Classification on CIFAR-10 using Backpropagation

Jared Zhang A15889667

Abstract

We aimed to perform classification tasks on the CIFAR-10 dataset, by implementing a multi-layer neural network with hidden layer, without using any high-level machine learning package. We trained the model using backpropagation, and combining different methods like cross-validation, stochastic gradient descent and momentum, early stopping, as well as L1/L2 regularization, we were able to achieve a 47.2% accuracy on the CIFAR-10 classification task.

Data Loading

The CIFAR-10 is a labeled tiny image dataset. Each image falls into one of ten distinct classes, and there are 50000 images in total. We split it into 80% for training, and the rest 20% for validation. Each image contains 3 channels for RGB color, and each channel is 32*32. We normalize the data by z-scoring it on a per channel per image basis. Specifically, for every value in one channel of an image, we subtract the mean and divide it by the standard deviation. After the normalization, the mean and standard deviation for any one train image would be 0 and 1.

Training

In each epoch, we train the model on a mini-batch by doing forward and backward propagation. In forward propagation, we calculate the weighted sum input from the previous layer, go through the activation function to have the output for the next layer. In back propagation, we get the delta

from the next layer, calculate the delta weight that is being used to update the weights associated with the current layer, and then update the weights.

When calculating the delta weight, we include a momentum term weighted by 0.9. Ideally momentum makes the gradient keep going in the previous direction, damps oscillations, and builds up speed in directions with a gentle but consistent gradient.

During training, we implemented early stop, which is when the validation loss increases for five consecutive epochs, we just save the weights with the minimum loss for later test.

Experiments

While calculating the delta weight, we apply regularization. L1 regularization makes the weight smaller at a constant rate, and L2 regularization make the weight smaller in proportion to its size. As a result, L1 penalizes small weights more, while L2 penalizes large weights more. Comparing the two regularization methods, L1 regularization performs better in our case. One possible explanation for this is that perhaps the dataset is not appropriate for regularization to make a huge difference, and since L1 is more aggressive at eliminating small weights, it did has a larger impact on the final test accuracy.

We also tried different activation functions. Among the two activation functions we've implemented, the tanh function yields a 42.9% accuracy, and the ReLU function yields a higher 47.2% accuracy, which agrees with our intuition, that ReLU is probably one of the best

activation functions out there. It has no vanishing gradient problem, its derivative is faster to compute and it does not activate all neurons at the same time.

Results

The optimal hyperparameters that yielded the highest validation accuracy are:

Learning rate: 0.004

Batch size: 128

Momentum gamma: 0.9

Regularization type: L1

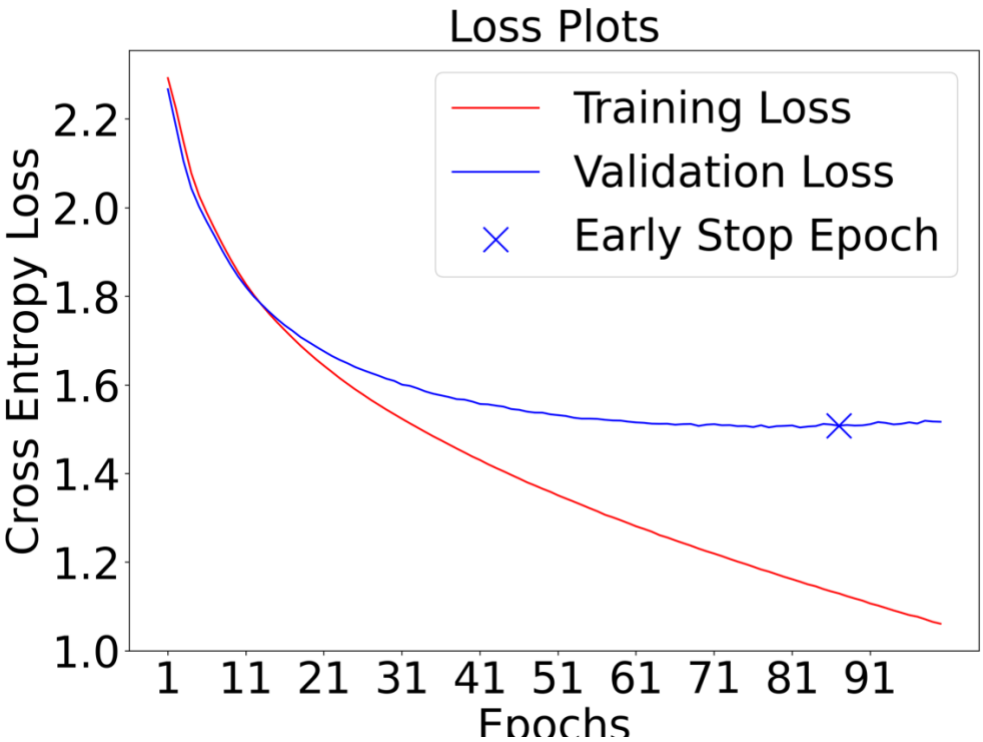
Regularization penalty: 0.0001

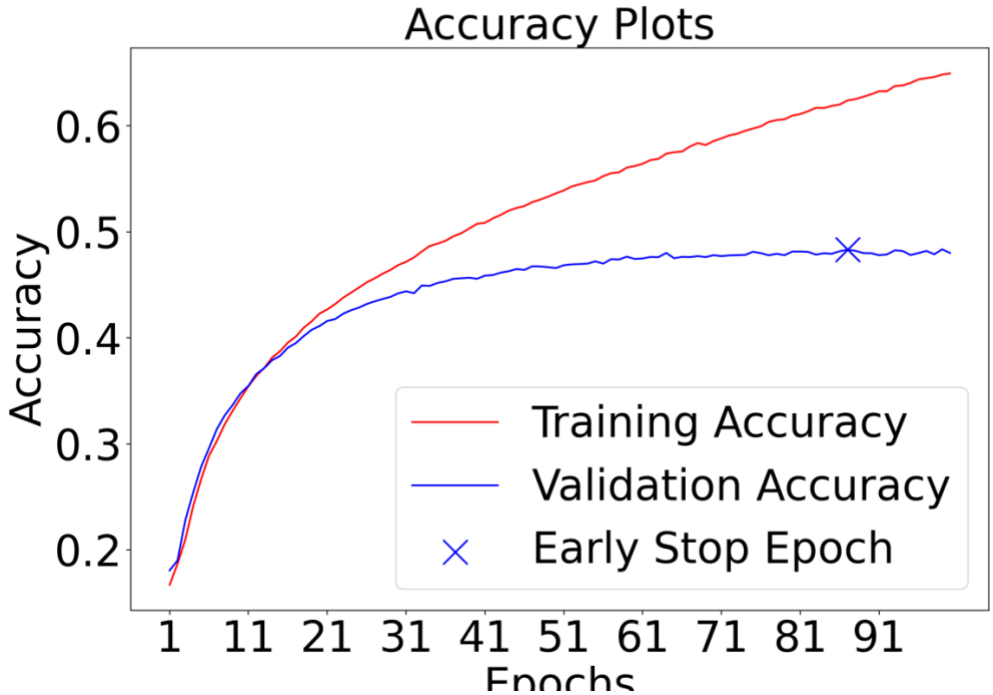
Epochs: 100

Activation function: ReLU

Final test accuracy: 47.2%

Below are the training/validation loss and accuracy plot:


Loss Plots
Training Loss
Validation Loss
Early Stop Epoch
Cross Entropy Loss
2.2
2.0
1.8
1.6
1.4
1.2
1.0
1
11
21
31
41
51
61
71
81
91
Epochs


Accuracy Plots
Training Accuracy
Validation Accuracy
Early Stop Epoch
Accuracy
0.6
0.5
0.4
0.3
0.2
1
11
21
31
41
51
61
71
81
91
Epochs

Contribution

This is a solo project that solely based on my understanding and experience with machine learning. No high-level machine learning package was used.