CS/CNS/EE 156b Learning Systems - Project Summary

http://www.cs.caltech.edu/~ling/course/cs156b/

Ling Li, ling@cs.caltech.edu

December 7, 2001

My project is the letter recognition, a classification problem. The objective is to identify any black-and-white rectangular pixel display as one of the 26 capital letters in the English alphabet. Each sample image was randomly distorted and converted into 16 integer values from 0 through 15.

The training set S consists of 15997 samples. To get a basic idea of these samples, I made a census with the whole training set. All 26 classes have almost the same number of samples. For any single feature, the in-class variance is almost same as the between-class variance. Hence it is difficult to distinguish 26 classes by only few features.

I used a subset of S for training and another subset for validation. Denote the training error as E_r and validation error as E_v. The nearest-neighbor method gave an $E_v \approx 7\%$ when half of S was used as training, and an E_v around 5% when the whole S was used. This servers as a base performance for classifiers.

1 Neural networks

The network structure I used is 16-H-26. The inputs are normalized sample features. Output i is the "belief" the network has that the sample belongs to class i. Thus the one with biggest value is declaimed as the class. H could be one or two hidden layers. Neurons use $\tanh(\cdot)$ as the sigmoid function.

The normalization could be a simple scale, or a little more sophisticatedly, be a scale by feature's mean and variance. Comparisons between these two prefer the sophisticated one.

The desired outputs are -0.99 everywhere except 0.99 for the one corresponding to the class. The normal mean-squared-error (MSE) was used as the goal for neural networks to minimize. However, due to the way the classification result is produced, we may want to "pull" the output corresponding to the correct class more than other outputs. Thus a weighted MSE was also used, and proved to be better than the normal one.

In order of find out the required complexity of this classification problem, several settings of the hidden layers were tested. My final choice was 16-70-50-26. This was a little arbitrary, though I found out that 16-50-26 was insufficient.

2 Learning algorithms

Different learning algorithms, such as gradient descent, gradient descent with adaptive learning rate, gradient descent with momentum, line search, and conjugate-gradient, were tried. Line search and conjugate-gradient use almost the same CPU time and the latter is much better. Among those "fast" algorithms, gradient descent with momentum is superior for this problem. So I worked a lot with momentum method and conjugate-gradient.

In order to speed up the training, I implemented a look-up-table version for $\tanh(\cdot)$, which is twice fast. For conjugate-gradient, since the Polak-Ribiére variant was used, a not-so-precise line search was adopted, which also reduced the training time without sacrificing the performance.

When dealing with aggregation, I also tested stochastic gradient descent. It gave better results than the momentum, and comparative results as the conjugate-gradient while the training time was much less.

A network of 16-50-26 typically has E_r and E_v around 4% and 12%, while a 16-70-50-26 one has errors around 2% and 10%.

3 Aggregation

A simple aggregation method, average, improved E_v to around 7% when 50 hypotheses were aggregated. Bagging got higher errors due to its re-sampling characteristics. The E_v of individual networks were higher, around 14%.

I designed simple boost, which decreased E_v to about 5%. (I also compared 2 different definition of weights.) AdaBoost.M1 did almost the same thing. However, the E_r became 0 after only a few hypotheses were aggregated.

The above results are with the 16-50-26 networks. AdaBoost.M2 with 16-70-50-26 networks got better E_v's, usually under 4%.

4 Final Test

My final (best) hypothesis is an aggregation of 100 (16-70-50-26) neural networks. The individual neural networks were trained by stochastic gradient descent (stop at epoch 1000), and the whole aggregation was controlled by AdaBoost.M2. The first 12000 train samples were used. Its E_v is 2.75%.

My estimation of the out-of-sample error is a 95% confidence interval $[2.24\%, 3.26\%]$. The error on the test set is really 2.348%. The result with momentum gave worse validation and testing errors but better generalization.

It seems that E_v decreases when more data are used for training. When all conditions kept the same but 15000 samples were used for training, a test error of 1.949% was got, which is quite close to the ever-best result people got (1.8%).