A Deep Learning Approach with an Ensemble-Based Neural Network Classifier for Black Box ICML 2013 Contest

Lukasz Romaszko LUKASZ.ROMASZKO@GMAIL.COM
University of Warsaw, Poland

Abstract

This paper presents a technical description of a solution for the International Conference on Machine Learning contest in Representation Learning: The Black Box Challenge. The organizers provided a dataset including one thousand labeled data samples and over 130 thousand of extra unlabeled samples. The task was to correctly classify examples into one of nine classes. The presented solution implements classification conducted with an ensemble of neural network classifiers with several improvements in the training method. In order to increase accuracy, classifier is given a reduced input vector preserving important correlations in the original input dataset. The approach is based on sparse filtering algorithm designed for deep learning problems. It was found that reducing dimensionality of samples by sparse filtering, even without the extra data, resulted in a significant improvement in accuracy.

1. Introduction

In this competition classifier had to be trained on a dataset that is not human readable, without knowledge of what the data consists of. Results were scored based on classification accuracy on a private test set. This challenge was designed to reduce the usefulness of having a human researcher working in the loop with the training algorithm.

2. Description of Data

Each sample consisted of 1875 numbers and one of nine classes should be assigned to each of them. The distribution of classes was diverse – frequencies ranging from 7% up to 20%. Only 1000 samples were labeled, and there were over 130 thousand of unlabeled samples. The test dataset consisted of ten thousand samples, split in a half into public and private parts.

3. Evaluation

The score was calculated as a percentage of correctly predicted classes. The maximum score could be 100%, while random choice of 1 out of 9 classes would give approximately 11%. The organizers provided a few baselines – logistic regression trained with stochastic gradient decent (REGR) which received result of 21.1% and multi layer perceptron – a feedforward neural network (MLP) with sigmoid units achieving score of 52.5%. This baseline was extraordinarily well-performing, because only one third of contestants outperformed this result. The scores of baselines, compared to our approaches and the winning solution are presented in the Figure 1.

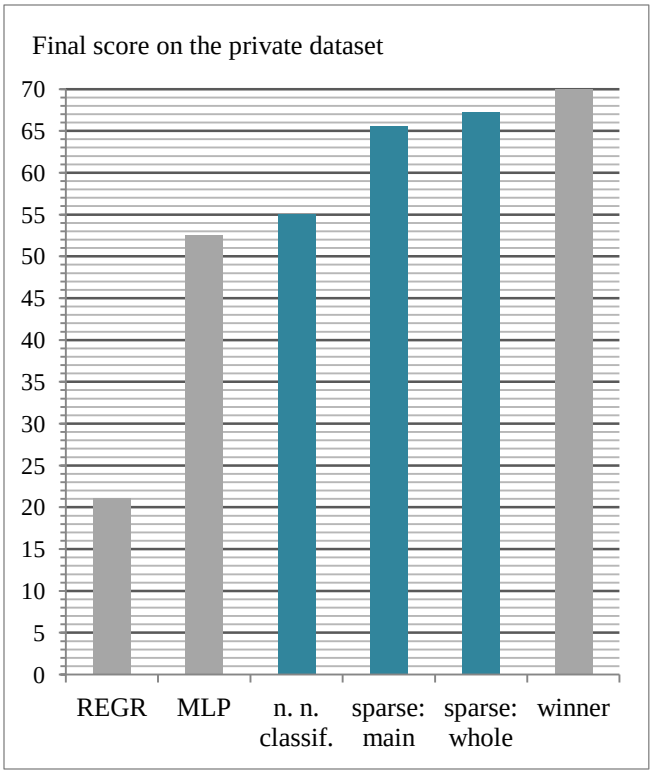

Figure 1. Algorithms performance

4. Neural Networks Classifier

4.1 Introduction

Since our goal was to develop a high quality classifier, we decided to employ algorithm based on neural networks (n. n. classifier in the Figure 1.). The first task was to choose the best available neural network implementation. Initially, we tested several R libraries of neural networks implementations, but the range of offered customizations was very limited. Moreover, the results of classification were poor. Finally, we have chosen a Fast Artificial Neural Network Library (Nissen, 2003). It is a free open source library, which implements multilayer artificial neural networks in C with support for both fully connected and sparsely connected networks. It includes a framework for easy handling of training data sets. It is versatile, customizable, well documented and fast.

4.2 Parameters and training method

In this section we present the parameters of the neural network which were used in each of the presented versions of our solution. Firstly, activation function set to all units was a sigmoid function. To calculate RMSE, a tanh error function is used, which punishes differences stronger than a standard linear function. Learning rate around 0.3 was the best in all tested configurations of input data/number of neurons. The most important feature of the presented solution is that training method is incremental – it is based on a standard back propagation algorithm, where the weights are updated after each training pattern. This means that the weights are updated many times during a single epoch. Initially cross validation was performed by learning on 90%, validating on 10% of the labeled dataset. A function which is called every epoch during training was overridden – it allowed to analyze learning progress and save only a network which outperformed the previous best. This kind of training was performed on each validation part. For instance, assuming the results on validation part were as follows, the network in [] was saved and only the last, written in bold, was finally produced:

[35], 34, 32, [45], 44, [56], [60], 58, **[66]**, 64, 66, 63

For a direct learning from 1875 input numbers (no features selection) the best results were obtained with the following parameterization of the network layers:

1875 (input) – 200 (hidden) – 200 (hidden) – 9 (output)

Each output neuron corresponds to a predictor of a particular class. A value of 1 referred to the correct class label, the rest eight were assigned a value of 0. The class was selected based on the maximum value of all nine output neurons predictions. The solution ran all trained neural networks and chose the most common class in a voting. The achieved result was around 55%, better when compared to 52.5% result of the neural network baseline.

4.3 Improvements

The accuracy of neural network depends on values of initial neuron weights used in training which are chosen randomly. Due to this randomization a network may sometimes converge worse than it could even with the same layers and fixed parameters configuration. In order to increase accuracy, our approach trained each network several times on each of all validation parts and saved only a network which performed the best on a validation part. The last improvement of the network classifier was change of learning type to 40-fold cross learning. This had two advantages: firstly allowed to conduct a voting with a higher number of classifiers, and secondly the training set was larger by about 8%. The improvements mentioned above resulted in an increase of performance of about 0.5 and 1.0 percentage point.

5. Sparse Filtering Approach

5.1 Overview

The goal of a sparse filtering approach for neural networks classifier was to decrease the length of input to make it easier for neural network to learn correctly, as well to perform an enhanced feature selection. The sparse filtering algorithm tries to find features that allow us to distinguish examples. The sparse filtering algorithm for MatLab was implemented by Ngiam et al. (2011) at Stanford University.

5.2 Sparse Filtering on labeled data

The first version of a solution using a sparse filtering took as the input only the labeled train data and the test dataset (main dataset). The accuracy of classification depending on the number of features used as a parameter in the sparse filtering algorithm is presented in the Figure 2. The best

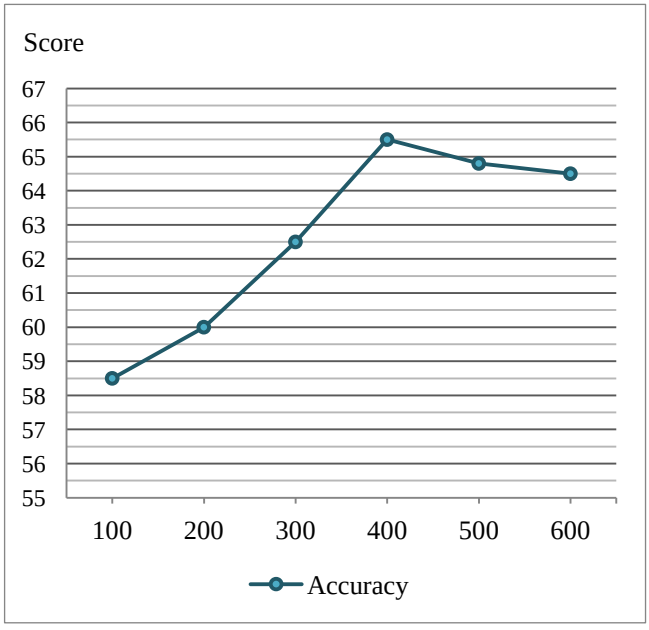

Figure 2. Result depending on number of features

scores were achieved after transforming 1875 to vector of 400 features. Only one level of feed forward was used, and the rest of detecting dependencies were given for a neural network classifier. Finally, the overall result on the main data increased significantly to 65.5%. Neural network configuration for data received form sparse filtering algorithm was different – the best occurred one layer of about one hundred of neurons, but nine neurons or even no hidden layer results were close to the best results.

5.3 Sparse Filtering on whole data

Sparse filtering on whole (test + train + unlabeled) gave slightly worse results. The reason was that sparse filtering algorithm normalizes each feature to be equally active among all dataset. Since features should allow to discriminate examples, the algorithm tried to distinct as many as possible dependencies among the whole data set, not only the main dataset. It tries to discriminate features beyond the obligatory test and train dataset, which cause lower quality of distinguishing samples in the test dataset. A length of a vector of features should follow a size of datasets. However, a long vector is unwanted for neural network classifier. On the other hand, sparse filtering on the whole dataset might find dependencies which are invisible in the small main dataset. The idea was to join features vector of main and whole datasets, leading to a vector of 800 (400 + 400) features. It gave ability to learn from both the local test set dependencies and whole dependencies in the given domain. Eventually, for this kind of input configuration of a one hidden layer consisting of 9 neurons was used: 800 – 9 – 9. The final score of the described approach was 67.18% and was achieved by our last submission sent to the Kaggle platform. This solution performed the best of all our predictions.

6. Results

The solution achieved the score of 67.18% which placed it in the top 3% of the submissions (6^{th} position on the leader board). The best performing algorithm received a result of 70.22%. It is worth noting that the overfit on the final private dataset, experienced by many of the teams, was significant. Ours approach clearly avoided overfitting. The results on the public and private sets were very similar: 67.26% and 67.18%.

References

Nissen, S. *Implementation of a Fast Artificial Neural Network Library FANN, Report.* Department of computer Science University of Copenhagen DIKU, 2003.

Ngiam, J., Koh, P., Chen Z., and Bhaskar S., Ng, A. Y. *Sparse filtering*. NIPS'11, p. 1125-1133, 2011.