Report for 2-Layer Neural Net with Softmax Classifer (CIFAR-10 Dataset)

Overview

I have filled in the blanks in **neural_net.py** and **two_layer_net.ipynb** and submitted them. The goal was to classify the CIFAR-10 dataset using a 2-layer neural net. In **two_layer_net.ipynb**, I used a Jupyter notebook to tune hyperparameters and documented the process of obtaining good results. Additionally, the **neural_net.py** file called in **two_layer_net.ipynb** contains the implementation of the 2-layer neural net, with methods such as predict and loss defined.

Code Description

(I won't attach the code because I need to put code description, results, and discoveries in only a page.)

Methods implemented in **natural_net.py** are loss, train, and predict. In the loss method, there are parts that make a score to enter softmax, makes probability using softmax with a score, obtains loss (data loss + L2 regularization), and a backpropagation for weight and bias. In the training method, the parts that obtains mini batch for learning, and makes gradient design with gradient through backpropagation and learning rate(hyperparameter). In the predict method, the index with the highest score was returned after calculating the score through the forward pass.

In the **two_layer_net.ipynb** notebook, I observed the results of using not good hyperparameters and then proceeded to perform hyperparameter tuning. I automated the process of training and comparing models with different combinations of hidden_layer_size, learning_rate, and regularization_strength values, all while evaluating the validation accuracy. As a result of this tuning, I was able to obtain a model with approximately 40% accuracy. I will discuss how I believe the parameters influenced the model's performance below.

Results and Discussions

(For detailed experiment such as result data, pre-plan and prediction, please refer to the **two_layer_net.ipynb** notebook.)

As a result, the model achieved a significantly high accuracy rate(42%), indicating the success of hyperparameter tuning. Upon analyzing the results, I was quite surprised to find that a significant factor contributing to this improvement was the increase in the learning rate. Initially, I had thought that the model's simplicity was the main issue, preventing it from representing complex patterns effectively. In fact, when examining the data from the parameter tuning experiments, I noticed that even with hidden layer size of 50, increasing the learning rate led to a substantial improvement(40%).

Additionally, it turned out that hidden layer sizes of 100, 300, and 500 did not lead to significant improvements in accuracy. I initially considered a hidden layer size of 50 to be quite small, but it possess good expressive power. Furthermore, I couldn't observe significant changes in performance with respect to epoch or regularization strength.

Through this project, I've learned that even a single bad hyperparameter value can have a significant impact on a model's performance. And, I learned by a lot of trial and error(although the report and Jupyter notebook only contain clear results and processes) about how other parameters can influence the outcome.