Name: Badr AlKhamissi

ID: 900141572

CSCE 4930

Practical Machine Deep Learning

Assignment #2

CIFAR-10 Classification using Multi-layer fully connected Neural Networks

Description:

For this assignment, I was able to achieve 60.18% accuracy on the CIFAR10 dataset using my own implementation of a neural network. I used PCA to reduce the data to 200 principal components and data augmentation to increase the data size. Full details of my implementation is written below.

Tuesday, March 21, 2017

Implementation: My Neural Network Library

I implemented my own modular neural network library using Python. It consist of the following files:

- `NeuralNetwork.py`: the actual code that implements the training, i.e. the feed forward and backward propagation
- `Layer.py`: it's a class that stores the data for each layer including the weights, biases, gradients, dropout masks and activation function for the layer. It also the performs the weight update based on the method selected by the user.
- `Activation.py`: includes multiple activation functions and their gradients
- `Preprocessing.py`: a file that includes classes for preprocessing
  - `Normalize`: for image normalization
  - `PCA`: to reduce the dimensionality of the image
  - `ImageProcessing`: includes functions for data augmentation such as mirroring, rotation, flipping vertically and shifting.

There are two additional files for this assignment:

- `cifar10_test.py`: this is the program that uses the library to classify the images
- `data_utils.py`: this is a helper file to load the CIFAR10 data into matrices

Data Preprocessing

For the preprocessing I first rescaled the images by dividing them by 255, then I normalized both the training and testing data set by centering the images; subtracting the mean image and dividing by the standard deviation both of which are of the training data set.

$$x_i := \frac{x_i - \mu_i}{s_i}$$

I then used PCA which is a dimensionality reduction tool to remove noise from the images, I reduced each image to 200 components, using this technique I was able to train the model much faster.

Network Architecture

I tried several network architecture during the past 2 weeks. Starting by small networks so I could test that my implementation is working correctly, afterwards I fixed a 1 hidden layer architecture with 500 neurons to find a lower and an upper bound for the learning rate. I found that my lower bound that makes the loss goes down is ~1e-6 and the upper bound that makes the loss overshoot is ~1e-1. The optimal alpha after many iterations of testing was 1e-2. I did the same process for the regularization parameter, and found that the lambda that got me good results was also 1e-2. I fixed both parameters so I could try different, larger, network architectures.

I gradually increased the network to be able to find the best solution, by trying different number of neurons in each layer and different number of layers in the network. I then used Keras, which is an open source neural network library, to be able to test more architectures faster with parameters similar to my network, but this technique didn't prove to be very useful.

The first architecture that showed promising results was a 2 hidden layer architecture with 700 and 500 neurons respectively. It got me around 54% accuracy, then a three hidden layer architecture of 1000, 750, 500 neurons respectively was able to get me 55%. Then I tired to decrease the depth and increase the number of neurons, so I tested with a 2 hidden layer architecture with 3000, 2000 neurons respectively which got me ~57% accuracy. For those architectures I was using normal stochastic gradient descent as the update method, L2 regularization, and normalizing the images as discussed before. The activation functions for all were relu in the hidden layers and sigmoid in the outer layer.

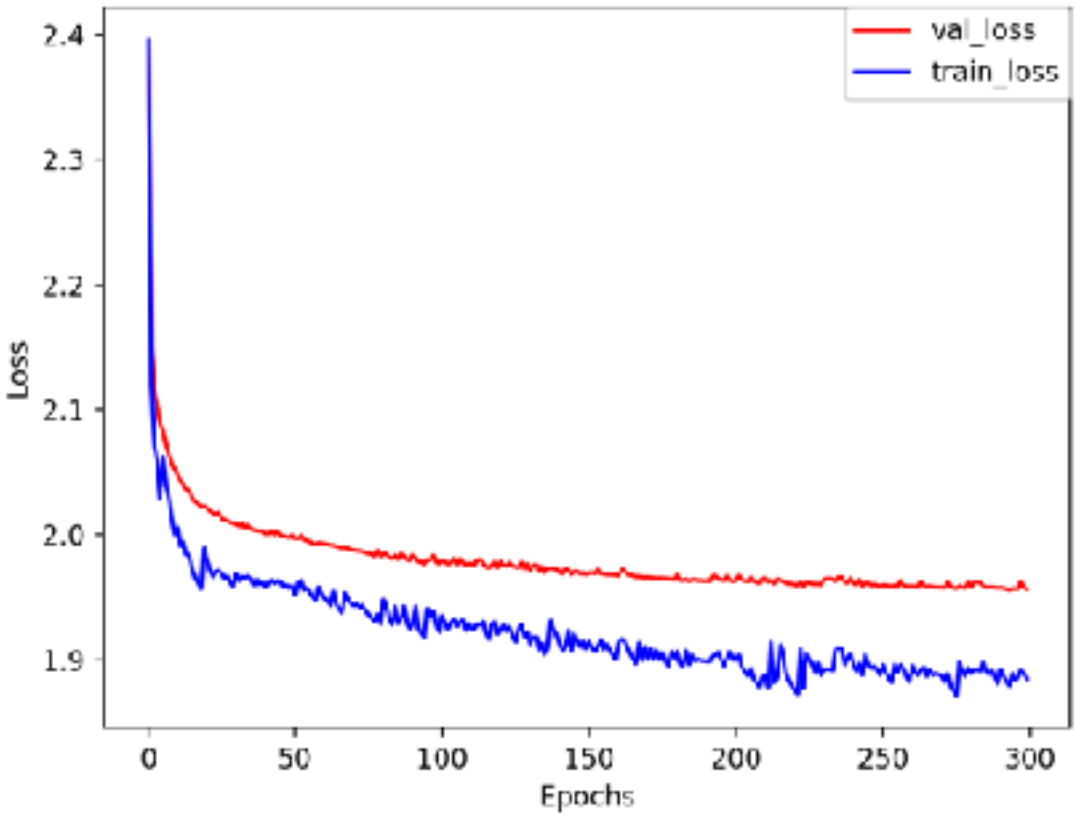
I then figured out that I should add more features to my library to be able to get a higher accuracy. So I added momentum and adam update methods, after experimenting with them, the results of both didn't look very promising to me, I then added dropout to my network, it was able to reduce the overfitting of the network, but didn't pass the 57% accuracy I reached before. I then introduced PCA and data augmentation, both of which were able to increase the accuracy to 58%. I then experimented with different data augmentation methods including shifting, flipping horizontally and vertically, rotating and zooming, then I figured out that shifting and mirroring got me the best results. I reduced the dimensionality to 200 after this paper: *CIFAR-10: KNN-based Ensemble of Classifiers, https://arxiv.org/pdf/1611.04905.pdf*

Final Architecture

The procedure and network architecture that got me 60.18% testing accuracy and a highest validation accuracy of 61.86% was as following:

- **Preprocessing**: Normalizing the images as mentioned before
- **Preprocessing**: Reducing the images using PCA to 200 components
- **Architecture**: Using a two hidden layer architecture with 750, 500 neurons respectively
- **Hyper-parameter**: Learning rate of 1e-2
- **Hyper-parameter**: L2 regularization with lambda of 1e-2
- **Cost Function**: Negative log likelihood
- **Update method**: normal mini-batch stochastic gradient descent
- **Weight Initialization**: Xavier method
- **Activation Functions**: ReLU activation function in the hidden layer, and sigmoid in the outer layer
- **Dropouts**: none
- **Batch Size**: is 120
- **Epoch Number**: I got this number of 300 epochs
- **Data Augmentation**: Shifting and Mirroring
- **Validation Split**: 10%

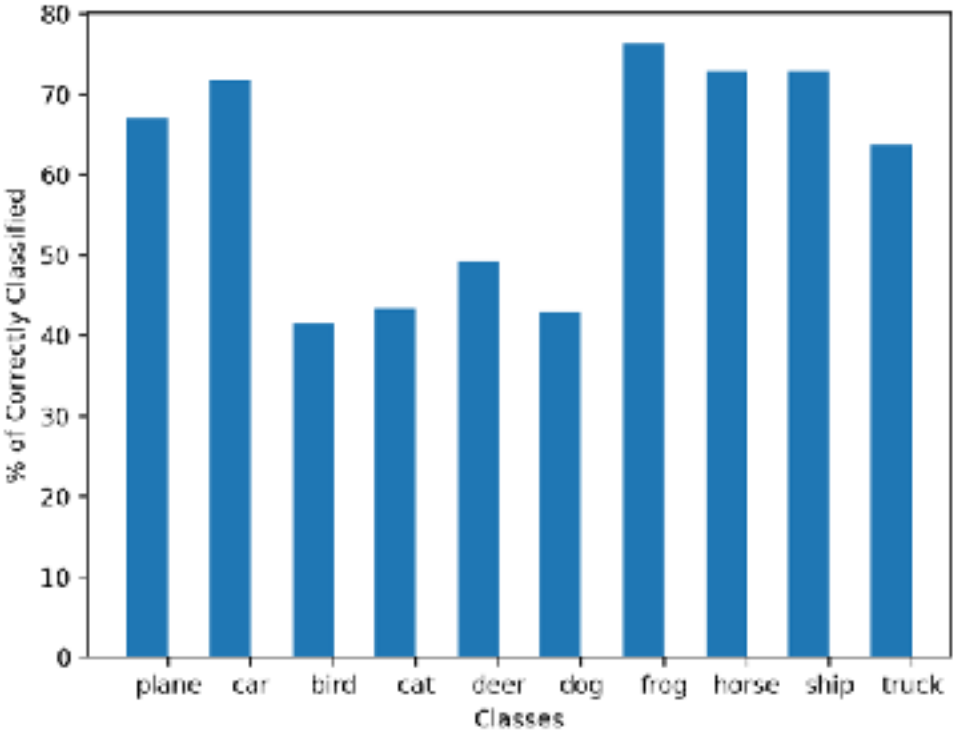
Validation and Training Loss versus Number of Epochs



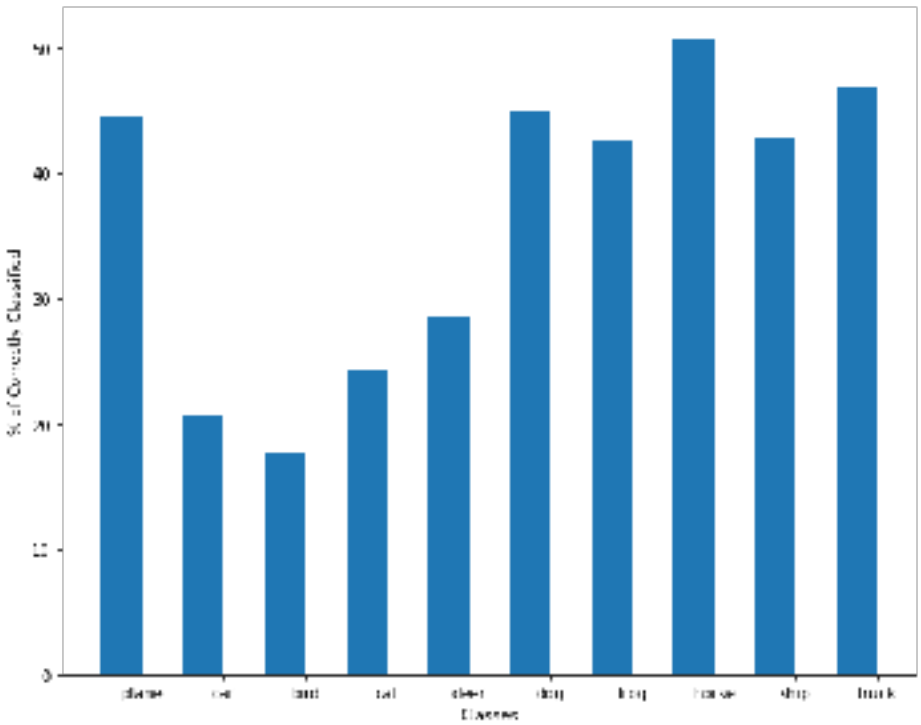
I stopped training when the validation loss was beginning to saturate, I attempted with the same architecture with larger number of epochs 500 and 666 but it got me the same result. I should have probably tried to stop, record the weights and decrease the learning rate then continue from there to be able to search for directions that require smaller steps to reach the local minimum

Correct Classification Rate

CCRN OF CIFAR10 USING NEURAL NETWORK CLASSIFER



CCRN OF CIFAR10 USING LLS CLASSIFER



From the previous page we can see that the correct classification rate of the neural network classifier is considerably higher than the linear least square classifier. Where the highest classification rate for one class in LLS is in the 50s whereas in the Neural Network is in the 70s.

Average Correct Classification Rate

I was able to achieve using the final architecture described a testing accuracy of 60.18%, and a highest validation accuracy of 61.86%.

I then attempted to use keras to try to get a considerable higher accuracy but didn't have enough time to fine-tune the parameters. However you will also find my keras code attached along with the source code of my implementation.