CS760 Project: Dynamically Modifying Network structure

December 22, 1999

Glenn Fung

Introduction

To obtain good generalization ability one has to build into the network as much knowledge about the problem as possible (e.g. the topology of the input space) and limit the number of connections appropriately. If we choose a network that is too big, it will be able to memorize all the examples by forming a large lookup table, but will not generalize well to inputs that have not seen before. In other words, like all the statistical models, neural networks are subject to overfitting when there are too many parameters (e.g. weights) in the model.

A variety of methods have been proposed to shrink the number of network units (in special the number of hidden units) in an attempt to improve generalization accuracy, training accuracy and training efficiency.

There are two basic ideas to handle this issue, the first one is to begin with a network containing no hidden units (e.g. perceptron), then grow the network as needed by adding hidden units until the training error is reduced to some acceptable level. One example of this kind of approach is the Cascade-Correlation algorithm (Fahlman and Lebiere 1990).

The second basic idea is the following: begin with with a complex network and prune it as we find that certain connections are inessential. This was used in one of the largest applications of Neural Networks to date from Le Cun et all (1989). They have implemented a network designed to read zip codes on hand-addressed envelopes. The used an approach called "Brain

Damage" to remove weights from the initial fully connected model. After the network is initially trained, an information theorical approach identifies an optimal selection of connections that can be dropped (i.e. the weights are set to zero). The network is then retrained, and if it is performing as well or better, the process is repeated. This process was able to eliminate 3/4 of the weights, and improve overall performance on test data. In addition to removing connections, it is also possible to remove units that are not contributing much to the result.

Another approach is to have the network itself remove non-useful connections during training. This can be achieved by giving each connection w_{ij} a tendency to decay to zero. The approach that will be proposed here in this work is based in this idea.

In my case I want to try to reduce the number of hidden units of a given network adding a weighted term to the error function that "penalize" the number of hidden Units.

One Standard Approach

One of the known and simpler methods to remove non-useful connections during training (Hinton, 1996; scalettar and Zee, 1998; Kramer and Sangiovanni-Vicentelli, 1989] is to adding a penalty term w_{ij} to the original cost function E_0

$$E = E_0 + \frac{1}{2}\gamma \sum_{ij} w_{ij}^2 \tag{1}$$

When gradient descent $\Delta w_{ij} = -\nu \partial E / \partial w_{ij}$ is performed on the resulting total E this is equivalent to use:

$$w_{ij}^{new} = (1 - \gamma\eta) w_{ij}^{old}$$

after each update of w_{ij} based in the original cost function. η is the learning rate parameter.

$$E_0(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

where D as defined in the Mitchell book.

The Proposed Approach

While 1 clearly penalizes use of more wij's than necessary, it overly discourages uses of large weights; one large weight cost much more than many small ones. This can be cured by using a different penalty term:

$$E(\vec{w}) = (1-\gamma)\frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd}-o_{kd})^2 + \gamma \sum_{i\in hiddens} |\sum_j w_{ji}^2|_* \qquad (2)$$

where the γ is just a weight parameter and D as defined in the Mitchell book.

The $|x|_*$ function is the "step" function:

$$\begin{array}{cc} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{array}$$

In order to use a gradient method we have to use a smooth approximation instead. I will use the sigmoid function:

$$\sigma(x) = \frac{1}{1+\exp^{-\alpha x}}$$

for a appropriate α.

Notice that in the added penalty term in the objective function we are not trying to decay all the weights. We are trying to decay for each hidden unit i all the weights going from the inputs units j to this hidden unit i, this is represented by the expression:

$$|\sum_j w_{ji}^2|_* \qquad (3)$$

for each hidden j.

Numerical Tests

The Datasets

The computations were made using JAVA and three different public available datasets. All of them consist in two category classification Datasets. A brief description of the datasets is given below. In order to accelerate the training process the initial number of hidden units was set to 15 and just one layer of hidden units was assumed.

- Cancer housing data set. With 13 continuous (linear) valued attributes and 506 examples. This dataset is available from the UCI Machine Learning Repository.

- Sonar dataset. With 60 numeric continuous valued attributes and 208 examples. This dataset is available from the UCI Machine Learning Repository.

- Intrusion detection dataset. The dataset constructed in class. With 19 attributes either numeric continuous valued or nominal and 1000 examples.

All the computations were made using standard Ten-Fold cross validation and for each fold the training process was stopped after 100 epochs. the parameters that works better with each dataset were chosen using tuning sets.

The Parameters

The parameter α in the step function approximation:

$$\sigma(x) = \frac{1}{1 + \exp^{-\alpha x}}$$

was chosen to be in the interval $[4, 5]$.

The value of μ was chosen as follows:

- 0.1 for the intrusion dataset

- 0.2 for the sonar dataset

- 0.01 for the housing dataset

The η Value (Learning rate)

The setting of η was as follows:

- initially $\eta \in [0, 1]$.

- after each epoch if the accuracy improved, $\eta = \eta * 1.1$ (if the previous step size improved the accuracy try to "speed" the convergence by making the next step size bigger)
- after each epoch if the accuracy decreased, $\eta = \eta * 0.75$ (if the previous step size decreased the accuracy try to "correct" the step size by making the next step size smaller)
- after 5 epochs if the accuracy remains constant, $\eta = \eta(1 + r), r \in [-0.1, 0.1]$ (r random, if the accuracy was not changed in the last 5 epochs make a perturbation of the step size).

This choice of η in general worked as well or better than considering it constant.

General Comments about the Numerical Tests

The new approach worked pretty well in the three tested datasets. we can see after a few epochs that the suppression term for the arcs going into each hidden units (3) begins to decrease for some of the hidden units and grow for a few of them. This allows to the network to "discard" some of the hidden units. In order to decide which hidden units to keep I set a threshold of 10^{-5} for the quantity (3). It means if this quantity is less than 10^{-5} the hidden unit is considered an unnecessary unit. So after the training in all the ten folds (100 epochs each), for each fold an average over the number of unnecessary units in each fold was taken (usually it was very similar in all the folds) and then the network was trained again but this time removing the unnecessary hidden units. As we will see this procedure pruned the original network quite a bit without a significant loss of generalization. The other approach (1) reduces the magnitude of the weights on the network but keep similar magnitudes in all of them. This was not useful to decide what hidden units to keep and which ones to remove.

Averages and t-tests

In the three data sets four different experiments were performed and compared:

- A Neural Network was trained using the “standard” error function and using 15 hidden units. For future reference this experiment will be called “standard”.
- A Neural Network was trained using the proposed error function in order to “shrink” the number of hidden units. For future reference this experiment will be called “shrink”.
- Once we decided which hidden units are unnecessary a A Neural Network was trained using the “standard” error function and eliminating the unnecessary hidden units. For future reference this experiment will be called “reduced standard”.
- A Neural Network was trained using the error function described in (1) and using 15 hidden units. For future reference this experiment will be called “wsquare”.

t-test were made between this experiments and average accuracy was reported for each one.

The Intrusion Dataset

- For “standard” we have an test set correctness average of 98.6 % and 15 hidden units.
- For “shrink” we have an test set correctness of 98.8 % and an average of 5 significant hidden units.
- For “reduced standard” we have an test set correctness average of 98.8 % and 5 hidden units.
- For “wsquare” we have an test set correctness average of 98.6 % and 15 hidden units.
- The t-tests shown that all the three methods performed equivalently.

The Housing Dataset

- For “standard” we have an test set correctness average of 87.36 % and 15 hidden units.

- For “shrink” we have an test set correctness of 87.76 % and an average of 4 significant hidden units.
- For “reduced standard” we have an test set correctness average of 87.56 % and 4 hidden units.
- For “wsquare” we have an test set correctness average of 78.79 % and 15 hidden units.
- The t-tests shown that ”standard”, “shrink” and “reduced standard” performed equally well but significantly better than “wsquare”.

The Sonar Dataset

- For “standard” we have an test set correctness average of 79.27 % and 15 hidden units.
- For “shrink” we have an test set correctness of 76.90 % and an average of 4 significant hidden units.
- For “reduced standard” we have an test set correctness average of 77.28 % and 4 hidden units.
- For “wsquare” we have an test set correctness average of 76.36 % and 115 hidden units.
- The t-tests shown that “standard” performed equivalently to “shrink” and “reduced standard” but significantly better than “wsquare”.

Conclusions

As is shown in the numerical results in the three datasets the proposed approach always gives an idea of the number of unnecessary hidden units. The t-tests shown that always the suggested pruned neural network (“reduced standard”) performed equally well to both the original neural network (“standard”) to the obtained neural network with the suppressed weights (“shrink”) but with a considerable reduction in the number of used hidden units. the proposed approach also performed as well or better than the classic approach (“wsquare”).

Future Work

In this work the initial weights always were taken randomly, and since the gradient descent algorithm converges to a local minimum is not clear whether or not the number of unnecessary hidden units can be reduced, my numerical experience suggest than a good initial weight vector would lead to a better suppression of the hidden units. This idea can also be extended to try to suppress the weights going out for each input unit in order to do “feature selection” and also reduce the number of input units.