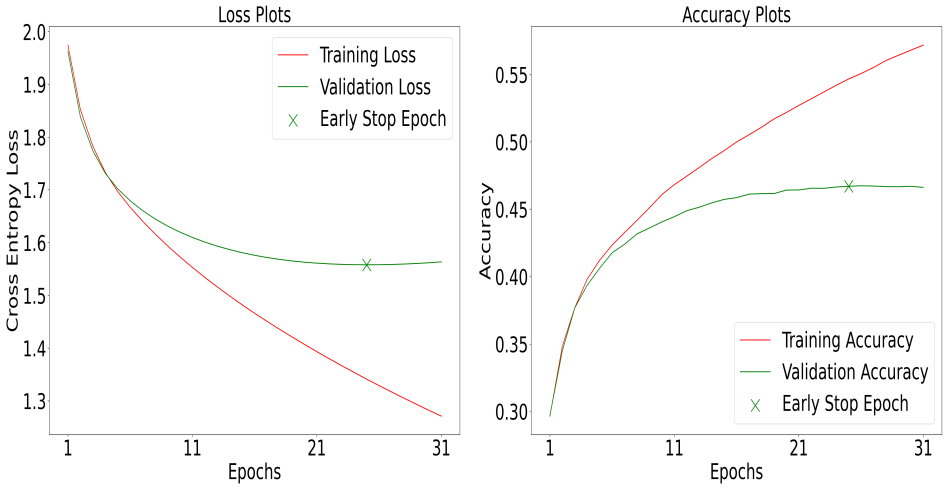
The final test result indicates that our Sigmoid model has a loss of 1.538 and accuracy of 47.29% on test set. Recall that in section 5, we get a final test accuracy of 47.36% with our tanh model (See figure 4). Unlike the Relu model that performs exceedingly well, test accuracy of this sigmoid model is lower than that of the tanh model by 0.07%. The sigmoid model finishes the whole 100 epochs without early stopping but ends up reaching a lower test accuracy than the tanh model, which runs only for about 25 epochs before early stopping. It can be observed that the sigmoid model is both inefficient and less accurate. One possible reason for such result is that although both the tanh and sigmoid function saturate in the lower and higher ends, sigmoid function turns all of the outputs to be non-negative, which makes the convergence take even longer because its gradient descent will have a lot of oscillation until convergence (See figure 3). This slows the training process of our sigmoid model.

8 Network Topology Experiments

Recall that in section 5, we trained the tanh model with 128 hidden units, 1×10^{-5} learning rate, and momentum parameter $\gamma = 0.9$, which achieves a final test accuracy of 47.36% with our tanh model (See figure 4). In the following experiment, we want to leave other parameters constants while halving and doubling the number of hidden units to see how the performance changes. After that, we want to add another hidden layer, both of size 237 which will give us the similar number of parameters with the network of single hidden layer with 256 units, to our original model to see how the performance changes.

8.1 Halving Hidden Units (128 to 64 units) - Train Accuracy vs. Validation Accuracy

The final test result indicates that our 64-hidden-unit model has a loss of 1.591 and accuracy of 45.66% on test set. This is lower than our original model with 128 hidden units. Generally, the more the hidden units, the more complex the function that can be learned by the network. Therefore, when we decrease the number of hidden units, the network can no longer learn the internal relation of the same complexity level, and hence our 64-hidden-unit model falls short for accuracy. However, this does not mean the more hidden units a network has, the better test accuracy will be. When our network trying to learn complex relation on relatively simpler problem, the prediction would likely be corrupted by nuances in the training data and thus lack of generalization on different test sets.

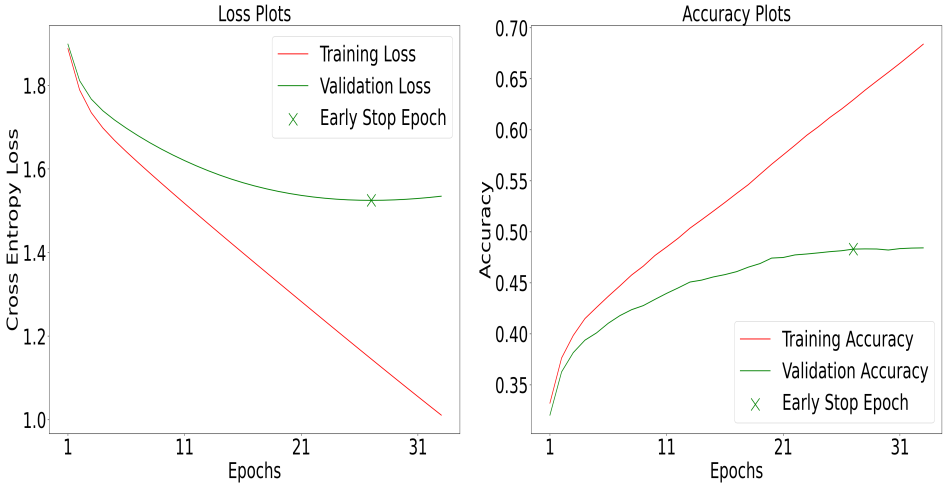


(a) Training and validation loss

(b) Training and validation accuracy

Figure 11: Training performance with 64 units hidden layer, tanh as activation function, learning rate as 1×10^{-5}, batch size as 64, momentum parameter $\gamma = 0.9$, 100 epochs

8.2 Doubling Units (128 to 256 units) - Train Accuracy vs. Validation Accuracy



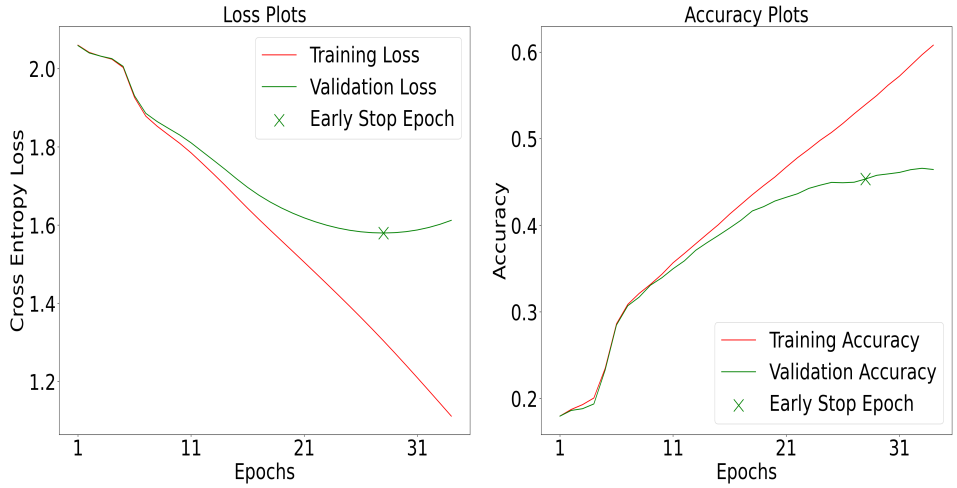
(a) Training and validation loss

(b) Training and validation accuracy

Figure 12: Training performance with 256 units hidden layer, tanh as activation function, learning rate as 1×10^{-5}, batch size as 128, momentum parameter $\gamma = 0.9$, 100 epochs

The final test result indicates that our 256-hidden-unit model has a loss of 1.520 and accuracy of 48.4% on test set. This is higher than our original model with 128 hidden units. Generally, the more the hidden units, the more complex the function that can be learned by the network. Therefore, when we increase the number of hidden units, the network can learn more complex internal relations, and hence our 256-hidden-unit model outperforms our original model.

8.3 Adding Another Hidden Layer (256 to 237 $\times$ 237) - Train Accuracy vs. Validation Accuracy



(a) Training and validation loss

(b) Training and validation accuracy

Figure 13: Training performance with 237×237 hidden units in hidden layer, tanh as activation function, learning rate as 1×10^{-5}, batch size as 128, momentum parameter $\gamma = 0.9$, 100 epochs

The final test result indicates that our 237×237-unit-hidden-layer model has a loss of 1.602 and accuracy of 45.59% on test set. This is much lower than our original model with only 256 hidden units. Although our network can learn more complex internal representation with more hidden layers, it backfires when complexity of the the problem itself does not require more complicated network with more hidden layers. As we can observe in the first 6 epochs, the training and validation accuracy completely stick together, which can be a sigh of over-fitting.

8.4 Conclusion & Inference

It's easy to make a mistake to believe that that the more hidden units, the better. However, this does not mean the more hidden units a network has, the better test accuracy will be. When our network trying to learn complex relation on relatively simpler problem, the prediction would likely be corrupted by nuances in the training data and thus lack of generalization on different test sets. As long as the problem is more complex than our current model can handle, increasing the number of hidden units will likely increase the prediction accuracy. Once our model is on par with the complexity of the problem, we should stop adding more hidden units or layers.

9 Contribution

Xiaoyan He, Dongze Li, and Yunfan Long equally distributed all the work in implementing the neural network and carrying out experiments, tuning hyperparameter, and debugging. We took turns that one wrote a test and the other made the test pass in the process of completing this project. During this process, we met together in library, took turn to serve as the primary programmer when constructed neural network, and conduct all the experiments together. During hyperparameter tuning, we all ran same amount of load during the two weeks.

Beyond the equally distributed works in implementing the neural network, every of us undertake some extra work in different areas:

Dongze Li: hyperparameter tuning of the momentum model and implementing code to setting up train() and test().

Xiaoyan He: hyperparameter tuning of the L1 regularization model finding appropriate parameters for regularization and implementing code to compute numerical gradient approximation.

Yunfan Long: hyperparameter tuning of the L2 regularization model and implementing code to improving visualization and management of result.

Note that we collaborate and help each other with debugging and testing in every aspect of this project along the way.

All of us contributed equally on writing the report.