A Simple Neural Network Contextual Classifier

Jens Tidemann and Allan Aasbjerg Nielsen

IMM – Department of Mathematical Modelling,
Technical University of Denmark, Building 321,
DK–2800 Lyngby, Denmark.
http://www.imm.dtu.dk

Summary. In this paper we describe a neural network used to make a simple contextual classifier using a two layer feed-forward network. The best number of hidden units is chosen by training a network with too many hidden units. We then prune the network using Optimal Brain Damage (OBD). The pruned networks have a better generalisation error because they only have the weights that reflect the structure of the data and not the noise. We study the possibility of using a Network Information Criterion (NIC) to decide when to stop pruning. When we use NIC we can estimate the test error of a network without using an independent validation set.

As a case study we use a four band Landsat-2 Multispectral Scanner (MSS) image from southern Greenland. To classify a pixel in the non-contextual case we use the four variables from the MSS bands only. In the simple contextual case we augment the feature vector with the four mean values of the MSS bands from the four nearest neighbours. We notice an increase in the number of correct classified pixels when using the contextual classifier. Also, the application of the simple contextual classifier gives a small overall increase in the posterior probability.

1. Introduction

In this paper we study the use of neural networks for classification of remote sensing images. When classifying remote sensing data it is important to use contextual information because the data often have a lot of noise.

When working with neural networks it is important to find the correct network size. We study the use of pruning weights in the network. Thereby we can train a network that is too big, prune some of the connections and then choose the network that is optimal. We investigate the use of a Network Information Criterion to find the optimal network.

In section 2 we describe the 2-layered network architecture that we use and in section 3 we discuss the optimisation method. In section 4 we discuss Optimal Brain Damage, the method that we use to remove weights from the network. Section 5 deals with a Network Information Criterion that can be used to test which of the many networks we train is best. In section 6 we discuss the contextual classification and section 7 is devoted to a case study where we apply our methods to an image taken with the Landsat-2 Multispectral Scanner (MSS).

2. Network Architecture

We use a 2-layer feed forward network. In the hidden layer we use hyperbolic tangent as activation function. The weight from input i to hidden unit j is denoted as w_{ji}. We include a bias by setting $x_0 = 1$. So a hidden unit is described by the equations

$$a_j = \sum_i w_{ji} x_i \qquad , \; x_0 = 1 \tag{2.1}$$

$$z_j = \tanh(a_j) \tag{2.2}$$

In the output layer we use soft-max as activation function (a_k is found by weighting the output from the hidden layer z_j, like (2.1))

$$y_k = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})} \tag{2.3}$$

We use the cross-entropy as energy function

$$E = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{c} t_k^n \ln(\frac{y_k^n}{t_k^n}) \tag{2.4}$$

where N is the number of training samples, c is the number of classes and t_k^n is the target value for observation n in class k.

By choosing soft-max as activation function in the output layer and cross-entropy as energy function we have ensured that the network is optimal for classification and we can interpret the output as probabilities [1].

3. Network Training

We use a Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi Newton method [4] for optimising the network. In a Newton method you make a second order estimation of the energy function and make a step towards the minimum

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \alpha^{\tau} \mathbf{G}^{\tau} \mathbf{g}^{\tau} \tag{3.1}$$

The α parameter is used to make sure that the energy is decreasing. It is found by line minimisation. When using a second order method you have to calculate the second derivative of the energy function (the Hessian matrix). This gives two problems. The Hessian might not be positive definite and is computationally expensive to estimate. Instead we use a quasi Newton method. This is much faster and we can ensure that the estimate of the Hessian will always be positive definite. We update the estimate of the Hessian after each step. The updating equation for the estimate of the Hessian ($\mathbf{G}^{\tau}$) is

$$\mathbf{G}^{\tau+1} = \mathbf{G}^{\tau} + \frac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{v}} - \frac{(\mathbf{G}^{\tau}\mathbf{v})(\mathbf{v}^T\mathbf{G}^{\tau})}{\mathbf{v}^T\mathbf{G}^{\tau}\mathbf{v}} + (\mathbf{v}^T\mathbf{G}^{\tau}\mathbf{v})\mathbf{u}\mathbf{u}^T \tag{3.2}$$

We have defined these vectors:

$$\mathbf{p} = \mathbf{w}^{\tau+1} - \mathbf{w}^{\tau} \tag{3.3}$$

$$\mathbf{v} = \mathbf{g}^{\tau+1} - \mathbf{g}^{\tau} \tag{3.4}$$

$$\mathbf{u} = \frac{\mathbf{p}}{\mathbf{p}^T\mathbf{v}} - \frac{\mathbf{G}^{\tau}\mathbf{v}}{\mathbf{v}^T\mathbf{G}^{\tau}\mathbf{v}} \tag{3.5}$$

Where $\mathbf{w}$ are the weights and $\mathbf{g}$ is the gradient. When we have estimated the Hessian we make a Newton step (3.1) in the descent direction of the energy function. In the line minimisation we make a parabolic fit [4] to the energy function. The algorithm is initiated by setting $\mathbf{G}^0$ to the identity matrix, so the first step is a gradient descent step.

4. Optimal Brain Damage

When we start training the network we have a fully connected network. But it is not certain that all the connections are needed for the classification. By pruning the network we increase the generalisation of the network because we reduce the number of parameters in the model without increasing the energy function significantly. The idea in Optimal Brain Damage (OBD) [3] is to estimate the increase in the energy function when we remove one weight from the network (the saliency for that weight). We then remove the weights with the lowest saliencies and retrain the network. We continue this process until the energy function starts growing dramatically.

OBD can be used to decide how many hidden units will be needed in the network. We can train a network that we know is too big and then we can prune it. During the pruning some of the hidden units will be removed. We can also use it to decide if any of the features we use in the network are unimportant. If a feature does not contribute to the classification all weights for that feature will be removed from the network.

When we estimate the increase in energy we make a Taylor series for the change in the energy. We assume that we are in a local minimum so the gradient is $\mathbf{0}$. Instead of calculating the full Hessian matrix we assume that it is diagonal and we make a diagonal approximation to the Hessian

$$\begin{aligned}\delta E &= \frac{1}{2}\delta\mathbf{w}^T\mathbf{H}\delta\mathbf{w} + O(\|\delta\mathbf{w}\|^3) \\ &\approx \frac{1}{2}\sum_i \delta w_i^2 H_{ii}\end{aligned} \tag{4.1}$$

When we calculate the diagonal approximation to the Hessian we further assume that off-diagonal elements have no influence on the diagonal elements

$$\frac{\partial^2 E^n}{\partial w_{ji}^2} = \frac{\partial^2 E^n}{\partial a_i^2} x_j^2 \tag{4.2}$$

$$\frac{\partial^2 E^n}{\partial a_j^2} = g'(a_j)^2 \sum_k w_{kj}^2 \frac{\partial^2 E^n}{\partial a_k^2} + g''(a_j) \sum_k w_{kj} \frac{\partial E^n}{\partial a_k} \tag{4.3}$$

By making this assumption we can reduce the computational time dramatically.

5. Network Information Criterion

The Network Information Criterion (NIC) [5] estimates the test energy for a neural network without using a test dataset. So when training and pruning a neural network you do not need an independent validation dataset to select the best network. When you calculate the NIC you use your energy function (2.4) and add a term for the complexity of the model

$$\mathrm{NIC} = E(\mathbf{x}, \mathbf{t}, \mathbf{w}) + \frac{1}{N} tr(\mathbf{G}(\mathbf{w})\mathbf{Q}(\mathbf{w})^{-1}) \tag{5.1}$$

where $\mathbf{x}$ is a matrix with all input data for the network and $\mathbf{t}$ is a matrix with all the target values. We have defined

$$\mathbf{G}(\mathbf{w}) = V[\nabla e(\mathbf{x}^n, \mathbf{t}^n, \mathbf{w})] \tag{5.2}$$

$$\mathbf{Q}(\mathbf{w}) = E[\nabla\nabla e(\mathbf{x}^n, \mathbf{t}^n, \mathbf{w})] \tag{5.3}$$

where e is the energy for one observation from the dataset and ∇ means the gradient.

Murata [5] argues that the models you compare with NIC should be hierarchical, so that each model is a sub-model of the other models. But Ripley [6] proves that this is not necessary. It is discussed in [6] how you should estimate (5.2) and (5.3). If NIC should work properly it requires that there is a strong single local minimum. If this is not the case the complexity term might change dramatically depending on which local minimum you end up in. Another assumption is that there is enough data in the training set.

6. Contextual Classification

We wish to use contextual information in the classification. We could use the features from all the neighbouring pixels. But that would make the number of weights 5 times bigger in the input layer. Instead we assume in our model that the pixel on the north side of the pixel has the same influence on the classification as the pixel on the east, south and west side. We can do this by forcing the weights in the input layer to be equal for the features from the neighbouring pixels.

But we can find a simpler approach if we look at equation (2.1). If we force some of the weights to be equal we get

$$\begin{aligned} a_j &= \cdots + wx_N + wx_E + wx_S + wx_W + \cdots \\ &= \cdots + w(x_N + x_E + x_S + x_W) + \cdots \end{aligned} \tag{6.1}$$

From this we see that instead of forcing the network to have equal weights we sum the features that have equal weights before we start training the network.

In the case of four neighbours we get the Switzer-filter [7] except for a scaling factor. In the Switzer-filter you take the mean value of the neighbouring pixels

$$y = \frac{1}{4}(x_N + x_E + x_S + x_W) \tag{6.2}$$

So we can introduce contextual information to the network by using the Switzer-filter. We thereby avoid a huge increase in network size.

7. A Case Study

As a case study we use a 512×512 pixel four band Landsat-2 Multispectral Scanner (MSS) image from southern Greenland. In Figure 7.1 is shown two of the bands of the image. We have five different training classes in the image with a total of 43000 pixels. We split the dataset up so $\frac{3}{4}$ of the pixels are used for training and the last $\frac{1}{4}$ are used for testing.

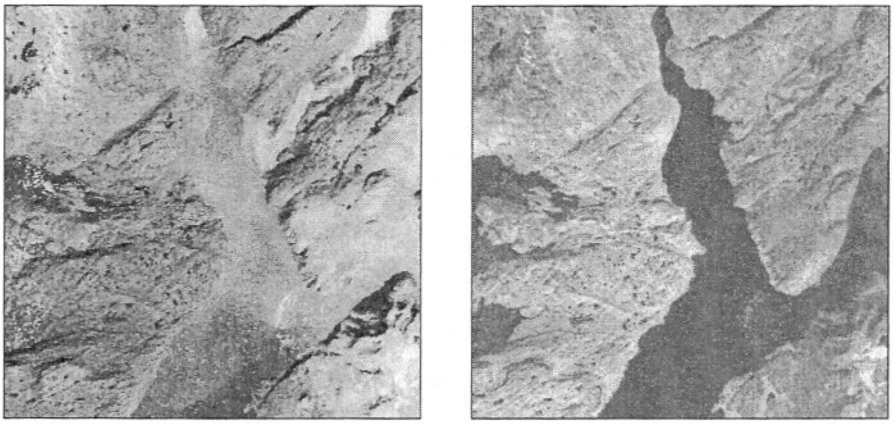

Fig. 7.1. Bands 4 and 7 of the Landsat MSS data.

We first study the NIC. In Figure 7.2 we have plotted the training- and test energy and the NIC against the number of weights in the network during one training and pruning session. We notice that the NIC is very unstable when there are many weights in the network. This might be because the number of weights is high compared with the number of data samples. So we have to be careful when we use NIC particularly when we have many weights in the network. But for a smaller number of weights it seems to be useful. When we prune the network we are able to remove half of the weights in the network and as can be seen from Table 7.1 we get a small increase in the generalisation because the pruned networks perform better. We train ten networks and prune them. We choose the network with the lowest NIC if the NIC is stable.

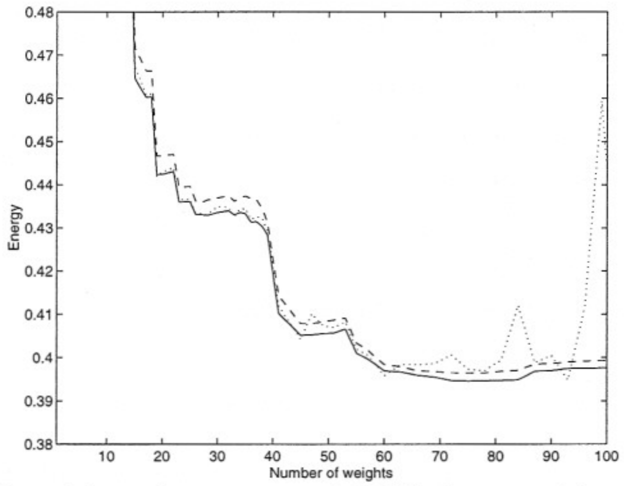

Fig. 7.2. The training and test energy and NIC during one training and pruning session. The full line is the training error, the dashed line is the test error and the dotted line is the NIC.

In Figure 7.3 we see the classified images. It is seen that in the image where we use the contextual classifier we get a significant reduction in the number of single pixels that are classified into another class than their neighbouring pixels. If we study Table 7.1 we see how many pixels are classified correctly for each of the classifiers. The classification of all classes is improved when we use the contextual classifier. When using the contextual classifier there is an overall increase in the probability which means that the network is more certain that it has made the correct classification.

We have compared our results with other methods. If we use linear or quadratic discriminant analysis [6, 1] we get results that are not as good as the non-contextual neural network. If we use CART [2] we get results that are comparable with neural networks. We have also compared our results with

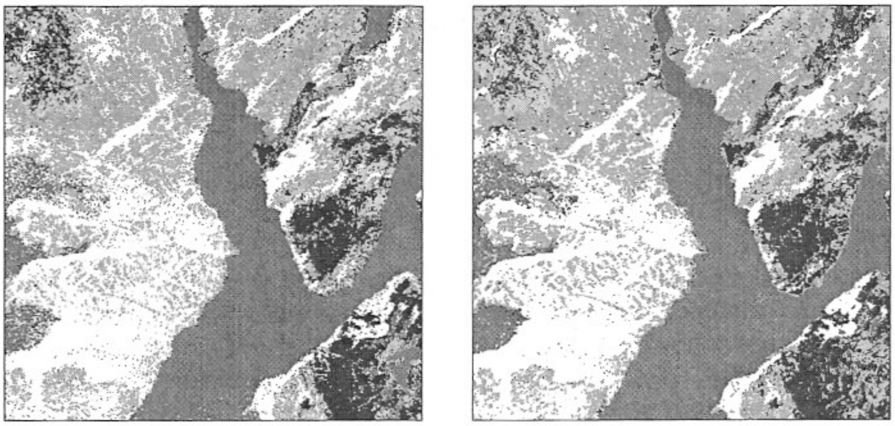

Fig. 7.3. The left image shows the classified image when we use the non-contextual classifier. The right image shows the image classified using the contextual classifier.

Table 7.1. Percentage of pixels that are classified correctly in the test set with the methods we have used.

Method	Correctly classified
Linear discriminant analysis	78.5 %
Quadratic discriminant analysis	79.7 %
Non-contextual NN (not pruned)	81.6 %
Non-contextual NN (pruned)	81.8 %
Non-contextual CART	81.8 %
Non-contextual NN (pruned) and modus filtering	84.8 %
Contextual CART	85.1 %
Contextual NN (not pruned)	86.6 %
Contextual NN (pruned)	86.8 %

another simple contextual method. For the image classified with the non-contextual classifier we have made a modus filtering of the classified image with a 3×3 kernel. The modus filter counts how many pixels belong to each class in the kernel and then assigns the most frequent class to the centre pixel. The result is shown in Table 7.1. It is better than the non-contextual classifier but not as good as the classifier where we use the contextual information as features to the neural network.

8. Conclusion

We have studied pruning of the network weights and we have come to the result that approximately half the weights can be removed from the network and this reduction results in an increase in generalisation error. The use of simple contextual information can improve the classification significantly. Also we get the best results if we use contextual information as input for the neural networks instead of doing post-processing on the data.

References

1. C. M. Bishop, *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.
2. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees.* Wadsworth and Brooks/Cole, 1984.
3. Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage", *Advances in Neural Information Processing Systems*, vol. 2, pp. 598–605, 1990.
4. D. G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, 2nd edition, 1984.
5. N. Murata, S. Yoshizawa, and S. Amari, "Network information criterion - determining the number of hidden units for an artificial neural network model", *IEEE-Transactions on Neural Networks*, vol. 5, no. 6, pp. 865–872, 1994.
6. B. D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996.
7. P. Switzer, "Extensions of linear discriminant analysis for statistical classification of remotely sensed satellite imagery", *Mathematical Geology*, vol. 12, no. 4, pp. 367–376, 1980.