Predicting the Mackey-Glass time series

Approach using Feed Forward neural networks and Takagi-Sugeno fuzzy neural networks.

Tobias Kallehauge

University of Bremen, January 29, 2020

1 Introduction

This report seeks to explore different machine learning methods with the goal of predicting the Mackey-Glass time series [4]. In particular the two methods used are *feed forward neural networks* (FNN's) and Takagi-Sugeno adaptive neuro fuzzy interference systems (TS-ANFIS's / NF networks). The Mackey-Glass (MG) time series is the solution to the time delay differential equation:

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{1}$$

with $x(0) = 1.2$, $x(t) = 0$ for $t < 0$ and $\tau = 17$. The solution is obtained by a forth order Runge-Kutta method and the data is included in the Fuzzy Logic Toolbox Version 2.5 in Matlab [3]. The solution contains 1201 datapoints sampled at integer values of t - see figure 1.

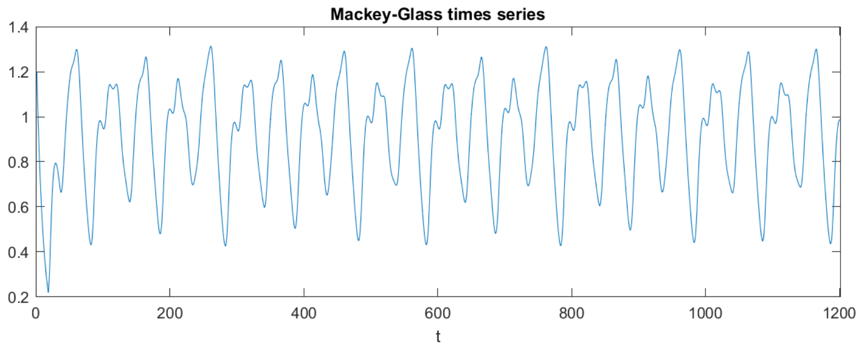

Figure 1

This report will seek to predict the MG time series using a multiple input single output (MISO) model where the input is a number of previous values up till time t and the output is the prediction at time $t + L$ where L is prediction step. In [5, p. 253] it is suggested that the input variables are spaced d samples apart with a total of D input variables. Thus if $\mathbf{X}_I(t)$ and $X_O(t)$ are the input and output at time t, these become:

$$\mathbf{X}_I(t) = \begin{bmatrix} X(t-(D-1)d) & X(t-(D-2)d) & \dots & X(t) \end{bmatrix} \tag{2}$$
$$X_O(t) = X(t+L), \tag{3}$$

where $X(t)$ is the MG time series at time t.
Notice that in figure 1 the first few data points behave differently than the remaining time series. To avoid these values $X(t)$ is only considered for $t > 100$.

The following sections 2 and 3 will explore the theory of the two applied methods and their training and section 4 will describe fitting and analyse the results.

2 Feed forward neural networks

Feed forward neural networks are somewhat of a classical machine learning model in the modern age. Their popularity might be due to their simplicity both in terms of interpretation, notation as well as

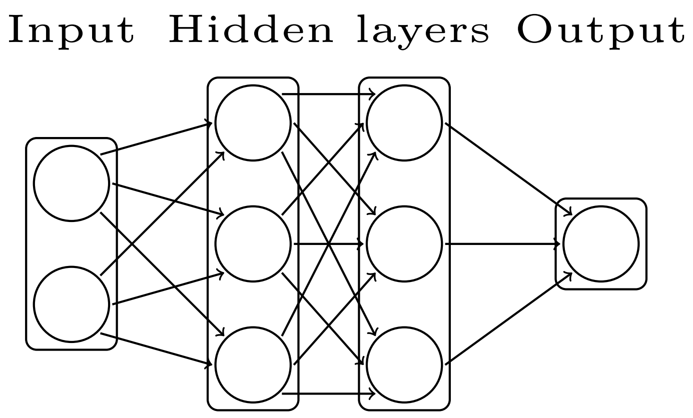

Figure 2: Feed forward neural network with 2 hidden layers.

training algorithm. The idea of FNN's is to invoke the idea of a universal approximator which can be achieved by a series of nested and appropriately chosen non-linear functions [1, p. 230]. In FNN's the non-linear functions are characterised by a matrix multiplication with the input followed by an activation function that is applied elementwise. For a MIMO system with D input values $\mathbf{x} \in \mathbb{R}^D$ an FNN is given by:

$$\mathbf{y} = h^{(M)}\left(\mathbf{W}^{(M)} h^{(M-1)}\left(\mathbf{W}^{(M-1)} \dots h^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x}\right)\dots\right)\right), \tag{4}$$

where $h^{(l)}$ is the so called activation functions possibly dependent on l and $\mathbf{W}^{(l)}$ are the weighs for each layer with dimensions dependent on the layer for $l = 1, \dots, l$. The dimension of the output is set by the dimension of the last weight $\mathbf{W}^{(M)}$ and will be 1 for predicting the MG time series. The activation functions $h^{(l)}$ are generally non-linear in order to invoke the universal approximation property. FNN's are perhaps better explained by its graphical representation as shown in figure 2.

In figure 2 each node (circle) holds a value that is feed forward (hence the name) to each node in the next layer. The edges (arrows) represent this operation which is done by a multiplication with a scalar, possible addition of a bias and finally evaluation through the activation function. Elementwise the oprations from layer to layer is characterised by the following set of equations:

$$a_j^{(l)} = \sum_{i=1}^{D_{l-1}} w_{ji}^{(l)} z_i^{(l)} + b_j^{(l)} \tag{5}$$

$$z_j^{(l)} = h^{(l)}\left(a_j^{(l)}\right) \quad \text{for} \quad j = 1, \dots, D_l, \tag{6}$$

where D_l is the number of nodes in each layer l. Note that $D = D_0$ and $y_j = z_j^M$. The input layer is considered layer 0 while the output layer is layer M. The layers in between - i.e. layer $1, \dots, M-1$, are not directly observed and are therefore often refereed to as the hidden layers.

There are various popular choices for non-linear activation functions. The three explored here will be the hyperbolic tangent tanh, the Sigmoid σ and finally the Rectified Linear Unit (ReLU) f given by:

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{7}$$

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{8}$$

$$f(a) = \max(0, a) \tag{9}$$

These activation functions are non-surjective so if the output of the FNN should span $\mathbb{R}$ the last activation function should be set to the identity $h^{(M)}(a) = a$.

2.1 Training using gradient descent

A FNN as described in (5) and (6) is charactered by the number of layers M, the number of nodes in each layer $D_0, \dots, D_M$, the choices of activation functions $h^{(1)}, \dots, h^{(M)}$, the weights in each layer

$\mathbf{W}^l \in \mathbb{R}^{D_l \times D_{l-1}}$ and the biases in each layer $\mathbf{b} \in \mathbb{R}^{D_l}$. Number of layers, nodes and activation functions are refereed to as hyperparameters and are determined through manual exprimentation while the weights and biasses can be determined through different iterative process. In this section gradient descent methods are explored.

A training set with N samples, $\{\mathbf{x}^p, \mathbf{d}^p\}_{p=1}^N$, is given where $\mathbf{x}^p \in \mathbb{R}^D$ is the input and $\mathbf{d} \in \mathbb{R}^{D_M}$ is the target output. The goal is to minimize the error between target output and output of the FNN under some measure. Here half sum of squared errors (SSE) is used as error measure and the total error S then becomes:

$$S = \frac{1}{2}\sum_{p=1}^{N} e_p = \sum_{p=1}^{N}\sum_{j=1}^{D_M} \left(y_j^p - d_j^p\right)^2 , \tag{10}$$

where y_j^p is value of node j of the FNN with input $\mathbf{x}^p$. Minimizing S with respect to the weights and biasses is not possible in closed form due to the non-linearities of the network so instead an iterative method is required. Gradient descent (GD) (or steepest descent) is one such method. If $\boldsymbol{\theta}$ represents a vector containing all the parameters (weights and biasses) then gradient descent for minimising S in (10) contains steps [1, p. 240]:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \eta \nabla S(\boldsymbol{\theta}^{(\tau)}), \tag{11}$$

where $\nabla S(\boldsymbol{\theta}^{(\tau)})$ is the gradient with respect to $\boldsymbol{\theta}^{(\tau)}$ and η is the so called learning rate. In [1, p. 240] it is argued that gradient descent is not well suited for optimizing neural networks such as FNN's. Reasons include not handling redundancy in data well and not being able to escape local minima of the objective function. Another method that can avoid these problems is sequential gradient descent (SQD) [1, p. 241][1]. SQD is similar to GD but instead of updating the parameters with respect to the entire training set the update is done with respect to each sample at a time:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \eta \nabla e_p(\boldsymbol{\theta}^{(\tau)}), \tag{12}$$

where $\nabla e_p(\boldsymbol{\theta}^{(\tau)})$ is the gradient with respect to $\theta^{(\tau)}$. One way to realise SQD is by cycling through the training data sequentially $p = 1, 2, \dots, M, 1, 2, \dots, M, \dots$. Each cycle through the training data is referred to as an epoch. GD and SQD are extremes in the sense that the parameters are updated with respect to either all or one training example at the time. In between lies minibatch methods where the parameters are updated with respect to a small batch of training examples a time. For this report SQD will be used, but the developed software (see section A) handles batches of any size. It remains to specify how to compute $\nabla e_p(\boldsymbol{\theta}^{(\tau)})$. This can be be done in a systematic way known as backpropagation.

2.2 Backpropagation

The gradient $\nabla e_p(\boldsymbol{\theta}^{(\tau)})$ is obtained using the generalised chain rule for partial derivatives which states that terms or factors contributing to a particular partial derivative should be summed up and each term or factor is computed using the usual chain rule. The gradient is computed starting from the last layer and propagating backward through the layers adding up relevant terms - hence backpropagation. For a weight $w_{ji}^{(M)}$ positioned in the last layer M, no summation is required since e_p only depends on $w_{ji}^{(M)}$ through layer j and the gradient is:

$$\frac{\partial e_p}{\partial w_{ji}^{(M)}} = \frac{\partial e_p}{\partial y_j}\frac{\partial y_j}{\partial a_j^{(M)}}\frac{\partial a_j^{(M)}}{\partial w_{ji}^{(M)}} = \underbrace{(y_j - d_j)h'(a_j^{(M)})}_{\delta_j^{(M)}} z_i^{(M-1)} = \delta_j^{(M)} z_i^{(M-1)}, \tag{13}$$

where the subscript p and possibility of different activation functions are ignored for readability. The quantity $\delta_j^{(M)}$ has the interpretation of the error derivative with respect to layer M and greatly simplifies the notation of the derivative as will soon become evident.
For a weight $w_{ji}^{(M-1)}$ positioned in the second last layer e_p now depends on this through all the

[1]Sequential gradient descent can also be refereed to as stochastic gradient descent or on-line gradient decent.

nodes in the last layer and node j in the second last layer. A summation over all the nodes in the last layer is then required:

$$\begin{aligned}
\frac{\partial e_p}{\partial w_{ji}^{(M-1)}} &= \sum_{k=1}^{D_M} \frac{\partial e_p}{\partial y_k} \frac{\partial y_k}{\partial a_k^{(M)}} \frac{\partial a_k^{(M)}}{\partial z_j^{(M-1)}} \frac{\partial z_j^{(M-1)}}{\partial a_j^{(M-1)}} \frac{\partial a_j^{(M-1)}}{\partial w_{ji}^{(M-1)}} \\
&= \sum_{k=1}^{D_M} \underbrace{(y_k - d_k) h'(a_k^{(M)})}_{\delta_k^{(M)}} w_{kj}^{(M)} h'(a_j^{(M-1)}) z_i^{(M-2)} \\
&= \underbrace{\sum_{k=1}^{D_M} \delta_k^{(M)} w_{kj}^{(M)} h'(a_j^{(M-1)})}_{\delta_j^{(M-1)}} z_i^{(M-2)} = \delta_j^{(M-1)} z_i^{(M-2)},
\end{aligned} \tag{14}$$

where the next error derivative term $\delta_j^{(M-1)}$ is introduced. The pattern of backpropagation should be realised with the expansion of the gradient with respect to a weight in the third last layer $w_{ji}^{(M-2)}$. e_p now depends on $w_{ji}^{(M-2)}$ through all the nodes in the last layer and second last layer so two summations are required:

$$\begin{aligned}
\frac{\partial e_p}{\partial w_{ji}^{(M-2)}} &= \sum_{k=1}^{D_M} \frac{\partial e_p}{\partial y_k} \frac{\partial y_k}{\partial a_k^{(M)}} \sum_{l=1}^{D_{M-1}} \frac{\partial a_k^{(M)}}{\partial z_l^{(M-1)}} \frac{\partial z_l^{(M-1)}}{\partial a_l^{(M-1)}} \frac{\partial a_l^{(M-1)}}{\partial z_j^{(M-2)}} \frac{\partial z_j^{(M-2)}}{\partial a_j^{(M-2)}} \frac{\partial a_j^{(M-2)}}{\partial w_{ji}^{(M-2)}} \\
&= \sum_{k=1}^{D_M} \underbrace{(y_k - d_k) h'(a_k^{(M)})}_{\delta_k^{(M)}} \sum_{l=1}^{D_{M-1}} w_{kl}^{(M)} h'(a_l^{(M-1)}) w_{lj}^{(M-1)} h'(a_j^{(M-2)}) z_i^{(M-3)} \\
&= \sum_{k=1}^{D_M} \delta_k^{(M)} \sum_{l=1}^{D_{M-1}} w_{kl}^{(M)} h'(a_l^{(M-1)}) w_{lj}^{(M-1)} h'(a_j^{(M-2)}) z_i^{(M-3)} \quad \text{(exchange the order of sums)} \\
&= \sum_{l=1}^{D_{M-1}} \underbrace{\sum_{k=1}^{D_M} \delta_k^{(M)} w_{kl}^{(M)} h'(a_l^{(M-1)})}_{\delta_l^{(M-1)}} w_{lj}^{(M-1)} h'(a_j^{(M-2)}) z_i^{(M-3)} \\
&= \underbrace{\sum_{l=1}^{D_{M-1}} \delta_l^{(M-1)} w_{lj}^{(M-1)} h'(a_j^{(M-2)})}_{\delta_j^{(M-2)}} z_i^{(M-2)} = \delta_j^{(M-2)} z_i^{(M-3)}
\end{aligned} \tag{15}$$

After some rearranging, the general form of the error derivative is then:

$$\delta_j^{(l)} = \begin{cases} h'(a_j^{(M)})(y_j - d_j) & \text{for} \quad l = M \\ h'(a_j^{(l)}) \sum_{k=1}^{D_l} \delta_k^{(l+1)} w_{kj}^{(l)} & \text{for} \quad l = M-1, M-2, \dots, 1 \end{cases}, \tag{16}$$

and the general form of the derivative with respect to the weights are simply:

$$\frac{\partial e_p}{\partial w_{ji}^{l}} = \delta_j^{(l)} z_i^{(l-1)} \tag{17}$$

The gradient with respect to the biasses is derived similarly and is:

$$\frac{\partial e_p}{\partial b_j^{l}} = \delta_j^{(l)} \tag{18}$$

The operations in equation (16)-(18) are easily carried out on vector form: The vectorised error derivative is:

$$\boldsymbol{\delta}^{(l)} = \begin{cases} h'(\mathbf{a}^{(M)}) \odot (\mathbf{y} - \mathbf{d}) & \text{for} \quad l = M \\ h'(\mathbf{a}^{(l)}) \odot \left(\left(\mathbf{W}^{(l)}\right)^T \boldsymbol{\delta}^{(l+1)} \right) & \text{for} \quad l = M-1, M-1 \dots, 1 \end{cases}, \tag{19}$$

where $\mathbf{W}^{(l)}$ has dimension $D_l \times D_{l-1}$, the error terms $\boldsymbol{\delta}^{(l)}$, activations $\mathbf{a}^{(l)}$ has dimension D_l, $\odot$ is the Hadamard product which carries out elementwise multiplication and T is the transpose. The derivatives are:

$$\frac{\partial e_p}{\partial \mathbf{W}^l} = \boldsymbol{\delta}^{(l)} \left(\mathbf{z}^{(l-1)}\right)^T \tag{20}$$

$$\frac{\partial e_p}{\partial \mathbf{b}^l} = \boldsymbol{\delta}^{(l)} \tag{21}$$

In practice, backpropagation is carried out by initially computing the network output for a given input (forward propagation) while saving the relevant variables, then evaluating the gradients using (19)-(21) starting from layer M and backward to layer 1. Initializing the parameters can be done in various ways but a common practise is to initialise the weights randomly as $w_{ji}^{(l)} \sim \text{uniform}(0,1)$ and $b_j^{(l)} = 0$ which will be used in the training in section 4.

It only remains to specify the derivatives of the activation functions which are [1]:

$$\tanh'(a) = 1 - \tanh(a)^2 \tag{22}$$

$$\sigma'(a) = \frac{e^{-a}}{(1+e^{-a})^2} \tag{23}$$

$$f'(a) = \begin{cases} 0 & \text{for} \quad a \leq 0 \\ 1 & \text{for} \quad a > 0 \end{cases} \quad \text{(Ignoring the non-differentiable point at 0)} \tag{24}$$

3 Takagi-Sugeno adaptive neuro fuzzy interference system

This section will describe the theory behind the TS type fuzzy model, the adaptive TS-ANFIS and the training of this. Derivations are not shown - see [5, p. 234 - 237] for further details.

3.1 TS fuzzy models

A TS fuzzy model is based on a number of rules M. In the MISO case each rule l is on the form:

$$R^l: \quad \text{IF } x_1 \text{ is } G_1^l \text{ and } \dots \text{ and } x_D \text{ is } G_D^l \text{ THEN } y_{\text{TS}}^l = \theta_0^l + \theta_1^l x_1 + \dots + \theta_D^l x_D, \tag{25}$$

where D is the number of inputs, G_i^l is a fuzzy set, $\theta_i^l \in \mathbb{R}$ are parameters and y_{TS}^l is the crisp output for rule l for $l = 1, \dots M$. The fuzzy sets are completely characterised by their membership functions $\mu_{G_i^l} : U \to [0,1]$. U is refereed to as the universe of discourse and is simply $U = \mathbb{R}$ here. As suggested by [5, pp. 162–163], the membership functions will be Gaussian kernels:

$$\mu_{G_i^l}(x) = \exp\left(-\left(\frac{x - c_i^l}{\sigma_i^l}\right)^2\right), \tag{26}$$

where c_i^l and σ_i^l are real valued parameters associated with the fuzzy set G_i^l. A TS fuzzy model is thus characterised by hyper parameters (M, D) as well as parameters $\mathbf{c} \in \mathbb{R}^{M\times D}$, $\boldsymbol{\sigma} \in \mathbb{R}^{M\times D}$ and $\boldsymbol{\theta} \in \mathbb{R}^{M\times D+1}$.

Given an input $\mathbf{x} \in \mathbb{R}^D$ inference for the TS fuzzy model is done in the following way. Initially the degree of membership for each input and associated fuzzy set is evaluated by computing $\mu_{G_i^l}(x_i)$ for $i = 1, \dots, D$ and $l = 1, \dots, M$. Then, in order to determine how well a particular rule is fulfilled, the rule degree of fulfilment β^l for each rule $l = 1, \dots, M$ is computed as:[2]

$$\beta^l = \prod_{i=1}^{D} \mu_{G_i^l}(x_i) = \prod_{i=1}^{D} \exp\left(-\left(\frac{x_i - c_i^l}{\sigma_i^l}\right)^2\right) = \exp\left(-\sum_{i=1}^{D}\left(\frac{x_i - c_i^l}{\sigma_i^l}\right)^2\right) \tag{27}$$

Now y_{TS}^l can be evaluated as shown in (25) for each rule. The crisp output is finally computed as the dot product between the crisp rule outputs and the normalised degree of fulfilment outputs:

$$y_0 = \frac{\sum_{l=1}^{M} \beta^l y_{\text{TS}}^l}{\sum_{l=1}^{M} \beta^l} = \sum_{l=1}^{M} \gamma^l y_{\text{TS}}^l, \quad \text{where} \quad \gamma^l = \frac{\beta^l}{\sum_{k=1}^{M} \beta^k} \tag{28}$$

is the normalised degree of fulfilment.

[2] Instead of the product operator the AND operator can be used which takes $\min_i \mu_{G_i^l}(x_i)$. But due to the non-differentiability of the AND operator it is not well suited for a TS-ANFIS system.

3.2 Training TS-ANFIS's

Given a training set, the parameters for a TS fuzzy model can be determined in an automated way using the Rules Generation Algorithm [5, pp. 157–169] although another approach is possible where the parameters are learned in an adaptive way similar to how the FNN is trained using SQD and backpropagation. When trained this way the TS fuzzy model is refereed to as a TS-ANFIS or neuro fuzzy (NF) netowrk.

The only difference in the training algorithm of a TS-ANFIS compared to a FNN is the computation of the gradient due to the different architecture of the network. Similarly to section 2.1, a training set, $\{\mathbf{x}^p, d^p\}_{p=1}^N$ for $\mathbf{x}^p \in \mathbb{R}^D$ and $d^p \in \mathbb{R}$, is given with the goal of minimising the cost S given by the objective function here being:

$$S = \frac{1}{2}\sum_{p=1}^{N} e^p = \frac{1}{2}\sum_{p=1}^{N} \left(f_{\text{TS}}(\mathbf{x}^p) - d^p\right)^2, \tag{29}$$

where f_{TS} will denotes the function that represent the TS fuzzy model such that $f_{\text{TS}} : \mathbb{R}^D \to \mathbb{R}$ and $y_0 = f_{\text{TS}}(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^D$.
The gradients used in SQD under the objective function (29) are derived in [5, pp. 234–237]. Let $y_{\text{TS}}^{l,p}$ and $\gamma^{l,p}$ denote the quantities computed in the TS fuzzy model with input $\mathbf{x}^p$. Furthermore denote $f^p \equiv f_{\text{TS}}(x^p)$ and $A^{l,p} \equiv (y_{TS}^{l,p} - f^p)(f^p - d^p)$. The gradients are then given elementwise as [5, p. 236]:

$$\frac{\partial e^p}{\partial \theta_0^l} = (f^p - d^p)\gamma^{l,p} \tag{30}$$

$$\frac{\partial e^p}{\partial \theta_i^l} = (f^p - d^p)\gamma^{l,p}x_i^p \quad \text{for} \quad 2 \le i \le D \tag{31}$$

$$\frac{\partial e^p}{\partial c_i^l} = 2A^{l,p}\gamma^p \frac{x_i^p - c_i^l}{(\sigma_i^l)^2} \tag{32}$$

$$\frac{\partial e^p}{\partial \sigma_i^l} = 2A^{l,p}\gamma^p \frac{(x_i^p - c_i^l)^2}{(\sigma_i^l)^3}, \tag{33}$$

for $i = 1, \ldots, D$ and $l = 1, \ldots, M$. Equations (30)-(33) are carried out on vector form using elementwise multiplication and division.

4 Training and performance analysis

As described in the introduction the input to the used methods will consist of D previous values spaced d samples apart and the prediction will be given L samples into the future. While the choices for D, d and L are definitely subject to investigation, this report will simply follow the suggestion by [5, p. 158] and use $D = 4$ and $d = L = 6$. Using (2) and (3) the input/output sets can be constructed and from the 1201 samples in the MG time series 1177 input/output sets (without missing values) are made. Thus $\mathbf{X}_I(t)$ and $X_O(t)$ is constructed for $t = 1, \ldots, 1177$ which is divided into the following subsets:

- $t = 1, \ldots, 100$: Discarded due to irregular behaviour.
- $t = 101, \ldots, 300$: Training data
- $t = 301, \ldots, 700$: Validation data
- $t = 701, \ldots, 1177$: Test data

The training data is used directly to train the networks - i.e. to compute the gradients used in SQD. The remaining data is split into validation data and test data as a machine learning practice. The validation data, technically a part of the training data, is used for evaluating the performance of the networks when designing them through choice of hyperparameters and the test data is used as a final performance evaluation after the networks have been designed and fitted. The reason for this split is to avoid overfitting to the test data by network design. See [2, pp. 110–122] for further details about training/validation/test sets. The procedure for training the networks will then be: 1)

Fit the networks using the training data with many different combinations of hyperparameters. 2) Use the validation data to decide which hyperparameters should be used by choosing the settings with the lowest error - here the mean squared error (MSE) of the trained networks evaluated on the validation data. 3) Analyse the performance of the trained networks using the test data again using the MSE.

4.1 Hyperparameter tests

Before the hyperparameters are tests the number of epochs for the SQD scheme is discussed. Performing SQD with one epoch on the training data took only 28.1ms and 4.9ms respectively for the FNN and TS-ANFIS[3]. It is therefore argued that the number of epochs should simply be set high enough to ensure convergence. In figure 3 and 4 examples of the training convergence for the two methods is shown. The settings for the networks was: learning rate $\eta = 10^{-2}$ for both methods, number of layers $M_{\text{FNN}} = 3$ for the FNN, activation function $h = \text{ReLU}$ for the FNN[4] and the number of rules $M_{\text{TS}} = 5$ for the TS-ANFIS. These settings will be refereed to as the defaults settings and will be used unless otherwise mentioned in the following sections.

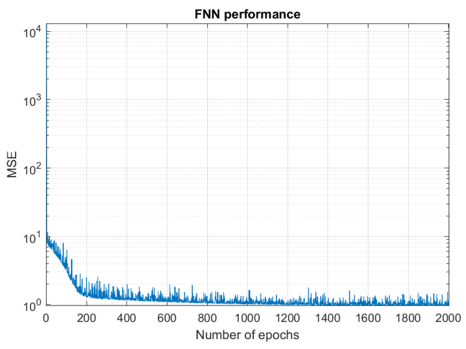

Figure 3: Example of training error after each epoch for the FNN.

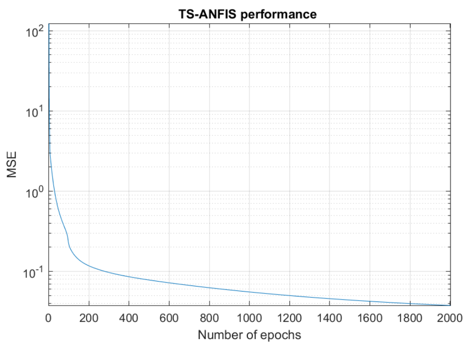

Figure 4: Example of training error after each epoch for the TS-ANFIS.

The errors shown in figure 3 and 4 suggest that the largest performance gain is seen after the first 200 epochs for both methods and only marginal improvement is seen after this (note that the MSE is shown on a logarithmic scale). Other examples was performed and these showed similar behaviour. It is therefore decided that for the hyperparameter tests $N_{\text{epochs}} = 500$ will be sufficient for both methods. In section 4.2, where the networks are trained with the found hyperparameters other stopping criteria will be used.

4.1.1 FNN

For the FNN serveral hyperparameters should be set. For a basic FNN the number of layers M, choice of activation for each layer and the number of nodes in each layer D_l are the hyperparameters. Additionally a suitable the learning rate η should be found. In order to limit the number of possible combinations of hyper parameters the following tests will have the same number of nodes in each layer $D_0 = D_1 = \cdots = D_{M-1} = 4$ except for the last where $D_M = 1$. The activation functions h will be the same in each layer except the last layer where h is the identity. Furthermore interaction between hyperparameters is non considered - e.g. if 3 layers are found to be optimal with $h =$ Sigmoid it will not also be investigated if 3 layers are optimal with $h = \text{ReLU}$. These delimitations are chosen somewhat arbitrarily and under them the optimal hyper parameters might not be found, but it limits the number of tests that have to be performed.

Thus 2 separate hyper parameter tests varying the number of layers and activation functions as well as the learning rate test are performed. When testing for one setting the other settings will be the default settings as introduced earlier with 500 epochs in the SQD. For each setting the FNN was fitted 100 times where one test corresponds to initialising the FNN parameters randomly then training using SQD for 500 epochs. The MSE was afterwards evaluated with the fitted models using

[3]Emperical results on a computing with an Intel® Core™ i7-4720HQ processor.

[4]The activation function in the last layer was the identity function

the validation data. Table 1 and figure 5 shows the results.

FNN Hyperparameter tests							
		Number of layers					
	N_{layers}	**1**	**2**	**3**	**4**	**5**	**6**
	min	$8.89 \cdot 10^{-3}$	$8.88 \cdot 10^{-3}$	$6.89 \cdot 10^{-4}$	$\underline{5.51 \cdot 10^{-4}}$	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$
MSE	median	$\underline{9.00 \cdot 10^{-3}}$	$9.08 \cdot 10^{-3}$	$9.01 \cdot 10^{-3}$	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$
	max	$\underline{1.02 \cdot 10^{-2}}$	$1.24 \cdot 10^{-2}$	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$
		Activation function					
	h	**tanh**		**Sigmoid**		**ReLU**	
	min	$1.15 \cdot 10^{-2}$		$1.14 \cdot 10^{-2}$		$\underline{6.88 \cdot 10^{-4}}$	
MSE	median	$5.02 \cdot 10^{-2}$		$5.02 \cdot 10^{-2}$		$\underline{9.01 \cdot 10^{-3}}$	
	max	$5.032 \cdot 10^{-2}$		$\underline{5.030 \cdot 10^{-2}}$		$9.13 \cdot 10^{-1}$	
		Learning rate					
	η	$\mathbf{3.3 \cdot 10^{-1}}$	$\mathbf{3.7 \cdot 10^{-2}}$	$\mathbf{1.2 \cdot 10^{-2}}$	$\mathbf{4.1 \cdot 10^{-3}}$	$\mathbf{4.6 \cdot 10^{-4}}$	$\mathbf{1.7 \cdot 10^{-5}}$
	min	$9.13 \cdot 10^{-1}$	$\underline{3.38 \cdot 10^{-4}}$	$5.02 \cdot 10^{-4}$	$1.15 \cdot 10^{-3}$	$8.89 \cdot 10^{-3}$	$4.65 \cdot 10^{-2}$
MSE	median	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$	$\underline{8.97 \cdot 10^{-3}}$	$9.29 \cdot 10^{-3}$	$1.09 \cdot 10^{-2}$	$9.28 \cdot 10^{-2}$
	max	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$	$9.13 \cdot 10^{-1}$	$\underline{4.17 \cdot 10^{-2}}$	$1.48 \cdot 10^{-1}$

Table 1: MSE on validation data under various hyper parameter settings. The minimum, median and maximum values are shown from a total of 100 test for each setting. The lowest MSE within each settings and each statistic have been underlined. The learning rate settings follows a negative exponential function $\eta_i = a^{-i}$ and $a = 3$ have been used here. A total of 10 settings with the learning rate have been tested but only 6 of them are shown here - see figure 5 for further results. MSE $= 9.13 \cdot 10^{-3}$ occurs often in the table. This value corresponds to the FNN learning the zero function - i.e. 0 everywhere for all inputs.

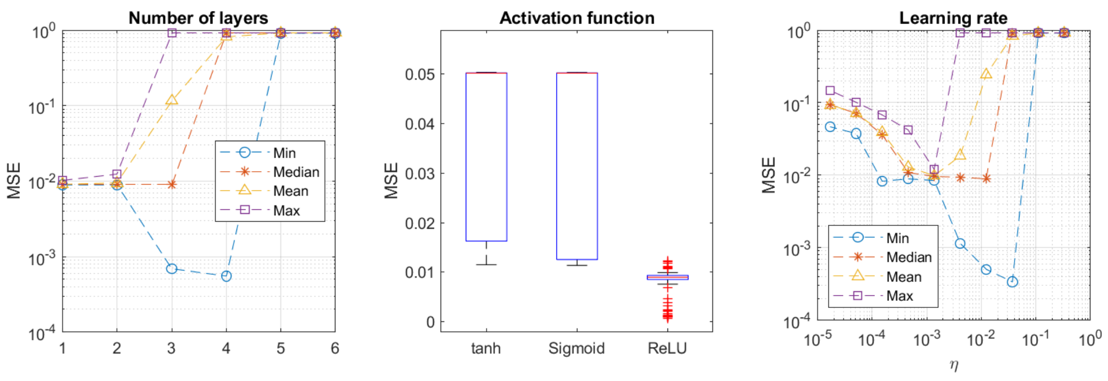

Figure 5: Graphical representation of the results represented in table 1. A boxplot is shown for the activation functions where the red vertical is the median and the box marks the 25th and 75th percentiles.

Based on the results in table 1 and figure 5 the hyperparameters for the FNN are chosen. In the test with number of layers it is observed that increasing N_{layers} can improve the performance but not necessarily. The minimum MSE should be of primary interest though and therefore $N_{\text{layers}} = 4$ is chosen.
For activation function the ReLU performed significantly better in most tests and is therefore chosen. For learning rate $\eta = 3.7 \cdot 10^{-2}$ had the best single outcome but it could be argued that with more epochs the smaller learning rates could have performed just as well. Thus there is no obvious choice for learning rate. Due to the issue of to few epochs the learning rate is chosen to be smaller than the one with the minimum MSE and it is chosen that $\eta = 4.1 \cdot 10^{-3}$.

4.1.2 TS-ANFIS

The TS-ANFIS is more limited in the network design and only the number of rules M are considered a hyperparameter here. Additionally a suitable learning rate η for the SQD for training the TS-ANFIS should also be found. The tests are carried out similarly to the hyper parameter test for the FNN. Table 2 and figure 6 shows the results.

TS-ANFIS Hyperparameter tests						
		Number of rules				
	N_{layers}	**1**	**2**	**3**	**4**	**5**
	min	$8.89 \cdot 10^{-3}$	$1.70 \cdot 10^{-4}$	$1.22 \cdot 10^{-4}$	$\underline{5.64 \cdot 10^{-5}}$	$9.37 \cdot 10^{-5}$
MSE	median	$8.96 \cdot 10^{-3}$	$3.84 \cdot 10^{-4}$	$3.03 \cdot 10^{-4}$	$2.57 \cdot 10^{-4}$	$2.18 \cdot 10^{-4}$
	max	$9.88 \cdot 10^{-3}$	$9.94 \cdot 10^{-3}$	$9.60 \cdot 10^{-3}$	$9.36 \cdot 10^{-3}$	$9.06 \cdot 10^{-3}$
	N_{layers}	**6**	**7**	**8**	**9**	**10**
	min	$9.15 \cdot 10^{-5}$	$9.08 \cdot 10^{-5}$	$8.40 \cdot 10^{-5}$	$9.73 \cdot 10^{-5}$	$8.57 \cdot 10^{-5}$
MSE	median	$2.17 \cdot 10^{-4}$	$2.13 \cdot 10^{-4}$	$1.92 \cdot 10^{-4}$	$1.83 \cdot 10^{-4}$	$\underline{1.81 \cdot 10^{-4}}$
	max	$1.11 \cdot 10^{-2}$	$1.54 \cdot 10^{-3}$	$\underline{1.39 \cdot 10^{-3}}$	$3.66 \cdot 10^{-3}$	$1.44 \cdot 10^{-3}$
		Learning rate				
	η	$\mathbf{8.0 \cdot 10^{-1}}$	$\mathbf{8.6 \cdot 10^{-2}}$	$\mathbf{9.2 \cdot 10^{-3}}$	$\mathbf{9.9 \cdot 10^{-4}}$	$\mathbf{1.1 \cdot 10^{-4}}$
	min	$1.13 \cdot 10^{-4}$	$4.49 \cdot 10^{-5}$	$9.33 \cdot 10^{-5}$	$2.80 \cdot 10^{-4}$	$1.88 \cdot 10^{-3}$
MSE	median	$1.38 \cdot 10^{-3}$	$1.14 \cdot 10^{-4}$	$2.40 \cdot 10^{-4}$	$1.05 \cdot 10^{-3}$	$9.05 \cdot 10^{-3}$
	max	$2.06 \cdot 10^{0}$	$9.72 \cdot 10^{-3}$	$3.46 \cdot 10^{-3}$	$1.50 \cdot 10^{-2}$	$5.39 \cdot 10^{-2}$

Table 2: MSE on validation date similar to table 1 where the statistics are based on 100 tests for each setting. The learning rate has been set to $\eta_i = a^{-i}$ for $i = 1, \ldots, 50$ and $a = 1.25$. Only a few of the learning rate parameters have been shown here. See figure 6 graphical representation of the results.

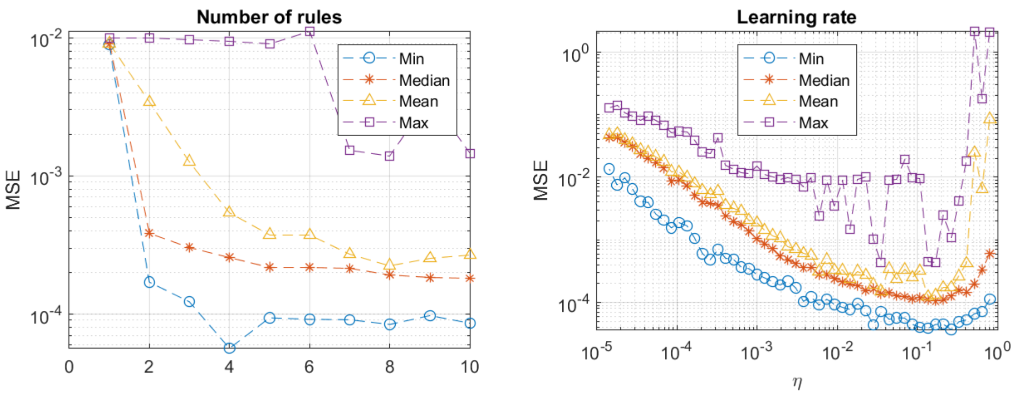

Figure 6: Graphical representation of the results shown in table 2.

Based on the results shown in table 2 and figure 6 the hyper parameters are chosen.
For the number of rules $M = 4$ had the best single performance, but $M = 8$ seems to train the best network on average. Since $M = 8$ seems easier to train it might be possible to get a better performing network than with $M = 4$. Therefore $M = 8$ is chosen.
For learning rate $\eta \approx 10^{-1}$ gives the overall best performance. As argued with FNN though the reason for higher η performing better might caused by too few epochs in the SQD. Therefore $\eta = 10^{-2}$ is also chosen as the learning rate since this value seems to perform well on average as well.

4.2 Evaluation on test data

Finally the two networks are trained with the hyperparameters and learning rate found in section 4.1. To summarize the networks are designed in the following way:

- **Input/Output** $D = 4$ inputs and 1 output for every time t with time between input d and prediction step L both 6.

- **FNN** with $M = 4$ layers and ReLU activation in the first 3 layers and the identity in the last. Layer 1-3 are of width $D_l = 4$ and the last layer of width $D_4 = 1$.
- **TS-ANFIS** with $M = 8$ rules.
- **Learning rate** $\eta = 4.1 \cdot 10^{-3}$ and $\eta = 10^{-2}$ for the FNN and TS-ANFIS respectively.

The training under these settings was done again using SQD but now with a stop criteria rather than a fixed amount of epochs. The stop criteria was to count for how many epochs the training error (SSE on the training data) did not fall and then stop when this count exceeds a certain number N_{stop}. The parameters with the lowest training error for all epochs is then returned as the final parameters. For the training of both networks $N_{\text{stop}} = 5000$ was used - i.e. the SQD stops when the training error has not fallen for 5000 epochs. After fitting both networks with the found hyper parameters they were evaluated on the test data - see table 3 and figure 7 for the results[5].

Performance analysis on test data

Network	**FNN**	**TS-ANFIS**
MSE	$2.50 \cdot 10^{-4}$	$6.14 \cdot 10^{-6}$
RMSE	$1.58 \cdot 10^{-2}$	$2.48 \cdot 10^{-3}$
MAE	$1.18 \cdot 10^{-2}$	$1.85 \cdot 10^{-3}$

Table 3: Different error measurements between the MG data series and its predicted values from the FNN and TS-ANIFS. The mean squared error (MSE), root mean squared error (RMSE) and mean absolute error (MAE) is displayed. The test dataset contains 477 data points.

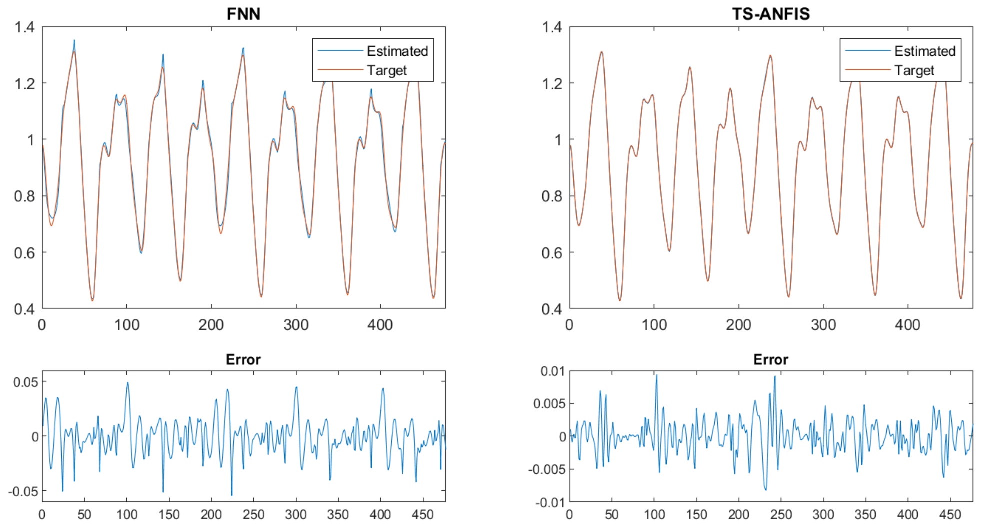

Figure 7: Prediction of the MG data series with FNN on the left and TS-ANFIS on the right. The test data is used and the error of each method is shown below.

Based on the results in table 3 and figure 7 it is quickly realised that the TS-ANFIS performs better than the FNN on the test data with errors at least an order of magnitude lower depending on the error measurement. For one epoch the TS-ANFIS was faster to train than the FNN although it took significantly more epochs before the stop criteria was met for the TS-ANFIS. The training scheme through SQD converged slowly so other optimization methods such as moment based gradient descent optimization could possibly be used to lower training time. In conclusion the TS-ANFIS is the recommended method for prediction the MG data series compared to the FNN although other types of networks or methods could also be considered.

[5]In practise for the training for each network type was done in the following way: 1) Randomly initialise 10 networks and train each of them for N_{epochs} epochs or till stopping criteria $N_{\text{stop}} = 5000$ is met. 2) Of those 10 networks chose the one with the lowest training error. 3) Continue to run SQD with this network now only stopping when the stopping criteria N_{stop} is met. The reason for training 10 different networks in step 1 was to avoid getting stuck in a "bad local minima" - e.g. learning the zero-function. In step 1, $N_{\text{epochs}} = 500,000$ for the TS-ANIFS and $N_{\text{epochs}} = 50,000$ for the FNN, where the latter is fewer epochs since it was slower to train the FNN. It was not recorded for how many epochs step 3 was done for each network, but is was less than $1,000,000$ for the TS-ANFIS and less then $50,000$ for the FNN before the stop criteria was met.

References

[1] Bishop, Christopher M. *Pattern Recognition and Machine Learning.* Ed. by Jordan, Michael, Kleinberg, Jon, and Scholkopf, Bernhard. Information Science and Statistics. Springer, 2006. ISBN: 978-0387-31073-2.

[2] Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning.* MIT Press, 2016. URL: `http://www.deeplearningbook.org`.

[3] Mathworks. *Fuzzy Logic Toolbox.* Visited 7/1-2020. URL: `https://se.mathworks.com/products/fuzzy-logic.html`.

[4] Mathworks. *Predict Chaotic Time-Series using ANFIS.* Visited 7/1-2020. URL: `https://se.mathworks.com/help/fuzzy/predict-chaotic-time-series-code.html`.

[5] Palit, Ajoy K and Popovic, Dobrivoje. *Computational Intelligence in Time Series Forecasting.* Springer, 2005.

A Script references

The scripts used in this report is build on Matlab version `9.6.0.1174912 (R2019a) Update 5`. No toolboxes are required. The scripts are compiled as a GitHub repository at:

`https://github.com/TobiasKallehauge/time_series_forecasting`,

where a description of the scripts can also be found.