MML INFERENCE OF SINGLE-LAYER NEURAL NETWORKS

ENES MAKALIC
School of Computer Science and Software Engineering
Monash University
Clayton, VIC 3800, Australia
email: enesm@csse.monash.edu.au

LLOYD ALLISON AND DAVID L. DOWE
School of Computer Science and Software Engineering
Monash University
Clayton, VIC 3800, Australia
email: lloyd@csse.monash.edu.au

ABSTRACT
The architecture selection problem is of great importance when designing neural networks. A network that is too simple does not learn the problem sufficiently well. Conversely, a larger than necessary network presumably indicates overfitting and provides low generalisation performance. This paper presents a novel architecture selection criterion for single hidden layer feedforward networks. The optimal network size is determined using a version of the Minimum Message Length (MML) inference method. Performance is demonstrated on several problems and compared with a Minimum Description Length (MDL) based selection criterion.

KEY WORDS
Architecture selection, Neural Networks, MML, MDL

1 Introduction

At the present time, the Multilayer Perceptron (MLP) is perhaps the most popular neural network type in use. MLPs are regularly trained with a first (for example, backpropagation [1]) or second order (for example, Levenberg-Marquardt [2]) training algorithm. The success of the training process depends on the chosen network architecture - that is, the number of hidden layers, hidden neurons etc. For the purposes of this paper, we are only concerned with single hidden layer feedforward neural networks. These networks are frequently used and can model any continuous function [3].

Selecting the number of hidden neurons for a particular problem is a non-trivial task. If we choose a network that is too small, the network may not be able to learn the problem sufficiently well. In contrast, a network that is too large may over-fit and will commonly exhibit low generalisation performance. Consequently, the task of selecting the initial architecture is important to researchers in this field.

There are two intuitive approaches to the architecture selection problem: constructive and pruning. Constructive algorithms [4] start with a trivial network and progressively add more hidden neurons until a satisfactory solution is found. When to add extra neurons and when to stop the growing process are two important problems a constructive algorithm must address. Conversely, pruning algorithms [5] start with a complex network and prune it by removing unnecessary components. The primary difficulty with pruning algorithms is the strategy used to decide whether a network component is removed.

Constructive algorithms have two important advantages over pruning. Firstly, it is trivial to specify the initial architecture before commencing the construction process. This is not obvious with pruning strategies because some initial topologies will lead to better solutions than others. Furthermore, pruning algorithms tend to spend the majority of time on larger than necessary neural networks. Clearly, the same is not true for constructive strategies where simple networks are examined first.

The primary difficulty encountered when using constructive algorithms is when to stop the construction process. Allowing the network to grow uncontrollably must be avoided as it will no doubt result in over-fitting. This paper introduces an architecture selection criterion based on Minimum Message Length (MML) that addresses the aforementioned issue. In the next section, we discuss the basic theory behind MML and its two part message scheme. Section 3 contains the MML87 approximation [6], which we use to evaluate neural network architectures. Our MML inference method and the block-diagonal Fisher information approximation are given in Sect. 4. Section 5 contains the performance analysis of the MML selection criterion. A comparison with a version [7, 8] of the Minimum Description Length (MDL) criterion [9] is also provided. Finally, a summary of the paper is given in Sect. 6.

2 Overview of MML

The MML [10, 6, 11] theory of inductive inference provides an objective function that may be used to estimate the goodness of an inferred model, θ. In general, MML assumes the existence of an arbitrary Sender wishing to communicate a message to a Receiver via a noiseless transmission channel. We can think of this message as some data, x, which the Receiver does not know. A natural objective measure of the goodness of the data encoding is the message length (for example, in bits). That is, the encoding with the shortest message length is said to be the 'best'. MML requires the message to be transmitted in two parts:

1. an encoding of the model, θ, and

2. an encoding of the data given the model, $x|\theta$.

The first part of the message is arranged to be the answer to an inference problem that we are interested in. For example, we may be trying to infer the best neural network that explains some data. MML states that the optimal model is one that minimises the length of this two part message - that is, the model with the shortest message length. Importantly, MML does not require us to work out the encoding of the two part message. Rather, we only need to be able to calculate the length of the message. Obviously, the first part of the message is short if the model is simple. Conversely, if the chosen model is complicated, the first part of the message is long. The second part of the message reflects how well the model fits the data. A model that fits the data well will naturally have a shorter second part message length than a model that fits the data poorly. We note that a simple model may fit the data poorly, whereas a more complex model may fit the data well. Therefore, when using MML we are trying to optimise the trade-off between model simplicity and goodness of fit.

3 MML Inference

MML87 [6] is an efficient approximation to strict MML [12, 6, 11, 13] and states that the total message length for a model Θ with parameters $\vec{\theta}$ is given by:

$$\begin{aligned}\text{msgLen}(\Theta) = &-\log\left(\frac{h(\vec{\theta})}{\kappa_n^{n/2}\sqrt{F(\vec{\theta})}}\right) \\ &-\log f(x|\vec{\theta}) + \frac{n}{2}\end{aligned} \tag{1}$$

where $h\left(\vec{\theta}\right)$ is the prior probability, $f(x|\vec{\theta})$ is the likelihood function, n is the number of parameters, κ_n is a dimension constant (also called lattice constant; the first two are: $\kappa_1 = \frac{1}{12}$, $\kappa_2 = \frac{5}{36\sqrt{3}}$) and $F(\vec{\theta})$ is the determinant of the expected Fisher information matrix, whose entries (i, j) are:

$$\sum_{x\,\in\mathcal{X}} f(x|\vec{\theta})\frac{\partial^2}{\partial\theta_i\partial\theta_j}\left(-\log f(x|\vec{\theta})\right) \quad . \tag{2}$$

The Fisher information defines how sensitive the likelihood function is to the parameters $\vec{\theta}$. We use the Fisher information in MML to determine how accurately to state the model. For example, if the second derivatives of the likelihood function are small, the Gaussian curvature of the likelihood function is low, indicating that we need to state the parameters less precisely. Conversely, a high curvature implies that the parameters need to be stated more accurately. A model that minimises the total message length approximation (1) is said to be 'optimal'.

4 MML Inference of Neural Networks

As previously mentioned, we are only concerned with fully connected single hidden layer neural networks. Furthermore, we limit our discussion to networks with one output. However, it is trivial to generalise all presented results to networks with multiple outputs [14]. We now introduce the notation used in this paper.

The output of a single hidden layer feedforward network for pattern p with I inputs and H hidden neurons is given below:

$$y_p^{(2)} = \sum_{h=1}^{H}\left(w_h^{(2)}y_{ph}^{(1)}\right) + b^{(2)} \tag{3}$$

where $\vec{w}^{(2)}$ and $b^{(2)}$ are the output weights and output bias respectively. That is, $w_h^{(2)}$ denotes the weight from hidden neuron h to the output neuron. Furthermore, the output of hidden neuron h, given pattern p and inputs $\vec{x}_p \in \mathbb{R}^I$ is:

$$y_{ph}^{(1)} = f^{(1)}\left(\sum_{i=1}^{I}\left(w_{ih}^{(1)}x_{pi}\right) + b_h^{(1)}\right) \tag{4}$$

where $f^{(1)}$ is the sigmoid function, $f^{(1)}(x) = \frac{1}{1+e^{-x}}$. Here, $w_{ih}^{(1)}$ denotes the weight from input i to hidden neuron h and $b_h^{(1)}$ represents the bias for hidden neuron h.

We would like the network to learn a function, ϕ, from the input space, $\mathcal{X} \in \mathbb{R}^I$, to the output space $\mathcal{Y} \in \mathbb{R}$; that is, $\phi : \mathcal{X} \to \mathcal{Y}$. In most cases, the function ϕ is unknown and we are only given a finite set of input-output examples, T, of ϕ, where $T = \{(\vec{x_1}, t_1), \ldots, (\vec{x_p}, t_p), \ldots, (\vec{x}_N, t_N)\}$. T is commonly referred to as the training set in neural network nomenclature. We use the least squares criterion to measure the network output error for a given pattern:

$$E(t_p, y_p^{(2)}) = \left(y_p^{(2)} - t_p\right)^2 \quad . \tag{5}$$

In order to apply MML to neural networks, we require a likelihood function, the Fisher information (see Sect. 4.2) and the priors on all network weights and biases (see Sect. 4.3). The likelihood function is presented in the next section.

4.1 Likelihood Function

It is desirable to incorporate the error function given in (5) into a likelihood function for a Gaussian noise model with a mean $\mu = 0$ and a standard deviation $\sigma = \frac{1}{\beta}$:

$$f(\vec{t}|\beta, \vec{w}) = \prod_{p\,=1}^{N}\frac{\beta}{\sqrt{2\pi}}e^{-\frac{\beta^2}{2}\left(y_p^{(2)}-t_p\right)^2} \tag{6}$$

where $\vec{w} = \{\vec{w}^{(1)}, \vec{w}^{(2)}, \vec{b}^{(1)}, b^{(2)}\}$; that is, all weights and biases in the network. The negative log-likelihood function is:

$$\begin{aligned}L(\vec{t}|\beta, \vec{w}) = &\frac{\beta^2}{2}\sum_{p=1}^{N}\left(y_p^{(2)} - t_p\right)^2 \\ &- N\log\beta + \frac{N}{2}\log(2\pi) \quad .\end{aligned} \tag{7}$$

Complete derivations of all formulae in this paper can be found in [14].

4.2 Fisher Information

The Fisher information represents how sensitive the likelihood function is to the network parameters (that is, weights, biases and β). Parameters that cause little or no change in the likelihood function can be stated less precisely. Similarly, parameters that cause great fluctuations in the likelihood must be stated more accurately (see Sect. 3).

The full Fisher information matrix for anything but the most trivial network is difficult to compute and prone to numerical instabilities. Therefore, rather than calculating the Fisher information for the entire neural network, we opted for the following approximation. The matrix is divided into blocks where each block corresponds to a single neuron. The size of each block depends on the number of inputs into the network and/or the number of hidden neurons. The output neuron block is of size $(H+2) \times (H+2)$, where H is the number of hidden neurons. Conversely, the dimensions of each hidden neuron block are identical and depend only on the number of network inputs. That is, each hidden neuron block is $(I+1) \times (I+1)$, where I is the number of network inputs. Figure 1 depicts a diagram of this approximation. It is important to note that the block-diagonal approximation assumes individual neurons are independent. In general, this is a safe thing to do and on occasion may result in the message length being overestimated.

$$\beta^2 \begin{pmatrix} \sum_{p=1}^{N}\left(y_{p1}^{(1)}\right)^2 & \sum_{p=1}^{N}\left(y_{p1}^{(1)}y_{p2}^{(1)}\right) & \cdots & \sum_{p=1}^{N}\left(y_{p1}^{(1)}y_{ph}^{(1)}\right) & \sum_{p=1}^{N}\left(y_{p1}^{(1)}\right) & 0 \\ \sum_{p=1}^{N}\left(y_{p2}^{(1)}y_{p1}^{(1)}\right) & \sum_{p=1}^{N}\left(y_{p2}^{(1)}\right)^2 & \cdots & \sum_{p=1}^{N}\left(y_{p2}^{(1)}y_{ph}^{(1)}\right) & \sum_{p=1}^{N}\left(y_{p2}^{(1)}\right) & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \sum_{p=1}^{N}\left(y_{ph}^{(1)}y_{p1}^{(1)}\right) & \sum_{p=1}^{N}\left(y_{ph}^{(1)}y_{p2}^{(1)}\right) & \cdots & \sum_{p=1}^{N}\left(y_{ph}^{(1)}\right)^2 & \sum_{p=1}^{N}\left(y_{ph}^{(1)}\right) & 0 \\ \sum_{p=1}^{N}\left(y_{p1}^{(1)}\right) & \sum_{p=1}^{N}\left(y_{p2}^{(1)}\right) & \cdots & \sum_{p=1}^{N}\left(y_{ph}^{(1)}\right) & N & 0 \\ 0 & 0 & \cdots & 0 & 0 & \frac{2N}{\beta^4} \end{pmatrix} \tag{8}$$

$$\beta^2\left(w_h^{(2)}\right)^2 \begin{pmatrix} \left(x_{p1}f_p'\right)^2 & \left[x_{p1}x_{p2}\left(f_p'\right)^2\right] & \cdots & \left[x_{p1}x_{pI}\left(f_p'\right)^2\right] & \left(x_{p1}\left(f_p'\right)^2\right) \\ \left[x_{p2}x_{p1}\left(f_p'\right)^2\right] & \left(x_{p2}f_p'\right)^2 & \cdots & \left[x_{p2}x_{pI}\left(f_p'\right)^2\right] & \left(x_{p2}\left(f_p'\right)^2\right) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \left[x_{pI}x_{p1}\left(f_p'\right)^2\right] & \left[x_{pI}x_{p2}\left(f_p'\right)^2\right] & \cdots & \left(x_{pI}f_p'\right)^2 & \left(x_{pI}\left(f_p'\right)^2\right) \\ \left(x_{p1}\left(f_p'\right)^2\right) & \left(x_{p2}\left(f_p'\right)^2\right) & \cdots & \left(x_{pI}\left(f_p'\right)^2\right) & \left(f_p'\right)^2 \end{pmatrix} \tag{9}$$

A desirable consequence of the block-division process is the simplified matrix determinant calculation; the full determinant is equal to the product of the individual block determinants. This property makes our approximation less prone to numerical errors. The Fisher information matrix for an output neuron, $F_o(\vec{w}, \beta)$, is given in (8). Additionally, the Fisher information matrix for a hidden neuron h, $F_h(\vec{w})$, is shown in (9). Here, f_p' denotes the derivative of the transfer function. For sigmoidal units, recalling equation (4), $f_p' = f_{ph}' = y_{ph}^{(1)}\left(1 - y_{ph}^{(1)}\right)$. The summation over all patterns is omitted from (9) for reasons of clarity.

We note that the Fisher information matrix, is closely related to the Hessian matrix which is known to be ill-conditioned and prone to numerical instabilities [15]. Similarly, the Fisher information matrix is badly behaved when: a hidden neuron is saturated, or a neuron's output weight is small, causing the influence of all hidden weights into this neuron to be greatly reduced. If a hidden neuron is saturated, its first (and second) derivatives tend toward zero, causing the determinant of the whole matrix to approach zero. In addition, we notice that the $\left(w_h^{(2)}\right)^2$ factor is present in every hidden neuron Fisher information matrix entry. This agrees with intuition:

1. a small output weight causes the influence of the corresponding hidden neuron's incoming weights on the likelihood function to be significantly reduced. In extreme cases this problem will show up in the matrix as rows (columns) of zero entries,
2. a large output weight greatly increases the influence of the hidden neuron's incoming weights on the likelihood function.

However, it is perhaps surprising that each matrix element is multiplied by the squared weight. When evaluating the determinant of such a matrix, the effect becomes much more drastic:

$$\det\left(F_h(\vec{w})\right) = \left(w_h^{(2)}\right)^{2(I+1)} \times \det\left(\beta^2 R\right) \tag{10}$$

where I is the number of network inputs and R denotes the rest of the matrix with the squared weight taken out.

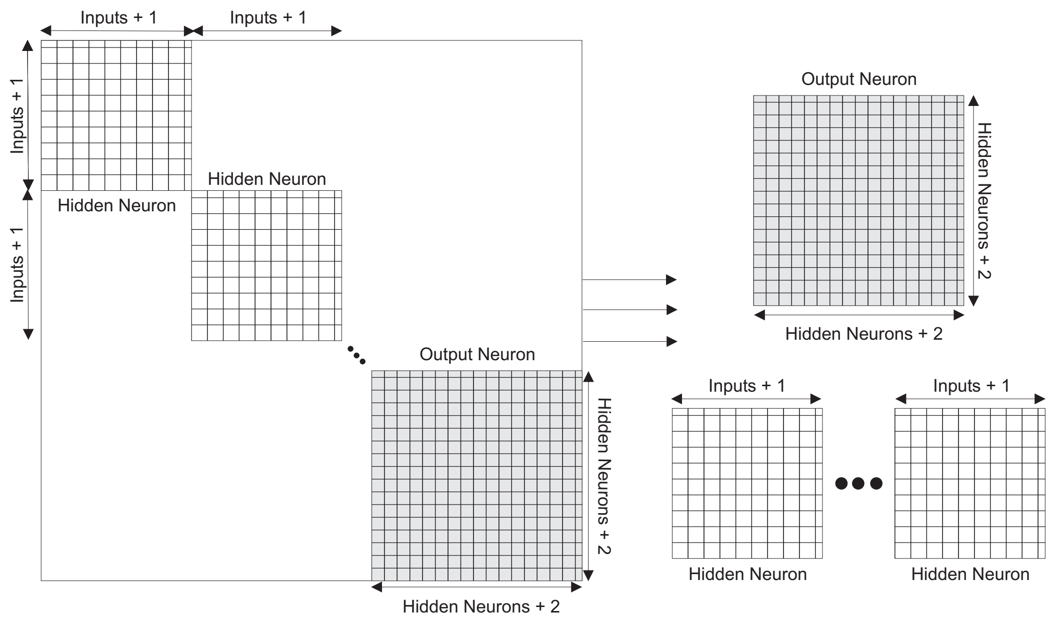

Figure 1. Fisher information block-diagonal approximation

4.3 Prior Probabilities

We opted for the following symmetric prior on weights and biases:

$$h_w(w) \; = \frac{1}{40} \mathrm{sech}^2 \left(\frac{w}{20} \right) \quad . \tag{11}$$

Furthermore, the prior on β, which is improper, is:

$$h_\beta \left(\beta \right) \; = \frac{1}{\beta} \quad . \tag{12}$$

4.4 Message Length of a Neural Network

From (1), the message length of a single hidden layer feed-forward neural network is:

$$\begin{aligned} \mathrm{msgLen} = \log^* H - \log \left(h_w(\vec{w}, \vec{b}) h_\beta(\beta) \right) \\ + \frac{1}{2} \log F + L + \frac{n}{2} \left(\log \kappa_n + 1 \right) \end{aligned} \tag{13}$$

where the likelihood function, the Fisher information and parameter priors are defined in Sect. 4.1, Sect. 4.2 and Sect. 4.3 respectively. The $\log^* H$ [16] in the message length expression denotes the cost of transmitting the network architecture; that is, the number of hidden neurons. We define it as:

$$\log^* H \; = \log(H) + \log\left(\log H\right) + \log(2.865) \quad . \tag{14}$$

Individual connections need not be transmitted because we assume fully connected networks.

4.5 MML Estimator for β

The determinant of the full Fisher information can be expressed as:

$$F \; = \beta^{2(M+1)} \det(C) \tag{15}$$

where C is the Fisher information matrix with β^2 taken out and M is the total number of weights and biases in the network. Differentiating (13) w.r.t. β and setting $\frac{\partial \mathrm{msgLen}}{\partial \beta} = 0$ yields the MML estimator:

$$\hat{\beta}_{MML} \quad = \quad \sqrt{\frac{N - M - 2}{\sum_{p=1}^{N} \left(y_p^{(2)} - t_p \right)^2}} \quad . \tag{16}$$

For (16) to be meaningful, we require that $N > (M + 2)$. In fact, we prefer that $N >> M$. Intuitively, inference of a model where the number of parameters is greater than the number of data does not make sense (although it can be done). Hence, the above constraint is minor.

5 Results and Discussion

The block-diagonal version of MML87 was tested on several artificial problems where the optimal neural network architecture is known. In addition, we examined real problems where the optimal neural network size is unknown. The neural network software was implemented in a data mining platform CDMS [17]. We compared our method to the MDL-based neural network selection criterion [7, 8]. Table 1 shows the functions used during testing. The testing procedure for each artificial function is explained below:

Function name	Function definition	Optimal network size
f_1	$f_1(x) = x^2 + N\left(0, \sigma^2\right)$	1 - 2 - 1
f_2	$f_2(x) = e^{-x}\sin(2\pi x) + N\left(0, \sigma^2\right)$	1 - 5 - 1
f_3	$f_3(x) = x(x-1)(x+1) + N\left(0, \sigma^2\right)$	1 - 3 - 1
Iris	UCI Repository of machine learning databases[18]	Unknown

Table 1. Functions used for method evaluation.

1. Train a neural network of optimal size using the conjugate gradient training algorithm with Polak-Ribiere updates. Since we know what the answer is for artificial problems, this is a reasonable thing to do.
2. Add noise to the output of this trained neural network.
3. Attempt to infer the original network (plus noise) from (1) using our MML approximation and the MDL-based criterion.

During the testing process, the number of data samples, N, was varied from $N = 25$ to $N = 400$. The noise standard deviation was also varied from $\sigma = 0.01$ to $\sigma = 0.35$. All function inputs were sampled from the uniform distribution $U(-1, 1)$. We investigated the effect of data size and noise on both the MML and MDL-based [7, 8] selection criteria. For each function, only selected curves are shown for clarity. A more detailed performance analysis of the presented architecture selection criteria on these (and other) functions can be found in [14].

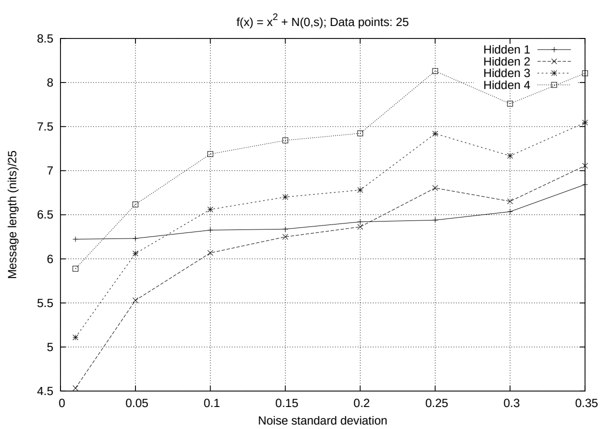

Figure 2. MML inference of function f_1, $N = 25$

The first function tested, namely f_1, was the simplest and required only two hidden neurons to be modeled accurately. MML infers the optimal architecture when $\sigma < 0.25$ even with 25 data points. However, as more noise is added, $\sigma \geq 0.25$, MML underfitted and picked the simpler model: one hidden neuron (see Fig. 2). This is of course a proper thing to do. Identical results were observed when the dataset was increased to 50 samples. For $N = 100$, MML inferred the optimal architecture for noise levels where $\sigma \leq 0.30$; when $\sigma = 0.35$ the MML criterion again underfitted. Our criterion inferred the optimal architecture for all noise levels using datasets of 200 and 400 points.

The MDL-based criterion inferred the optimal architecture for $\sigma = 0.01, 0.05, 0.15, 0.20$ and 0.25 using a dataset of 25 points. However, when $\sigma = 0.10, 0.30, 0.35$, the MDL-based criterion overfitted and inferred three, four and three hidden neurons to be optimal for the respective σ values (see Fig. 3). For $N = 50$ data points, the MDL-based criterion inferred the optimal architecture for a number of noise levels, $\sigma = 0.01, 0.05, 0.10, 0.15, 0.20$ and 0.30. Nevertheless, overfitting was again seen for $\sigma = 0.25$ where three hidden neurons where inferred as optimal rather than two. Moreover, similar overfitting was observed even for 'large' amounts of data - 100, 200 and 400 data points.

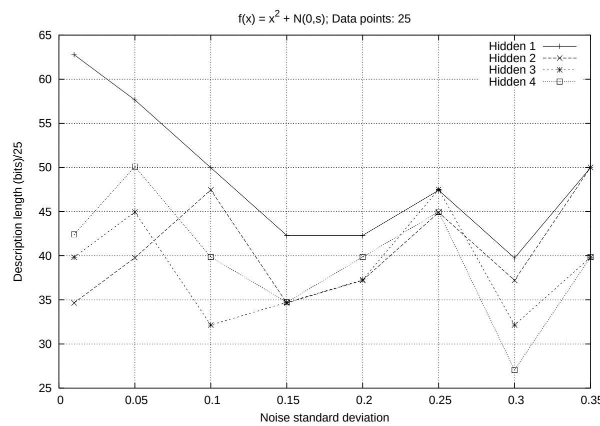

Figure 3. MDL inference of function f_1, $N = 25$

Function f_2 is somewhat more complex than f_1 - a network of five hidden neurons is required to model f_2 accurately. With only 25 data points, MML inferred three hidden neurons as the optimal architecture for the lowest noise level. As more noise was added, MML inferred the simplest network (that of one hidden neuron) as optimal. This is of course a proper thing to do given such a small dataset. For $N = 50$, MML inferred five and three hidden neurons as the optimal architecture when $\sigma = 0.01$ and $\sigma \geq 0.05$ respectively. Increasing the dataset to 100 or more samples, always resulted in an improved inference. Importantly, no overfitting was observed in all datasets considered. In comparison, The MDL-based criterion produced similar results. For small datasets ($N < 100$), MDL often inferred three hidden neurons as optimal. For the lowest noise levels, MDL did infer the optimal architecture albeit rarely. The inference results improved as the amount of data was increased.

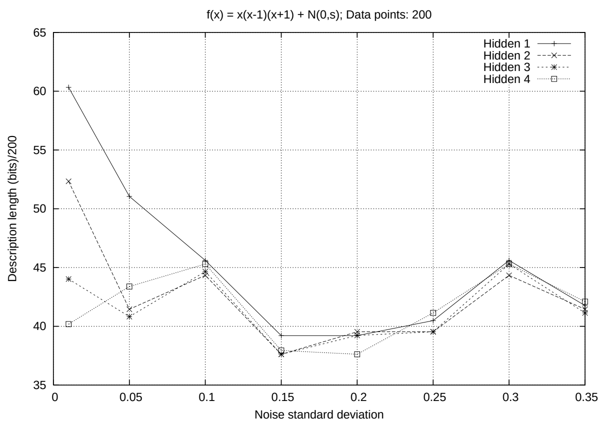

Figure 4. MDL inference of function f_3, $N = 200$

Our criterion did well when inferring function f_3. No overfitting was observed for all datasets considered. The optimal architecture was inferred for low levels of noise. As the noise was increased, MML was more conservative and picked the simpler model as is only reasonable. Conversely, the MDL-based criterion also inferred the optimal architecture for small noise levels. However, the observed results were inconsistent. For example, when $N = 100$ and $\sigma = 0.1$, the MDL-based criterion inferred a four hidden neuron network as more optimal than a two hidden neuron network. Additionally, for $N = 200$ and $\sigma = 0.01, 0.20$, overfitting was observed – MDL inferred four hidden neurons as optimal rather than three (see Fig. 4).

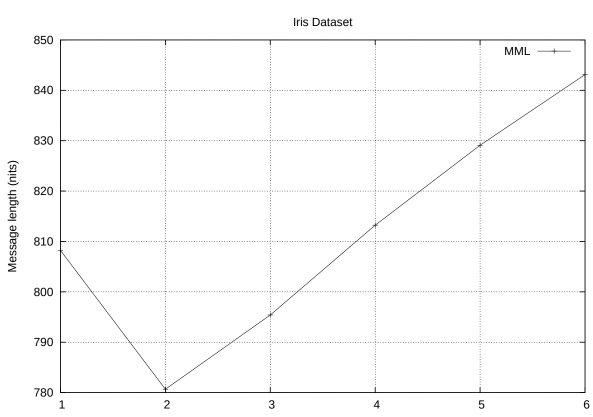

Figure 5. MML inference of the Iris dataset

Finally, we have tested both MML and MDL-based criteria on a real problem: the Iris dataset from the UCI machine learning repository. This is essentially a classification task, where the predicting attribute is a class of an iris plant. The data set comprises three classes of 50 instances each. The first two attributes, namely sepal length and sepal width, were removed due to low class correlation. We assigned a number, ranging from zero to two, to each of the three classes. Alternatively, Engelbrecht [19, 20] uses a network with three outputs where each output corresponds to a distinct class. Both MML (see Figure 5) and MDL-based (see Figure 6) criteria inferred a network with two hidden neurons as optimal. However, the MDL-based criterion inferred that five hidden neurons is preferable to four. The consensus is that a two hidden neuron network is optimal for this dataset [19, 20].

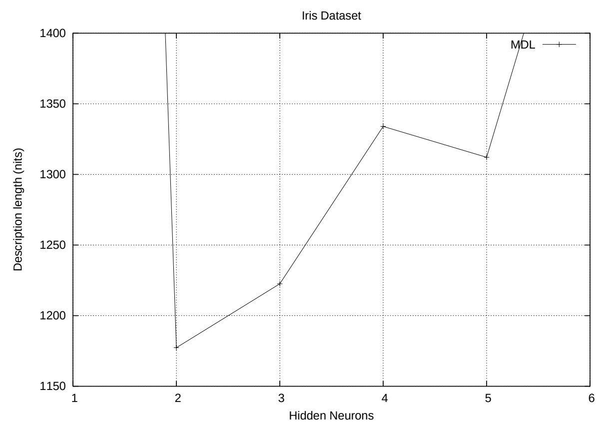

Figure 6. MDL inference of the Iris dataset

6 Conclusion

This paper presented a new approach to optimal architecture selection using the MML87 approximation. We developed a message length formula and used it to cost single hidden layer feedforward neural networks. We introduced a block-diagonal Fisher information approximation that is less prone to numerical instabilities and improves computational efficiency. This approximation could also be used to train and prune neural networks. Moreover, MML87 based neural networks have already been used successfully in musical genre classification [21].

We compared our method to an MDL selection criterion on artificial and real problems. The MML criterion was able to infer the optimal network in the majority of cases. Unlike the MDL-based criterion, our block-diagonal MML approximation exhibited no overfitting for all tested datasets. Interestingly, when a large amount of noise was present in the data, MML did not attempt to model the noise; instead, it inferred a simpler model: commonly one or two hidden neurons. This is a good thing to do because the original function was lost in the noise - inferring a more complex model would be overfitting.

Although our method performed well on several artificial functions and the Iris dataset, a number of possible improvements remain. Future work will include pruning of network inputs, the ability to handle multi-layer feed-forward networks and an improved Fisher information approximation. Additionally, a more comprehensive comparison against newer Bayesian approaches with more testing functions is in development.

Acknowledgments:

The authors thank Mr. Slavisa Garic for his work on Nimrod/G scripts which allowed all tests to run on the CSSE department cluster.

References

[1] D. E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation, *Parallel Data Processing*, 1(8), 1986, 318–362.

[2] M. T. Hagan and M. B. Menhaj, Training feedforward networks with the Marquardt algorithm, *IEEE Transactions on Neural Networks*, 5(6), 1994, 989–993.

[3] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, 2, 1989, 359–366.

[4] T.Y. Kwok and D.Y. Yeung, Constructive algorithms for structure learning in feedforward neural networks for regression problems, *IEEE Transactions on Neural Networks*, 8(3), 1997, 630–645.

[5] R. Reed, Pruning algorithms - a survey, *IEEE Transactions on Neural Networks*, 4(5), 1993, 740–747.

[6] C. S. Wallace and P. R. Freeman, Estimation and inference by compact encoding (with discussion), *Journal of the Royal Statistical Society series B*, 49(3), 1987, 240–265.

[7] G. te Brake, J.N. Kok and P.M.B. Vit'anyi, Model selection for neural networks: Comparing MDL and NIC, *Proc. European Symposium on Artificial Neural Networks*, Brussels, Belgium, 1994, 31–36.

[8] Q. Gao, M. Li and P.M.B. Vit'anyi, Applying MDL to learning best model granularity, *Artificial Intelligence*, 121(1-2), 2000, 1–29.

[9] J.J. Rissanen, Modeling by shortest data description, *Automatica*, 14, 1978, 465–471.

[10] C. S. Wallace and D. M. Boulton, An Information Measure for Classification, *Computer Journal*, 11(2), 1968, 195–209.

[11] C. S. Wallace and D. L. Dowe, Minimum Message Length and Kolmogorov complexity, *Computer Journal*, 42(4), 1999, 270–283.

[12] C. S. Wallace and D. M. Boulton, An invariant Bayes method for point estimation, *Classification Soc. Bull.*, 3(3), 1975, 11–34.

[13] G. E. Farr and C. S. Wallace, The Complexity of Strict Minimum Message Length Inference, *The Computer Journal*, 45(3), 2002, 285–292.

[14] E. Makalic, L. Allison and D. L. Dowe, MML Inference of Single-layer Neural Networks, *Technical Report* 2003/142, Monash University, Australia, 2003.

[15] P. van der Smagt and G. Hirzinger, Why feed-forward networks are in a bad shape, *Proc. Intl. Conf. on Artificial Neural Networks*, Skovde, Sweden, 1998, 159–164.

[16] J.J. Rissanen, A universal prior for integers and estimation by minimum description length, *Annals of Statistics*, 11(2), 1983, 416–431.

[17] J. W. Comley, L. Allison and L. J. Fitzgibbon, Flexible decision trees in a general data-mining environment, *4th Intl. Conf. on Intelligent Data Engineering and Automated Learning (IDEAL)*, Hong Kong, China, 2003.

[18] C.L. Blake and C.J. Merz, UCI Repository of machine learning databases, [http://www.ics.uci.edu/˜mlearn/], 1998.

[19] A. P. Engelbrecht, A New Pruning Heuristic Based on Variance Analysis of Sensitivity Information, *IEEE Transactions on Neural Networks*, 12(6), 2001, 1386–1399.

[20] A. P. Engelbrecht, L. Fletcher and I. Cloete, Variance Analysis of Sensitivity Information for Pruning Feedforward Neural Networks, *IEEE Intl. Joint Conf. on Neural Networks*, Washington DC, USA, 1999.

[21] A. C. Bickerstaffe and E. Makalic, MML Classification of Music Genres, submitted to *Proc. 16th Australian Joint Conf. on Artificial Intelligence*, 2003.