Classify Images by Multi-layer Neural Networks

Zongze Liu

Abstract

We trained a neural network with one hidden layer to classify images with 20 classes. The model can have a better performance if we introduced in momentum and regularization. $ReLU$ is the best candidate for the activation for hidden layers. Number of units in the hidden layers is also a factor in performance: we enhanced it from 128 to 256, the accuracy of test sets increased. So the best model we have trained has a test accuracy 30.05%.

Introduction

When training a neural network with multi-layers, we need to minimize loss function with respect to weights. By chain rule, the gradients in some layer depend on the gradients in the next layer. So gradients pass from the output layer back to the input layer. The efficient and feasible algorithm for computing the gradients is called the backpropagation.

In this homeowrk assignment, we trained a neural network with one hidden layer to classify images with 20 classes. We implement some techniques learned from class to improve our model. The first technique is momentum. It can keep us moving quickly in directions with small but consistent gradients, and moving slowly in directions with big but inconsistent gradients. The second technique is called regularization: minimize the size of weights and the loss function at the meantime. It can reduce the complexity of our model and make it more general for more data. We also tried different functions for the activation in the hidden layer. It turns out that $ReLU$ is the best. In addition to it, we played around the architectures of the neural network and found that increasing units can make model better, but increasing the number of layers doesn't necessarily do it. Finally, we test on all 100 classes. The model does not behave as good as its on 20 classes.

The combination of parameters of the model with the best performance is momentum-gamma $= 0.9$, L2-penalty $= 0.01$, the activation function $= ReLU$. Test accuracy is around 30.05%.

1 Related Work

We apply the perceptron algorithm with hidden layers. The algorithm was studied in class, so the report cites Professor's lecture notes on "Backprop: Representations". The formulas for delta rules, or say the derivation of backprop, were carried from the lecture note.

To deal with overfitting and improve the generalization, the loss function is chosen as the entropy together with the model complexity. We used L2 regularization: minimize the L2-norm of weights. The general idea is by Ockham's Razor. The idea and the method are stated in Professor's lecture note "A Few Notes on Improving Generalization".

To speed up mini-batch learning, we use momentum in gradient descent. The idea is from Nesterov in 1983. We cite the stochastic gradient descent with momentum algorithm in the discussion note by Chaitanya Animesh.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

2 Data Loading

There are 10000 training images loaded from CIFAR-100 dataset. They have both 20 target coarse labels and 100 target fine labels. The fine labels are only used in the last experiment, so we basically only use the coarse labels.

For the label sets, we implement one-hot-encoding over them in convenience of computing entropy.

For the training set, we split them into two sets: training set and validation set. The splitting point is $4:1$, i.e 8000 images for training and 2000 images for validation. For each image, it has 3072 pixels made up with 3 channels. More explicitly, such image have three channels and each channel has 32×32 pixels. We implement z-normalization on a per channel per image basis. We randomly pick one image. The means on three channels are $151, 149, 115$. The standard deviations on three channels are $64, 72, 76$.

3 3b - Numerical Approximation of Gradients

We performed this experiment separately on an output bias weight, a hidden bias weight, two hidden-to-output weights and two input to hidden weights. First we make $\epsilon = 10^{-2}$. Then we change the chosen weight w to $w + \epsilon$ and calculate the loss function $E(w + \epsilon)$. We do the same with $w - \epsilon$ and calculate $E(w - \epsilon)$. Next we computed the numerical gradient using the formula $\frac{d}{dw}E(w) \approx \frac{E(w+\epsilon)-E(w-\epsilon)}{2\epsilon}$. Then we calculate the true gradient of the original weight with the back propagation function in the neural network model. Finally we computed the absolute difference between the numerical and true gradients. The results are summarized in the table below and one can see that in all cases the absolute difference is within $O(\epsilon^2) = O(10^{-4})$.

```
weight type                numerical gradient           true gradient         absolute difference
('output layer bias weight', -7.473294338029568e-05, 5.1535591235313295e-06, 7.9886502503827e-05)

('hiden layer bias weight', -5.879822996135431e-07, 5.1535591235313295e-06, 5.741541423144872e-06)

('hidden to output weight #1', -2.065973648862851e-05, 9.803444316535384e-05, 0.00011869417965398235)

('hidden to output weight #2', -1.4476268714016528e-05, 4.889707240854833e-05, 6.337334112256485e-05)

('input to hidden weight #1', -4.136323141423759e-07, 4.889707240854833e-05, 4.9310704722690706e-05)

('input to hidden weight #2', -6.562391244635535e-07, 9.803444316535384e-05, 9.86906822898174e-05)
```

Figure 1: Numerical vs True Gradient Table

4 3c - Neural Network with Momentum

In this section, we use mini-batch stochastic gradient descent to avoid excessive memory usage when training, and use vectorized update rule obtained from individual assignment instead of "for" loops to improve efficiency. We turn off the regularization but turn on the momentum in our update rule. Set the epochs to be at most 100, however, the number of training steps is not even close to the upper bound because of the overfitting and early stop. The validation set is used to test whether overfitting happens. We set patience number of epochs to be 5, i.e the loss on validation set can increase at most 5 times. If it reaches 5, the algorithm stops.

The training procedure is that first, preprocessing data with z-normalization and splitting them into training set and validation set. Then, initialize model with default parameters in config document. The initial weight is random and initial velocity is 0. Train the model by forward and backpropagation. The training halts when it touches upper bound of number of epochs or early stops because of overfitting. Finally, check the trained model on the test sets and plot training and

validation accuracy and loss respectively.

The accuracy on the test dataset is 25.56% with default config. We implement hyperparameter tuning, and run experiments with different momentum-gamma. We try $\gamma = 0.7, 0.8, 0.9, 1.0$ and the corresponding accuracy on the test sets are 25.4%, 25.24%, 25.56%, 12.92%. The performance drops dramatically when $\gamma = 1$ and we cannot see performance gap between $\gamma = 0.7, 0.8, 0.9$ since the difference of accuracy is within 0.5%. However, we notice the training speed drops a lot with small γ. More explicitly, when $\gamma = 0.9$, the number of training epochs is 13 and when $\gamma = 0.7$, the number of training epochs is more than 20. With the same performance, we tend to choose the parameter with higher speed. So we set momentum-gamma to be 0.9. In another word, we don't change default config.

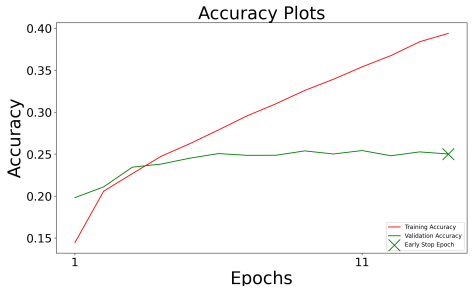

Figure 2: 3c accuracy

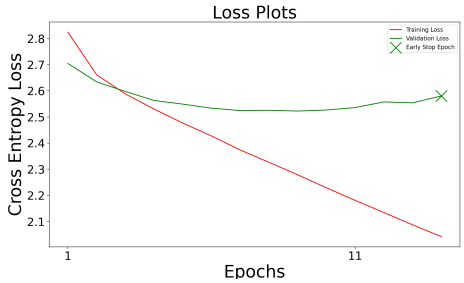

Figure 3: 3c loss

We observe huge overfitting from our plot, so overfitting keeps happening in our training. The reason I guess is that we are using too big learning rate. The final accuracy on the validation set is about 25% which is almost same as its on test set.

5 3d - Regularization Experiments

Consider optimization over regularization, we add weight decay to the update rule. We experiment with L2 regularization using value of 0.01, 0.001, 0.0001. The corresponding accuracy on validation sets are 25.7%, 25.28%, 25.57%. So $\lambda = 0.01$ outperforms the other two values and we set it to be 0.01 from this section on.

The data processing and training procedure is almost same as its in the section 3c. The only difference is that we turn on L2 regularization by setting L2-penalty equals to 0.01 in config document. By reading the plot, the accuracy on validation is around 25.7%. The test accuracy is 25.69%, which is pretty close to accuracy on validation. Since the training stops in 23 epochs, it doesn't matter if we enlarge the number of epochs to 110.

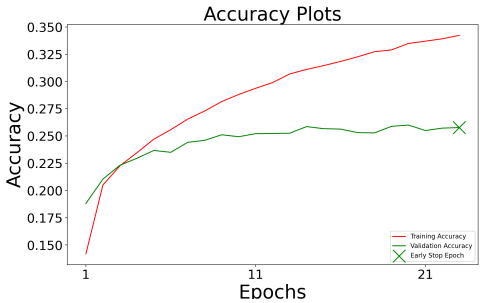

Figure 4: 3d accuracy

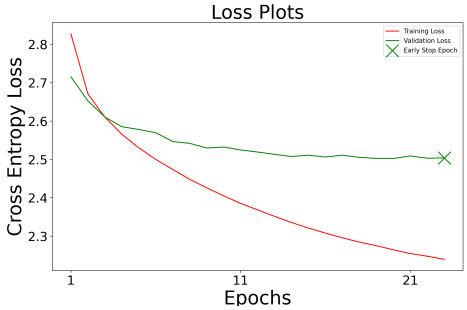

Figure 5: 3d loss

6 3e - Activation Experiments

In the previous two sections, we do hyperparameter tuning on momentum and L2 regularization, and find the best combination is momentum-lambda $= 0.9$ and L2-penalty $= 0.01$, which are the numbers we are going to use in the model of this section. We try using activation functions $sigmoid$ and $ReLU$ for the hidden layers, and compare their performance with the performance of activation function $tanh$ we got in the section 3d.

When using sigmoid function as activation on hidden layers, the accuracy on validation and test sets are 22.3%, 21.69% respectively. From section 3d, when using $tanh$, the accuracy on validation and test sets are 25.7%, 25.69% respectively. When using $ReLU$, the accuracy on validation and test sets are 29.1%, 29.1% respectively. The data tells us $ReLU$ is best for hidden layers and $sigmoid$ is the worst. But there is one interesting observation: the overfitting condition is much better when using sigmoid function. The loss curves of training and validation are closer compared with using other functions for activation.

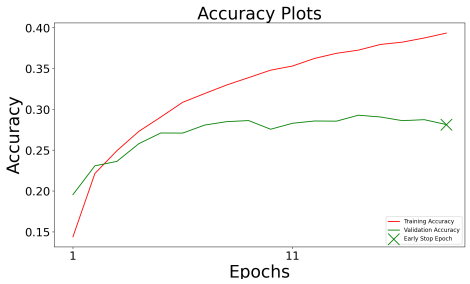

Figure 6: 3e accuracy: ReLU

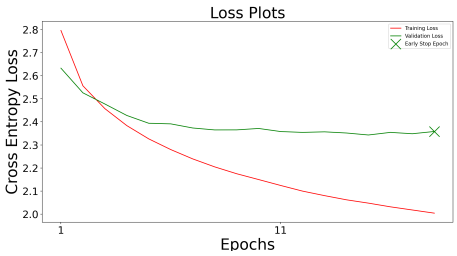

Figure 7: 3e loss: ReLU

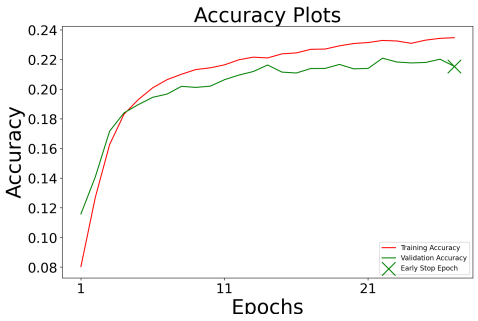

Figure 8: 3e accuracy: sigmoid

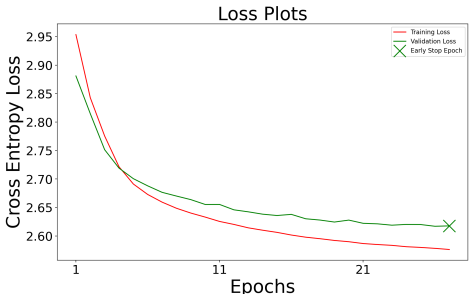

Figure 9: 3e loss: sigmoid

7 3f - Network Topology Experiments

First we experimented with number of hidden units in the single hidden layer network. We work with the best model in 3e with an accuracy of 29.1% on the test set (128 hidden units). We halved the number of hidden units to 64 and observed that the accuracy on the test set dropped to 27%. Then we doubled the number of hidden units to 256 and observed that the accuracy on the test set is 30.05%. Note that the number of hidden units is the only thing we modified. We may conclude that the increase in the number of hidden units is positively correlated with the model's accuracy on the test set and there may even be a linear relationship between the number of hidden units and the performance of the model.

Next we experimented with the number of hidden layers. We created a network with two hidden layers, each of which has 128 hidden units, based on the optimal model from 3e. The final test accuracy is 28.5%, which is worse than the performance of the model from 3e(29.1%). This

shows that the inner structure of neural networks can be very delicate and that the performance will not improve linearly with the increase in the number parameters. The 'information' in backward propagation may have been deformed or depreciated in the 'longer chain of travelling' due to the presence of the extra hidden layer. Further increasing the number of hidden units in each hidden layer may lead to better performances.

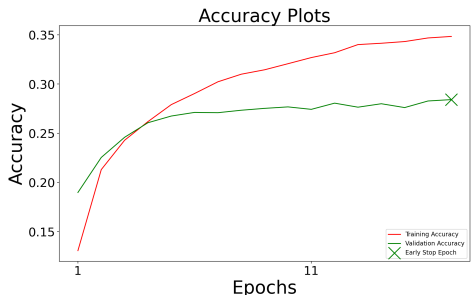

Figure 10: 3fi accuracy: 64 hidden units

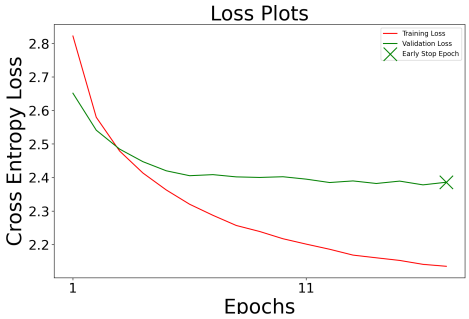

Figure 11: 3fi loss: 64 hidden units

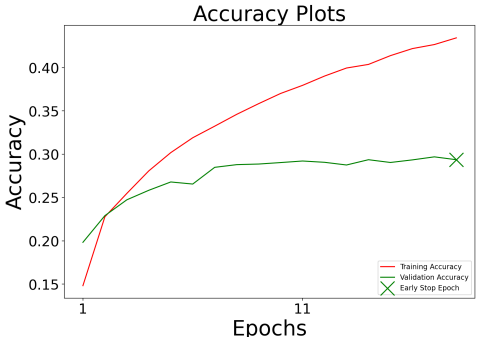

Figure 12: 3fi accuracy: 256 hidden units

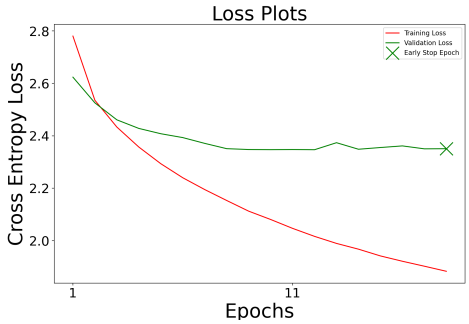

Figure 13: 3fi loss: 256 hidden units

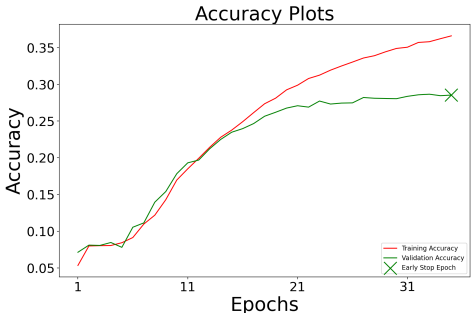

Figure 14: accuracy: 2 hidden layers with 128 hidden units per layer

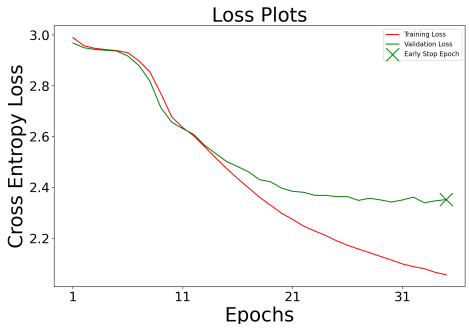

Figure 15: loss: 2 hidden layers with 128 units per layer

8 3g - Experiment with 100 classes

Base on the results from section 3e, the network config : momentum-gamma $= 0.9$, L2-penalty $= 0.01$, the activation function is $ReLU$. For the training procedure, we need to load fine target labels which have 100 classes. Others are the same.

From the plot, the accuracy on the validation is around 19%. Compared with its in section

3e, it drops 10%. It is not surprised because we are doing finer classification with more output units. So it requires better model. Something surprised me is that the test accuracy is only 1.5%. It indicates that the training overfits badly. To improve the performance, we can add the number of units in hidden layers to store and process more features of images. One way to overcome overfitting is to train on more data (8000 images compared with 100 classes may be too small). So import more images from the dataset.

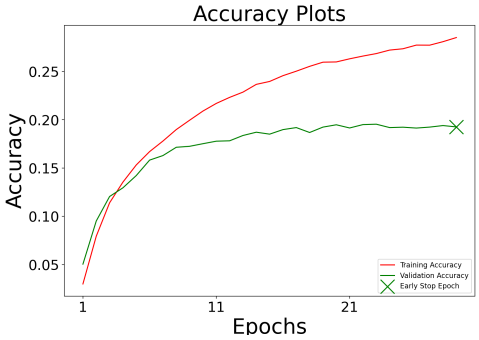

Figure 16: 3g accuracy

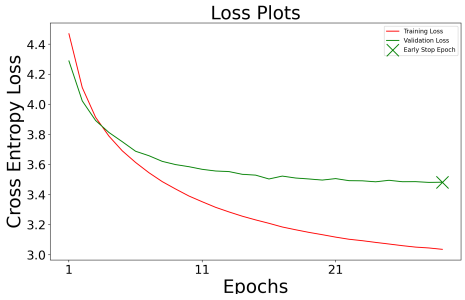

Figure 17: 3g loss