Various optimization methods of the performance in multiclass learning problem based on auto-encoder neural network

Author Name (Paper 215 assignment 2)
Research School of Computer Science, Australia National University

Abstract

Multiclass learning problems are everywhere in our real life and optimization about this kind of problems in different aspects were interested by computer scientists since a long time ago. The recent Auto-encoder technique using Pytorch in Python provides us a much faster and efficient way to implement multi-layer neural network to solve the multiclass learning problem.

This article focus on optimizing the performance of the letter recognition and classification problem using multiple neural network and Pytorch. The optimization methods include processing the original input data, adjust the parameters of the multiple neural network, choices of different optimizer and activation function, reduction techniques applied on the hidden units and the idea of batch normalisation. And after applying all the techniques above, the result shows a better or similar output as a published research paper.

Introduction

There are few reasons that I chose the letter recognition dataset. Firstly, it has a large amount of data which guarantee that we have enough data to process our training progress. Secondly, there are 16 features of the dataset, which allows us to design a relatively complicated neural network. Furthermore, it is really close to our real life and we have clear right or wrong classification about this dataset. With the letter dataset, I decided to perform the classification and divide them into 26 letters according to the alphabet. To accomplish this task, I firstly modify

the dataset and randomly divide them into a training set with 1,600 instances and a testing set with 400 examples. Then train the neural network which built by Pytorch using the training set and adjust the parameter to achieve best learning result. In this task, I choose the Neural Network with two hidden layers and 300 neurons for each hidden layers. From the input layer to the first hidden layer I use the rectifier activation function, from the second hidden layer to the output layer I chose the sigmoid activation function and for the rest connections simple linear functions are applied. To balance between the learning time and the precision of the outcome, the learning rate is set to 0.2 and the stochastic gradient descent optimizer with momentum set to 0.8 is used. And to avoid over-fitting problems, reduction techniques are applied to the hidden units and a unit or groups of units having too little effect to the outcome are removed. [1]

Method

In this task, several tests are implemented to compare the performance of two-layer neural network (one hidden layer) and three-layer neural network (two hidden layers). And different tests to find the relative ideal parameters for the neural network are also carried out. I chose the three-layer neural network with 300 hidden neurons in each hidden layer and the learning rate set to 0.2 in the end. As the momentum optimizer obviously improve the training efficiency compared to other methods, it is chosen in this task.

To avoid over-fitting and low efficiency of the neural network, some reduction techniques are applied. Using the dropout() function in Pytorch, we can simply and efficiently remove the undesirable units in each epoch with low probabilities having effect on the outcome. In this task, the units from first hidden layer with probability influenced the outcome lower than 0.4 and the units from second hidden layer with probability lower than 0.2 are removed. Batch normalization is implemented to accelerate deep network training. Confusion matrix is designed to test the performance of my neural network.

Accelerating with GPU using the cuda() function is considered, however, it

doesn’t have significant change in this particular example, so I didn't use it.

Results and Discussion

Using the neural network with the specific parameter and methods mentioned above, we have a pretty rapid increase in training accuracy and the figure below is one test result.

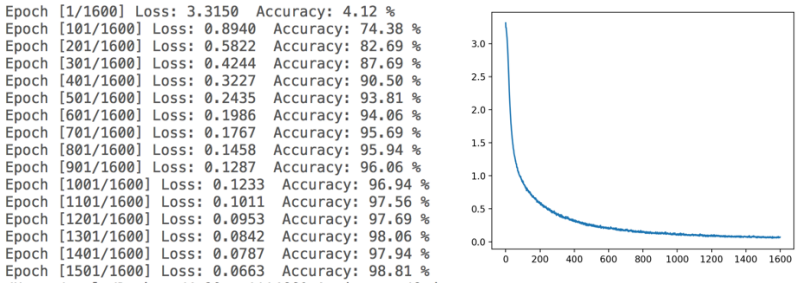

Figure1. result of one training (left) and the corresponding loss

And the test accuracy is always maintained at 82%~87%. For the specific test above, the test accuracy is 82.5%.

From solving multiclass learning problems via error-correcting output codes by Thomas G. and Ghulum B. [2], we can see the training outcome using the same dataset with this one below:

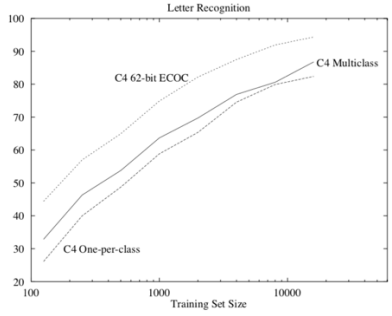

Figure2. Training outcome from a published paper using different methods

Conclusion and future work

By implementing this specific task, we can conclude that neural network can achieve pretty ideal learning outcome with some tasks and it seems that more hidden layers and more complicated structure of the neural network can help improve the ability of learning. To achieve the best learning result, we usually need to test many times to find the best matching parameters and functions for the neural network. To avoid slow learning period, I only use 1/10 data of the dataset and there are no data for evaluation. That's one aspect which I should improve. Apart from that, although the activation function I choose improves the result compared to my initial choice, it's still not perfect. The batch normalisation part is not being carried out smoothly too.

References

1. Gedeon, T. D., & Harris, D. (1991). Network reduction techniques. In Proceedings International Conference on Neural Networks Methodologies and Applications (Vol. 1, pp. 119-126).
2. Thomas, G., Ghulum, Bakiri.: Solving Multiclass Learning Problem using Error-Correcting Output Codes. Journal of Artificial Intelligence Research 2 (1995, pp. 263-286).