ECE532 Project Report

---- Peter Li

Introduction

CIFAR10 is a well-used dataset for neural network researchers. The dataset consists of 60000 32-by-32 color images in 10 classes, with 6000 images per class. The labels of classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. A snapshot of the dataset is shown below.

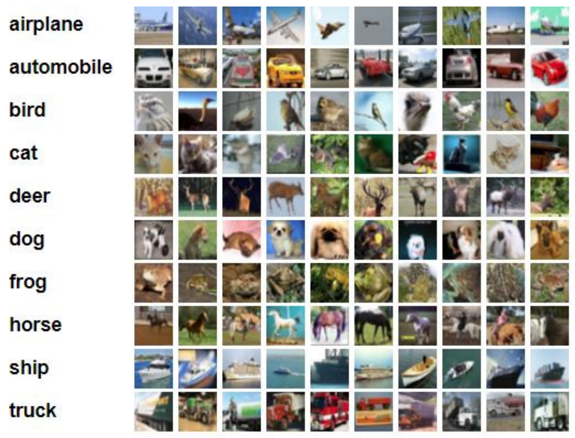

Figure 1. Sample images from CIFAR10 dataset.

A couple of methods covered in class were used to create linear classifier model for CIFAR10 dataset. The methods applied include: the least square (LS), principal component analysis (PCA), support vector machine (SVM), and neural network. Due to the limited depth of each method covered in class and the complex nature of the dataset, performance of each method is not as good as expected.

Details of the dataset

The ratio between number of training and test images is 5-to-1, meaning that there are 50000 images for training and 10000 images for testing. The color information of each image is put in an array of size 3072, with the first 1024 elements representing the red channel, the next 1024 being green, and the final 1024 being blue. The label of each image is an integer ranging from 0-9 corresponding to 10 classes listed earlier. Images in CIFAR10 represent a large variability in size, position, pose, and illumination. So it’s difficult to capture certain features for images in the same class

Algorithms & Results Walkthrough

The training dataset X_train has the size of 50000-by-3072 and the label dataset y_train has the size of 5000-by-1. If we assign weight to each color value. Then we can write

$$y = Xw \tag{1}$$

The Least Square Method

The least square method has the form of:

$$\min_{\mathrm{w}} \|\boldsymbol{y_{train}} - \boldsymbol{X_{train}} \boldsymbol{w}\|_2^2 \quad (2)$$

solution of weight **w** being

$$\boldsymbol{w} = (\boldsymbol{X}_{\text{train}}{}^{\boldsymbol{T}} \boldsymbol{X}_{\text{train}})^{-1} \boldsymbol{X}_{\text{train}}{}^{\boldsymbol{T}} \boldsymbol{y}_{\text{train}}.$$

Plug the calculated weight into Eqn.2 we got a success rate of 11.84%. Plug the calculated weight, test set, and test label into Eqn.2, we got a success rate of 10.88%. Considering there are 10 classes in the dataset, the performance of the least square method on both training and testing set is no better than a random guess (even a random guess has a 10% chance being right).

The reason of such poor performance is partially due to the complexity of the dataset. Another reason might the noise in the background of the picture prohibiting the weight being assigned to the right color channel. This leads to the idea of using PCA to get rid of the noise.

Principal Component Analysis in combination with Least Square method

The principal component analysis is based on the singular value decomposition of a matrix.

$$S = V\Lambda V^T = \sum_{i=1}^{r} \lambda_i v_i v_i^T \quad (3)$$

The method is achieved thru Python’s built-in package PCA from sklearn.decomposition. It is found that the first 500 components by PCA method have an accuracy over 98% to represent the whole training matrix, as shown in Fig.2

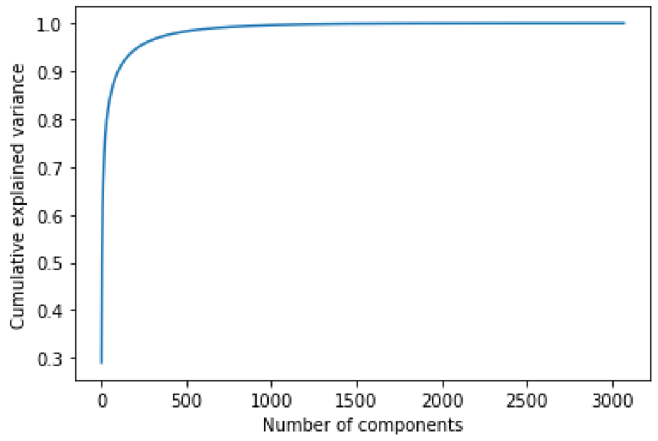

Figure 2. The cumulative variance of PCA vs number of components.

Then the training data’s size changes from 50000-by-3072 to 500-by-3072. Plugging the new training dataset into Eqn.2 we got the same success rate of 11.84% for training data and 7.49% for the test data set, which is worse than original 10.88%.

So PCA did not help improve the performance of LS method. We could probably try some Ridge regression but the selection of regularizer parameter seems arbitrary and time consuming. So we will move onto SVM method next.

Support Vector Machine

The SVM method is achieved thru Python's built-in package svm from sklearn. More information can be found here. Both linear kernel and polynomial kernel are applied. The results are shown below.

	Linear Kernel	Polynomial Kernel
Success rate on training set	100 %	98.7%
Success rate on testing set	29.72 %	32.62%

It is found that the success rates of SVM method of both training and testing dataset are much higher than those by LS method. The reason could be that the SVM method allows more flexible boundaries to maximize margins between data points. One this to notice is that only the first 1000 training samples and their labels are fed into svm function to save computing time.

Neural Network

The neural network is implemented thru Multi-layer Perceptron classifier (MLP) from Python's build-in sklearn.neural_network package. This method was the most complicated one since there are so many different parameters to play with. More information can be found here. The MLP classifier optimize the log-loss function using stochastic gradient descent (SGD) or LBFGS.

For a MLP classifier using SDG method, 5 hidden layers, 1000 max iterations, and 10 verbose, the relationship between loss and the number of iterations is shown below. The success rate of such classifier on the testing dataset is 10.0%.

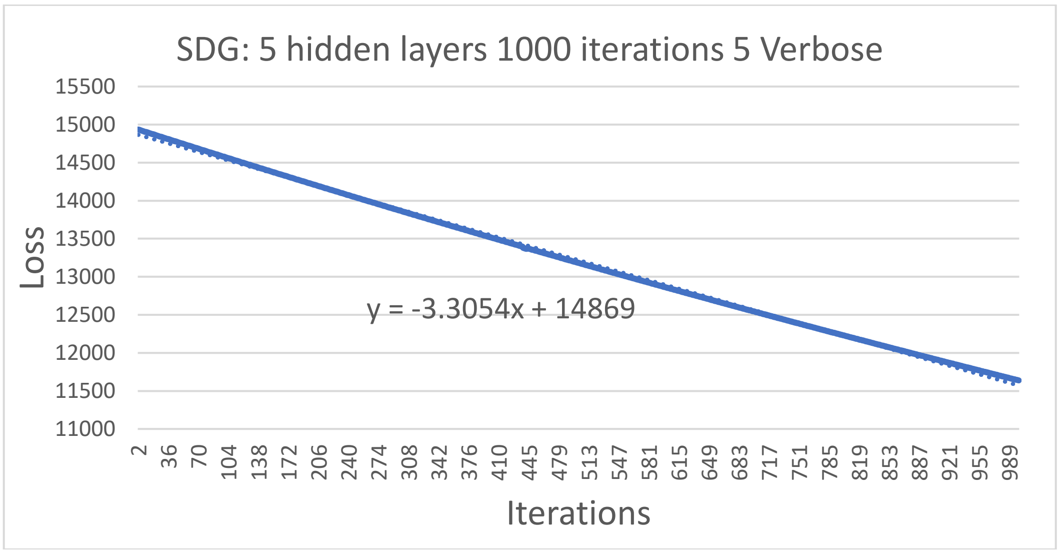

Figure 4. Loss vs. # of iterations for MLP classifier with SDG, 5 hidden layers, 1000 iterations and 5 verbose

Results above takes about 15 min to produce. From figure 4, it is found that loss for such method is very high but the converging rate of loss is very slow. Another run with 3 hidden layers is shown on the next page. The method yields a success rate of 10.02%.

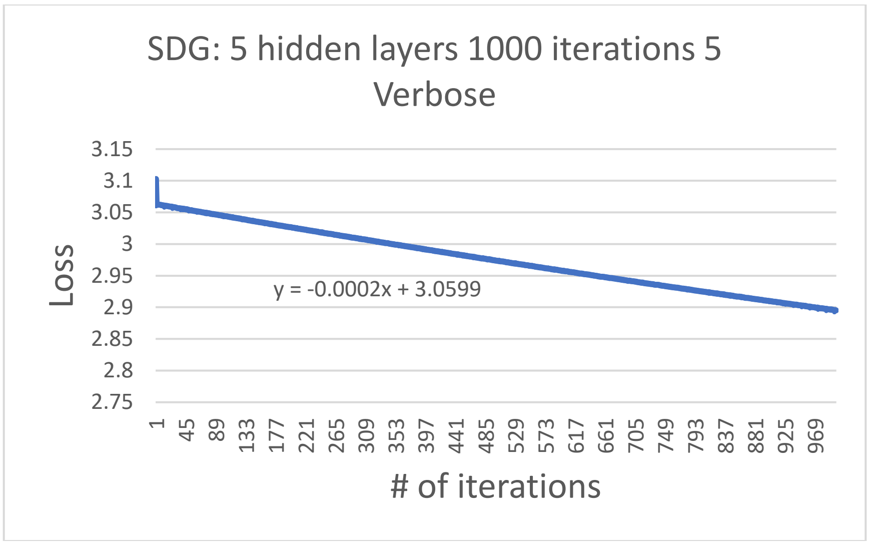

Figure 5. Loss vs. # of iterations for MLP classifier with SDG, 3 hidden layers, 1000 iterations and 5 verbose

There are more parametric runs like this with hidden layers being 3, 5, and 10, maximum iterations being 1000, and verbose being 5 and 10. In general, all methods using SDG take long time to converge losses, which makes sense because of the stochastic nature. Their success rates were only about 10%.

If we use LBFGS method instead of SDG for MLP classifier, the computing time reduces significantly, but the success rate was kept around 10%.

Discussion

Based on some literature, the accuracy of some convoluted neural networks can be up to 80%. It is expected that the neural network would have the best performance overall, but it was comparable to the performance of PCA+LS. SVM gives the best results.

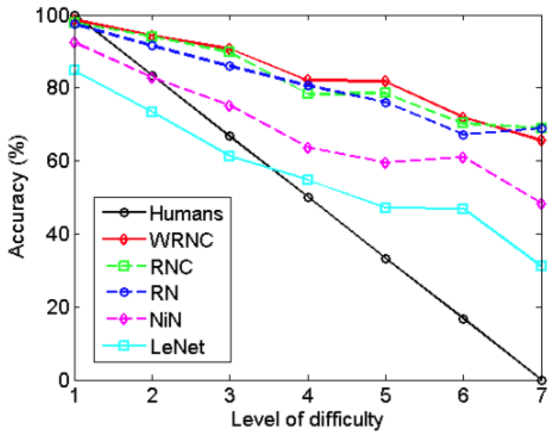

Figure 6. Performance of some neural networks on CIFAR10. ref

One reason for poor performance of my neural network is the lack of familiarity of python's build-in package. Some other studies applying MLP classifier using keras achieved over 90% accuracy. Some tricks such as cross validation might help improve the performance of all three methods, but since the samples are already in random order, I don't see it will help much.

Conclusion

The fact that SVM performs the best overall (29.7% for linear kernel and 32.6% for polynomial kernel) proves that it works better on complex boundaries. My neural network performs much worse than other studies using similar classifiers. More work needs to be devoted to improve the performance of the code.

Dataset:

CIFAR-10 and CIFAR-100 datasets (toronto.edu)

GitHub Page:

GitHub - peterlipeixuan/machine_learning_final_project: UW-Madison ECE532 final project