Simulation meta-modeling of complex industrial production systems using neural networks

Axel Thor Asthorsson
Simulation meta-modeling of complex industrial production systems using neural networks

Submitted by Axel Thor Asthorsson to the University of Skövde as dissertation towards the degree of B.Sc. by examination and dissertation in the School of Humanities and Informatics.
This final year project was supervised by Anders Dahlbom.

2006-09-01

I hereby certify that all material in this dissertation which is not my own work has been identified and that no work is included for which a degree has already been conferred on me.

Signature: ____________________________________________
Simulation meta-modeling of complex industrial production systems using neural networks

Axel Thor Asthorsson

Abstract

Simulations are widely used for analysis and design of complex systems. Real-world complex systems are often too complex to be expressed with tractable mathematical formulations. Therefore simulations are often used instead of mathematical formulations because of their flexibility and ability to model real-world complex systems in some detail. Simulation models can often be complex and slow which lead to the development of simulation meta-models that are simpler and faster models of complex simulation models. Artificial neural networks (ANNs) have been studied for use as simulation meta-models with different results. This final year project further studies the use of ANNs as simulation meta-models by comparing the predictability of five different neural network architectures: feed-forward-, generalized feed-forward-, modular-, radial basis- and Elman artificial neural networks where the underlying simulation is of complex production system. The results where that all architectures gave acceptable results even though it can be said that Elman- and feed-forward ANNs performed the best of the tests conducted here. The difference in accuracy and generalization was considerably small.

Keywords: Artificial neural networks, meta-models, simulation
1 Introduction

Simulations are widely used for analysis and design of complex systems. Real-world complex systems are often too complex to be expressed with tractable mathematical formulations (April et al., 2001). Therefore simulations are often used instead of mathematical formulations because of their flexibility and ability to model real-world complex systems in some detail (Alam, F.M. et al, 2004).

It is often valuable to optimize a simulation to get the desired output from it. By optimizing a simulation the simulation’s parameters are changed to get the preferred behavior. An example of simulation’s parameters is for example buffer sizes in a production line of a large complex production system. When some changes are needed for a production system a simulation can give a good idea how changes will affect the actual system. This indeed saves a lot of work and money, as it is obviously cheaper to run a simulation than changing a large factory.

There is one problem though with simulation/optimization. Simulations can be complex and often needs much computational power. This makes many simulations time consuming. A complex simulation can take hours to run. Hence, the parameters for each simulation run have to be wisely chosen. This is obviously a problem where quick answers are needed for many different input parameters. A typical scenario when quick answers are needed is for example when answers to questions like: “What happens if we reduce buffer-size two in the production-line by half?” are wanted. These types of questions are often called “what if” questions.

To solve this problem meta-models are used. A meta-model is a simpler model of a complex model. It can be used to represent input-output relationships over a wide range of parameter space. These meta-models can be used to answer “what if” types of questions quickly (in a second). Recently artificial neural networks (ANNs) gained popularity as being used as meta-models. Studies focusing on ANNs as a simulation meta-modeling approach have given quite encouraging results such as when Badiru and Seiger (1993) used a neural network as simulation meta-model for economic analysis of risky projects. They apply a neural network model to predict the potential returns from investment projects with stochastic parameters, e.g. initial investment, the rate of return, and the investment period. The meta-model developed could analyze the performance of potential future projects without re-running the time-consuming simulation. The authors carried out a similar study in 1998 when they experimented with simple investment project. The result showed that good predictive capability was achieved using conventional simulation along with artificial neural networks.

Another example of successful usage of ANNs as meta-models is e.g. experiment done by Kilmer and Smith (1993) and Kilmer et al. (1994) on an inventory problem. It showed that neural networks perform better than first-order and second-order linear regression models when compared with these two traditional approaches.

The last example was done by Altiparmak, Dengiz and Bulgak (2002) when they developed an ANN as a meta-model for asynchronous assembly system to search for optimal buffer sizes. This meta-model turned out to be more precise and better than exponential meta-model.

Unfortunately ANNs have not always been the best choice as meta-models. In an early study, Fishwick et al. (1989) compared ANNs with traditional approaches, i.e.
linear regression and response surface methods, and concluded that the former is inadequate to represent system characteristics.

To sum up, most of the examples give positive results and show that using artificial neural networks is a good choice when building a simulation meta-model.

Even though many studies have given positive results by using ANNs as simulation meta-models a methodology on how to develop them has not yet been well established (Alam, F.M. et al., 2004).

This final year project investigates the difference several known ANN architectures have on predictability when the underlying simulation model is of a complex production system. The primary distinction between this work and earlier published work on meta-modeling is the problem domain and the size and complexity of the simulation that is to be meta-modeled.
2 Background

This section gives information about what a simulation and a simulation meta-model are as well as presenting artificial neural networks in general.

2.1 Simulation

What is a simulation? A simulation is imitating or estimating how events might occur in a real situation. It can for example involve complex mathematical modeling. A simulation can place you under realistic conditions. These conditions can change as a result of behaviour of others involved in the simulation. Hence, it is not possible to anticipate the sequence of events or the final outcome. Complex systems or processes that are difficult to change in reality are often simulated. Simulation can be changed quite easily by changing its parameters. These parameters can for example represent a buffer stock size in a production line or the time it takes each unit in the production line to finish its job. These are parameters that are very difficult to change in the reality. It is therefore great benefit to estimate the results of such changes by testing it first in a simulation.

2.2 Simulation meta-models

A meta-model is a model of a model. As mentioned in the introduction simulations can often be very complex and slow. Therefore it is often great need for simpler and faster model that can give results from different configurations quickly.

A simulation can be represented as a mathematical function $g(v)$, where $v$ is a vector that usually includes random components. The vector $v$ could for example include components like number of machines and buffer sizes in a production line.

Meta-models are typically developed separately for each component of $v$, that is, for each coordinate function of $g$ (Barton, R.R., 1998). Mathematical representation of meta-models could be the function $f$ and the predicted output from it would be $f(x)$. This gives

\[
g(x) = f(x)
\]

Barton, R.R. (1998) mentions three major problems in meta-modeling. They are: i) the choice of a functional form for $f$, ii) the design of experiments, i.e., the selection of a set of $x$ points at which to observe $g(v)$ (run the full model) to adjust the fit of $f$ to $g$, the assignment of random number streams, the length of runs, etc., and iii) the assessment of the adequacy of the fitted metamodel (confidence intervals, hypothesis tests, lack of fit and other model diagnostics).

There exist several kinds of meta-models such as response surface meta-models, spline meta-models, radial basis function meta-models and artificial neural network meta-models. For more information about those models, refer to Barton, R.R. (1998).

2.3 Artificial neural network (ANN)

As mentioned in last section (2.2) artificial neural networks can be used for meta-modeling. This section presents artificial neural networks in general as well as different ANN architectures that are used through this final year project.

According to Callan, R., the human brain inspired artificial neural networks. They are parallel computing devices consisting of many interconnected processors. These
computing devices, or nodes, are simple in structure. They have one mathematical activation function, which is a function that calculates the output from the node from the input it receives. The input to each node or the net value as it is called is the summation of the weights connected to it. The net value for the node in figure 1 can be calculated with equation 1 (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 10).

**Equation 1**

\[
net = \sum_{i=1}^{n} w_i x_i
\]

Figure 1 shows how typical node looks like.

![Figure 1](image)

Figure 1 The figure shows how the three leftmost nodes are connected to the node to the right. These connections have weights that are summed and used as an input to the rightmost node.

Each node sends its output further to the next nodes it is connected to in the network (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 9).

A typical ANN consists of three layers, input layer, hidden layer and output layer (Badiru, A.B., Sieger, D.B., 1998). Every node in each layer is connected to some other nodes in the network. Figure 2 shows a feed-forward network which means that every node in each layer is connected to every node in the next preceding layer. The nodes are connected to each other by weights. These weights begin in a random state but are adjusted through a training process (Altiparmak, F., Dengiz, B., Bulgak, A.A., 2002).

An ANN is based on some form of learning model (Judd, 1990) Learning can be either supervised, unsupervised or reinforcement. In supervised learning ANN can be trained to learn the connection between different inputs and outputs. The training is done by presenting the ANN a training set. The training data set consists of many lines of exemplars. An exemplar can e.g. be on the form \((x_1, x_2, x_3) \rightarrow y_1, y_2\) where \(x_1, x_2\) and \(x_3\) are the inputs that generate the outputs \(y_1\) and \(y_2\). When the training set has been presented to the ANN the weights between the nodes are updated. Each time the ANN is presented the training set it has been trained one epoch. This is done over and over again until the difference between the target output and the ANN output is small enough or when no weight changes occur (Altiparmak, F., Dengiz, B., Bulgak, A.A., 2002). It is important for ANNs to generalize well, i.e. it is important for ANNs to give reasonably good answers for inputs it has not seen before. Overtraining is a term used for ANNs that do not generalize well. The problem with ANNs that have been over trained is that instead of learning the “function” behind the proble, they memorize the answers provided for training examples. The number of epochs ANNs need to finish their training can vary from one ANN to another. When an ANN has been trained the weights can yield the connection between the input and the output.
In unsupervised learning there is no data that the ANN can learn from like in supervised learning. Instead the ANN clusters patterns in the input data. All patterns in the same cluster have something in common, i.e. they are similar in some way. An example would be if the network is given a task of grouping animals, then all animals with legs would be in the same group and all animals with no legs would be in another group. (Callan, R., 1999).

Reinforcement learning is between supervised and unsupervised learning. Instead of being told what to learn like in supervised learning the system (ANN or something that is learning with this approach) learns from reinforcement. For example if the system does something bad it is rewarded with something bad and when it does something good it is rewarded with something good. When time passes the system learns to only do things it will be rewarded with something good.

When supervised learning is used an algorithm called backpropagation is used to change the weights in the ANN. In short the backpropagation algorithm defines two sweeps of the ANN: first a forward sweep from the input layer to the output layer, and then a backward sweep from the output layer to the input layer. The forward sweep propagates the input vectors through the ANN to provide outputs to the output layer. The output from the forward sweep is compared to the preferred output and the error is calculated. The backward sweep is similar to the forward sweep except that the error value gathered after the forward sweeps is propagated back through the network to determine how the weights are to be changed during training. Figure 2 illustrates this.

![Diagram](image)

Figure 2 The shaded hidden unit sends activation to each output unit, and so during the backward sweep this hidden unit will receive error signals from the output units. (Callan, R., 1999, p. 33)

2.4 Artificial neural network architectures

2.4.1 Feed-forward ANN

Mehrotra, K., Mohan, C.K. and Ranka, S. (1997) describe feed-forward ANNs as networks where connections are only allowed from a node in layer i to nodes in layer i+1. These networks are often described by sequence of numbers like e.g. 4-2-2 which means that the network has four input nodes, two hidden nodes and two output nodes. This is the most common type of neural network in use. The concept behind feed-
forward networks is that successively higher layers abstract successively higher-level features from preceding layers. (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 20)

These networks are very popular because they are very general and can be used for many kinds of problems. Feed-forward networks are called universal function approximators which mean that in theory they are able to imitate any mathematical function that exist.

Figure 3 shows a feed-forward ANN. The circles on the figure are the nodes and the lines between the nodes are the weighted connections. The input to the nodes in the hidden layer and the output layer are is the net value for each node (section 2.3).

![Weighted synapses (connections between nodes)](image)

During training Error feedback

Hidden layer Input layer Output layer

**Figure 3 A typical neural network model (this one is feed-forward network)**

**2.4.2 Generalized feed-forward ANN**

A generalized feed-forward ANN is precisely like the ordinary feed-forward ANN, i.e. connections can only made to successive layers. The only difference is that connections are allowed to bypass layers, i.e. connections can be made from the input layer to the second or third hidden layer etc.

The good thing about generalized feed-forward ANNs is that they often can solve problems much more efficiently than ordinary feed-forward ANNs. In some cases a standard feed-forward ANN requires hundreds of times more epochs of training than the generalized feed-forward (for the same size ANN). This is due to the ability to project activities forward by bypassing layers. The result is that the training of the layers closer to the input becomes much more efficient.

**2.4.3 Elman ANN**

According to Mehrotra, K., Mohan, C.K. and Ranka, S. (1997) a recurrent ANN is an ANN that contains connections from output nodes to hidden layer and/or input layer nodes or from hidden layer nodes to input nodes. Recurrent network also allow interconnections between nodes in the same layer. (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 136)

Elman ANNs are early forms of recurrent ANNs. These types of ANNs are often called simple recurrent networks (SRN) but it will be referred as Elman ANN in this
final year project. These ANNs have a form of short term memory. Elman (1990) demonstrated that an SRN can predict the next item in a sequence from the current and preceding input. In an Elman ANN the hidden units are fed back into the input layer as shown in figure 4.

![Sample of an Elman ANN. Connections from the hidden layer feed back to the input layer.](image)

### 2.4.4 Radial basis ANNs

Radial basis function artificial neural networks (RBF nets) are three-layered ANNs with the usual input layer to distribute a pattern to the first layer of weights, a hidden layer and an output layer. The activation function used in the radial basis ANN in this final year project is the Gaussian function. It is described by equation 2

Equation 2

\[ \rho(u) \propto e^{-\frac{(u/c)^2}{2}} \]

RBF ANNs are general purpose ANNs just like feed-forward ANNs and are most often used for function approximation problems. (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 141-149).

The simulation of the complex production system can be looked at as a mathematical function. Therefore it is interesting to investigate radial basis ANNs because they could be able to perform very well in this case.

### 2.4.5 Modular ANNs

There are many problems that are best solved with modular ANNs. Modular ANNs consist of several (neural network) modules that are combined in logical manner. These networks can learn different aspects of a problem by solving smaller tasks separately. (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 21)
3 Problem description and statement

3.1 Production system description

The complex production system that is referred to in this final year project produces cylinder blocks for fuel and diesel motors in personal cars (Figure 5). The whole production line is very complex and includes many individual stations that have different tasks in the production. The work process has eleven buffer stocks where some are more important than others. The production line has been modeled with simulation software as precisely as possible. The simulation takes into account possible jams on individual stations in the production. The production line has three significant buffer stocks. Optimization is needed for these buffer stocks to minimize the production time of each cylinder block and to minimize the work in process (WIP). What is meant by work in process is the amount of cylinder blocks that are being produced in the production line. The reason for minimizing work in process is to minimize the capital binding. The three significant buffer stocks have been tested with different capacities in three experiment plans in the simulation. Totally there were 277 runs through the simulation. Because of the amount of time that is needed to complete these simulations it is not feasible to find the optimal buffer sizes to get the desired output. The optimal buffer sizes could be somewhere far from the sizes that were tested. To be able to find these buffer sizes it would require many more simulation runs. Therefore it is necessary to create a simulation meta-model that can be used to search for optimal buffer sizes in the production line quickly.

Figure 5 A simplified image of how the real production system looks like.

3.2 Problem description

The problem with simulation-based optimization is that a simulation can be too time consuming for problems of practical interest (Boesel et al., 2001). Therefore it is necessary to make the process more effective by using meta-models instead of the actual simulation. Regression models have widely been used as meta-models successfully but recently it has been shown that ANNs can often perform better. Altiparmak, Dengiz and Bulgak (2002) showed that neural networks worked better
than regression models as meta-models for asynchronous assembly system. Nasereddin and Mollaghasemi (1999) investigated different ANN architectures when developing a reverse meta-model of a particular system. What is meant by reverse metamodel is that the output from the simulation works as an input to the meta-model and the input to the simulation works as desired output in the ANN metamodel. The purpose of their research was to provide a methodology of how to build and examine a reverse simulation metamodel. Therefore it was necessary to investigate how different architectures worked for the given problem. They also state that a future research would be to determine the best meta-model for other types of problems. The meta-models developed in this final year project are not reverse but direct, which means that the input in the simulation works as an input to the artificial neural network meta-model and the output from the simulation works as a desired output for it. As mentioned in the introduction a methodology on how to develop simulation meta-models has not yet been well established. In this final year project different artificial neural network architectures will be used as meta-models and compared in terms of predictability when the underlying simulation is of a complex production system.

3.3 Aim and objectives

As mentioned in the introduction it can take long time to run complex simulations. The simulation for the underlying complex production system is a rather complex simulation. Therefore a meta-model is needed that can be used instead of the simulation to search for optimal buffer sizes.

The aim of this final year project is to investigate the effects different ANN architectures have on predictability when ANNs are used as meta-models for simulation of complex production system.

In order to accomplish this task (i) several artificial neural network architectures have to be chosen that have the characteristics of handling such a problem (function approximation). (ii) Training data has to be gathered that represents the system as precisely as possible. The better the training data represents the simulation of the system the better the ANN will imitate the simulation. (iii) These ANNs have to be constructed by some means and trained with the training data. (iii) Performance of these networks has to be compared to be able to compare their predictability and accuracy.

Figure 6 shows what parts of the production system the meta-models undertake.

![Figure 6 This is how the production system looks like for the meta-model.](image-url)
4 Approach

4.1 Choice of ANN architectures

Before architecture can be chosen it is necessary to look at the problem. In this case the data from the simulation that is to be meta-modeled is available that has known inputs and outputs, i.e. there exists data that gives information about how different inputs give different outputs. When such data is available supervised learning is used.

There exist many different ANN architectures that are used with supervised learning. These architectures are used with different problems with different characteristics. The simulation of the complex production system can be seen as a mathematical function. Different input values give continuous output values. The ANNs will therefore be used as function approximations.

All of the chosen architectures are well known and can be used together with supervised learning. All of the architectures can be used for function approximation problems.

Feed-forward and generalized feed-forward ANNs

Feed-forward (along with generalized feed-forward) ANNs are widely used architectures in industry. These ANNs are trained with the backpropagation learning algorithm and according to Mehrotra, K., Mohan, C.K. and Ranka, S. (1997) they suit well for function approximation problems such as the problem accounted in this final year project.

Elman ANNs

Elman (1990) networks extend the ordinary feed-forward ANN such that it can remember past activity. This ability could be helpful for meta-modeling the simulation of the complex production system. According to Mehrotra, K., Mohan, C.K. and Ranka, S. (1997) recurrent ANNs (Elman ANN is one type of recurrent ANN) have been used successfully for problems that have complex implicit relationships between inputs and outputs like the problem accounted in this project. This ANN can be trained using backpropagation training algorithm.

Radial basis ANNs

Radial basis ANNs are general-purpose ANNs that are mainly used for system modeling, prediction and classification. Radial basis ANNs can be used to solve the same problems. This means that radial basis ANNs can be applied to a range of tasks, including function approximation that is the problem accounted for here. Radial basis ANNs can be trained with backpropagation training algorithm.

Modular ANNs

Modular ANNs are chosen because Nasereddin and Mollaghasemi (1999) proved successful as a simulation meta-model and also because it can learn different aspects of a problem. Each module could for example concentrate on the connection between the inputs and one of the outputs. Modular ANN can be trained with backpropagation algorithm.
4.2 Obtaining training data

Training data is the most important factor when training ANNs. The data should represent as many states of the problem being tackled as possible and there should be sufficient data to allow test and validation data sets to be extracted.

The training data used to train the ANNs in this final year project comes from simulation runs that have been designed carefully with the help of a technique called design of experiments. The data was generated in a research project at the University of Skövde called Optimist (www.his.se/optimist) to be used in this final year project. Design of Experiments (DoE) (Kapur, K.C., 1993) is a technique that can be of great help in finding important factors in building a simulation model. According to Sanchez, S. M., (2005) a well-designed experiment makes it possible to examine many more factors than would otherwise be possible, while providing insights that could not be gleaned from trial-and-error approaches or by sampling factors one at a time.

The production line has eleven buffer stocks where three of them are more significant than the others. By applying DoE considerable amount of information about the production system can be obtained with limited number of experimental runs (Mezgár et al., 1997). This is practically good as it takes each simulation 1.5 hours to complete. The experiment to find the three buffer stocks was done by running the simulation on the eleven buffer stocks with two different capacity levels, low and high. The buffer stocks that showed to be most important for the production were chosen for further investigation.

The three chosen buffer stocks are tested in three experiment plans. The first experiment plan includes five different buffer stock capacities for the three buffer stocks. That results in $5^3 = 125$ exemplars that the ANN can be trained with. Experiment plan number two includes three different capacities for the buffer stocks. That gives $3^3 = 27$ exemplars of data. The third and last experiment plan has five capacities (of course different from the first experiment plan) that give another 125 exemplars of data. The data that can be used results in being 277 exemplars. A graphical illustration of this process can be seen in figure 7.

The data from the 277 simulation runs are used to train the different artificial neural network architectures. Using data generated with the help of DoE is a good idea for training an ANN because it is necessary that the training set represents the system characteristics as precisely as possible. The idea of having DoE generating the training and test data for the neural network has been discussed e.g. by Mezgár et al (1997).
4.3 Construction of ANNs

The ANNs are made with the help of software package from NeuroDimension Inc called NeuroSolutions. This software package makes it possible to build, train and analyze many different kinds of ANN architectures as well as using them in own software later on. Detailed description on how the different architectures were built and trained can be found in section 5 in this report.

When building ANNs there are few practical things that have to be considered. These practical things are the choice of ANN parameters.

Learning rate ($\eta$) is one parameter that has to be configured. The magnitude change of weights in the network depends on the choice of learning rate. Large value of $\eta$ leads to rapid learning but the weights may then oscillate, while low values imply slow
learning. This parameter is chosen adaptively. (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 81)

Number of hidden layers and number of nodes in each hidden layer is another important parameter. With too many nodes in a hidden layer the ANN will not generalize well. It is more likely that the ANN will memorize the training set it is presented to and therefore be unable to give good answers to inputs it has not seen before. With too many hidden layer nodes the computations get more expensive. (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 85)

Momentum ($\alpha$) is yet another parameter that has to be configured. The backpropagation learning algorithm leads the weights in the ANN to a local minimum of the mean square error (MSE), possibly substantially different from the global minimum that corresponds to the best choice of weights (see figure 8). The momentum term is then added to the weight update rule. The weight change results in being the average of the current suggestion of weight change and the weight change used in previous step of the learning process. The value between 0 and 1 is chosen for the momentum. It is like the learning rate ($\eta$) chosen adaptively. A value close to 0 implies that the past history does not have much effect on the weight change, while a value closer to 1 suggests that the current error has little effect on the weight change. (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 83)

![Diagram of optimization problem](image)

**Figure 8** Graph of jagged error surface

All ANNs use 20% of the available data for cross-validation and 20% of the available data as test set. Cross validation is a highly recommended method for stopping network training. This method monitors the error on an independent set of data and stops training when this error begins to increase. This is considered to be the point of best generalization (Mehrotra, K., Mohan, C.K., Ranka, S. 1997, p. 38). When the training is over the testing set is used to test the performance of the network. Once the network is trained the weights are then frozen, the testing set is fed into the network and the network output is compared with the desired output. The same test set is used
for all the different ANN architectures. By using this method the number of epochs needed for training can vary between architectures.

4.4 Performance measures for comparison of ANNs

In order to compare the ANNs it is necessary to have some performance measures to measure different capabilities of the networks. It is important to measure how well the ANNs adapt to the training data, i.e. they give relatively small error on patterns they have already seen. Another important factor is for the ANNs to generalize well, i.e. that they give reasonably good answers to inputs they have not been presented to. If the ANNs generalize well they have captured the characteristics of the production system well. There are six different performance measures used through the experiment to compare the different ANN architectures.

Mean Square Error (MSE)

The mean square error is used to determine how well the ANN output fits the desired output presented in the training data.

The formula for the mean square error is:

$$\text{MSE} = \frac{\sum_{j=0}^{P} \sum_{i=0}^{N} (d_{ij} - y_{ij})^2}{NP}$$

Where

- $P$: number of output processing elements
- $N$: number of exemplars in data set
- $y_{ij}$: network output for exemplar $i$ at processing element $j$
- $d_{ij}$: desired output for exemplar $i$ at processing element $j$

Normalized Mean Square Error (NMSE)

Battaglia, G.J. (1996), states that in application, MSE has some drawbacks which can be avoided using NMSE, a normalized form of the MSE. These disadvantages are:

- Because MSE depends upon unit of measure, it cannot be averaged meaningfully over a mixture of processes. This means that usually it is no point in comparing the MSE of one process with the MSE of a different process.
- The utility of MSEs is reduced by the fact that they are often awkward numbers like $5.734 \times 10^{-6}$ and have no apparent significance except in comparison with previous values for the same process.

The formula for the normalized mean square error is:
\[ \text{NMSE} = \frac{P \cdot N \cdot \text{MSE}}{\sum_{j=0}^{P} N \sum_{i=0}^{N} d_{ij}^2 - \left( \frac{\sum_{i=0}^{N} d_{ij}}{N} \right)^2} \]

Where

- \( P \): number of output processing elements
- \( N \): number of exemplars in data set
- \( \text{MSE} \): Mean Square Error
- \( d_{ij} \): desired output for exemplar \( i \) at processing element \( j \)

The correlation coefficient \( r \)

The size of the mean square error (MSE) can be used to determine how well the network output fits the desired output, but it doesn't necessarily reflect whether the two sets of data move in the same direction. For instance, by simply scaling the network output, we can change the MSE without changing the directionality of the data. The correlation coefficient \( (r) \) solves this problem. By definition, the correlation coefficient between a network output \( x \) and a desired output \( d \) is:

\[
\begin{align*}
    r &= \frac{\sum_{i} (x_i - \bar{x})(d_i - \bar{d})}{\sqrt{\frac{\sum_{i} (d_i - \bar{d})^2}{N} \sqrt{\frac{\sum_{i} (x_i - \bar{x})^2}{N}}}} \\
\end{align*}
\]

The correlation coefficient is confined to the range \([-1, 1]\). When \( r = 1 \) there is \( x \) fits \( d \) perfectly. When \( r = -1 \) on the other hand, there is a perfectly linear negative correlation (relationship) between \( x \) and \( d \), that is, they vary in opposite ways (when \( x \) increases, \( d \) decreases by the same amount). When \( r = 0 \) there is no correlation between \( x \) and \( d \), i.e. the variables are called uncorrelated. Intermediate values describe partial correlations. For example a correlation coefficient of 0.88 means that the fit of the model to the data is reasonably good.

Percent error

The percent error is defined by the following formula:

\[
%\text{Error} = \frac{100}{N} \sum_{j=0}^{P} \sum_{i=0}^{N} \frac{|dy_{ij} - dd_{ij}|}{dd_{ij}} \]

Where

- \( P \): number of output processing elements
- \( N \): number of exemplars in data set
- \( dy_{ij} \): de-normalized network output for exemplar \( i \) at processing element \( j \)
- \( dd_{ij} \): de-normalized desired output for exemplar \( i \) at processing element \( j \)
Note that this value can easily be misleading. For example, say that output data is in the range of 0 to 100. For one exemplar desired output is 0.1 and actual output is 0.2. Even though the two values are quite close, the percent error for this exemplar is 100.

**Akaike's information criterion (AIC)**

Akaike's information criterion (AIC) is used to measure the tradeoff between training performance and network size. The goal is to minimize this term to produce a network with the best generalization. Generalization is important for ANNs to work well on data they have not seen before. This measurement can be helpful when comparing different ANN architectures on how well they generalize. The mathematical representation of AIC is:

\[
AIC(k) = N \ln(MSE) + 2k
\]

Where

k: number of network weights  
N: number of exemplars in training set  
MSE: mean square error

**Rissanen's minimum description length (MDL)**

Rissanen's minimum description length (MDL) criterion is similar to the AIC in that it tries to combine the model's error with the number of degrees of freedom to determine the level of generalization. Like AIC this measurement can be helpful when comparing different ANN architectures on how well they generalize. The goal is also to minimize this term. The mathematical representation of MDL is:

\[
MDL(k) = N \ln(MSE) + 0.5K \ln(N)
\]

Where

k: number of network weights  
N: number of exemplars in training set  
MSE: mean square error
5 Implementation

There are many parameters that have to be adjusted when building, training and testing ANNs like mentioned in section 4.3. The different architectures differ a little bit in configuration although there are some settings that are the same for all of them.

There were 277 exemplars of data available for training and testing the ANNs. The training set consisted of 155 exemplars found in the available data. All ANNs use the same 20% (55 exemplars) of the available data as test set. When the ANNs have been trained the test set is used to test their performance. Once an ANN is trained the weights are frozen, the test set is fed into the ANN and its output is compared with the desired output.

All ANNs have three input nodes that represent the three chosen buffer stocks in the production line and two output nodes that represent work-in-process and cycle-time.

As mentioned in section 4.3 some parameters are chosen adaptively in order to maximize the performance of the ANNs. This can result in slightly different parameter settings for different architectures. The best settings found are used.

Implementation of feed-forward ANN

The feed-forward ANN has one hidden layer with four nodes. The weights from the input layer to the hidden layer were adjusted with learning rate ($\eta$) = 0.7 while the weights from the hidden layer to the output layer were adjusted with learning rate ($\eta$) = 0.1. This means that after each epoch the weights in the first part of the ANN change more than the weights in the latter part (i.e. the first part learns more rapidly). The momentum ($\alpha$) for the ANN was equal to 0.7, which means that the current error as well as the past history of weight changes has effect on current weight changes. The ANN stopped training after 2888 epochs out of maximum 5000 epochs. It stopped because the error on the cross validation set increased (potential overtraining).

Implementation of generalized feed-forward ANN

The generalized feed-forward ANN has one hidden layer with four nodes. The generalized feed-forward network built has learning rate ($\eta$) = 0.7 from input layer to the hidden layer and 0.1 from hidden layer to the output layer. It has momentum ($\alpha$) = 0.7. The training procedure stopped after 4037 epochs.

Implementation of radial basis ANN

The radial basis ANN has two hidden layers. The first hidden layer has four nodes that use the Gaussian transfer function and the second layer includes 4 nodes which all use the ordinary sigmoid activation function. In order to implement a successful radial basis ANN it is necessary to find suitable centers for the Gaussian function. This is done with unsupervised learning approach. This particular network results in being a hybrid supervised-unsupervised ANN.

The first layer of weights (weights changed by unsupervised learning) has learning rate ($\eta$) = 0.01, the second layer of weights (weights changed by supervised learning) has learning rate ($\eta$) = 0.7 and the third layer of weights (weights changed by supervised learning) has learning rate ($\eta$) = 0.1. The network stopped training after 4123 epochs.

Implementation of Elman ANN
The Elman network has one hidden layer with four nodes. The first layer of weights has learning rate $\eta = 0.7$ while the second layer had $\eta = 0.7$. The momentum $\alpha$ was 0.7. The network stopped training after 2665 epochs.

**Implementation of modular ANN**

The modular ANN built had one hidden layer of nodes. The hidden layer was divided into two parts which each had four nodes. The network can be looked at as it is two feed-forward networks combined. The learning rate $\eta$ for the first layer of weights is 0.7 while the second layer had 0.1. The momentum $\alpha$ for the ANN was 0.7. The ANN stopped training after 2695 epochs.
6 Results

There were four objectives that had to be accomplished in this final year project. The first objective was to find ANN architectures that could be used for the given problem. Five different architectures were investigated that all had the potentialities of being useful for the problem as can be seen in following sections. These architectures were: Feed-forward ANN, Generalized feed-forward ANN, Modular ANN, Radial Basis ANN and Elman ANN. The second objective was to obtain training data that could represent the problem as precisely as possible. This was solved with well-designed experimental runs through simulation of the complex system described earlier in this dissertation. The amount of data obtained resulted in being enough to capture the characteristics of the complex system quite well. The third objective was to implement the chosen ANN architectures and train them with the data obtained. This was solved with number of experiments in configuration on ANN parameters such as learning rate, number of hidden layers and nodes etc. The best configuration found for the architectures was chosen. The fourth objective was to compare the performance of these ANNs. This included finding some performance measures that could be used to compare the ANNs to each other. Six performance measures were introduced, were some of them were more interesting than others.

Training results for the architectures investigated can be seen in the appendix. The appendix includes graphs and tables to illustrate how accurate the architectures are on their test set as well as their training set. Following section (6.1) compares these results.

6.1 Comparison of architectures

Figure 9 shows normalized mean squared error for the different architectures investigated. The Elman ANN architecture has the lowest error for both the test set and the training set. Figure 10 shows the mean square error for the different ANN architectures. The Elman ANN produces also the lowest mean square error of the five architectures investigated, both for the test set and the training set. This means that the output of the Elman ANN is closest to the actual output of all the five ANN architectures investigated. The ANN that produces the lowest percent error for the training set is also the Elman ANN but the modular ANN has the lowest percent error for the test set. Like can be seen the difference between the mean square errors for the architectures is very small. To understand this difference let’s look at the architecture that performed the best in terms of MSE for its test set, i.e. Elman ANN. The MSE for the Elman ANN is 0.010024944. This means by the definition of MSE (section 4.4) that the average error for each output node in the ANN is \( \sqrt{0.010024944} = 0.10012464 \). This is the average error of each output node if the output from it lies between 0 and 1. The reason for is this because the target data (the desired output) needs to be converted to lie within the operational range of the ANN. For example the target data for backpropagation network with sigmoid activation units need to lie between 0 and 1 because that is the range of the sigmoid function. To relate this small error value to the actual target it must be converted back. This will not be done here. It can easily be seen in figure 12 that the error for all the ANNs lays around 3%. The difference between architectures is not worth noticing.
Figure 9 Normalized mean square error column chart. The lower the error is the less the difference between the ANN output and the desired output is.

Figure 10 Mean square error column chart. The lower the error is the less the difference between the ANN output and the desired output is.
So far Elman ANN has given the best results in terms of accuracy even tough the difference between the architectures is considerably small. The two performance measures left measure how well the ANNs generalize. Figure 13 shows the Akaike's information criterion (AIC) for the different ANN architectures. The column chart shows the absolute value of the AIC for the ANNs so the ANN with the largest column should have the best generalization. The ordinary feed-forward ANN has the best generalization according to AIC. The generalized feed-forward ANN has the next best generalization. Figure 14 shows Rissanen's minimum description length (MDL). The results for that measurement are the same as for Akaike’s information criterion, i.e. the feed-forward ANNs have the best generalization.

Both AIC and MDL are calculated from MSE for the given network and number of network weights. With respect to the small difference in MSE between architectures and the fact that the ANNs are rather similar in size these results cannot be taken very seriously. The difference between the architectures for these generalization measurement tools is also considerably small.
Figure 12 this column chart shows Akaike's information criterion (AIC) for the different architectures. The higher the column is, the better the generalization is.

Figure 13 this column chart shows Rissanen's minimum description length (MDL) criterion for the different architectures. The higher the column is, the better the generalization is.
7 Related Work

There are some projects that are similar to this final year project in the sense of using artificial neural networks as meta-models. The difference is that they are used for different models of different complexity.

Badiru and Seiger (1993) used a neural network as simulation meta-model for economic analysis of risky projects. They apply a neural network model to predict the potential returns from investment projects with stochastic parameters, e.g. initial investment, the rate of return, and the investment period. The meta-model developed could analyze the performance of potential future projects without re-running the time-consuming simulation. The authors carried out a similar study in 1998 when they experimented with simple investment project. The result showed that good predictive capability was achieved using conventional simulation along with artificial neural networks.

Kilmer and Smith (1993) and Kilmer et al. (1994) showed that neural networks perform better on an inventory problem than first-order and second-order linear regression models when compared with these two traditional approaches.

Altiparmak, Dengiz and Bulgak (2002) developed an ANN as a meta-model for asynchronous assembly system to search for optimal buffer sizes. This meta-model turned out to be more precise and better than exponential meta-model. They used feed-forward as ANN architecture.

Early study in this field was done by Fishwick et al. (1989) when they compared ANNs with traditional approaches, i.e. linear regression and response surface methods, and concluded that the former is inadequate to represent system characteristics.

Nasereddin and Mollaghasemi (1999) investigated different ANN architectures when developing a reverse simulation meta-model of a particular system. The purpose of their research was to provide a methodology of how to build and examine a reverse simulation meta-model. They examined different ANN architectures and found out that modular ANN performed the best in terms of minimizing the error of prediction.

Alam, F.M, et al investigated the effects of experimental designs on the development of artificial neural networks as simulation meta-models. The data they used to train the ANNs was obtained with traditional full factorial design, random sampling design, a central composite design, a modified Latin Hypercube design and designs supplemented with domain knowledge. The results from their case study showed how much impact the experimental design chosen for the ANN training set can have on the predictive accuracy achieved by the meta-model. The best result came from modified Latin Hypercube design supplemented with domain knowledge.

Kilmer, A.R. et al used artificial neural networks as simulation meta-model for an emergency department simulation. They used the output from the simulation as a training data for the ANNs. The performance of the ANN meta-models was then compared to the simulation performance for estimating the mean and variance of patient time in the emergency department.

The primary distinction between this work and the published work mentioned here is the problem domain and the size and complexity of the simulation which is to be meta-modeled.
8 Conclusion

8.1 Summary

The aim of this final year project was to investigate the effects of different artificial neural network architectures on predictability when used as meta-models for complex production system. Four objectives had to be accomplished to successfully finish the project. These objectives were (i) several network architectures had to be chosen that had the characteristics of handling such a problem, (ii) training data had to be chosen that represents the system as precisely as possible, (iii) these ANNs had to be constructed and trained with the training data, (iii) performance of these networks had to be compared.

Five ANN architectures were used in this project. They were feed-forward ANN, generalized feed-forward ANN, modular ANN, radial basis ANN and Elman ANN. These are all well known architectures that are often used in function approximation problems similar to the one in this final year project.

The training data was obtained with design of experiments. This means that the input to the existing simulation was carefully chosen to get as precise image of the complex simulation in limited number of experimental runs.

The ANN architectures were trained with the chosen data. The actual ANNs were constructed and configured with trial-and-error approach, i.e. learning rate, momentum, number of hidden nodes etc was found through trial-and-error approach.

After the ANNs were trained they had to be compared. Six performance measures were found to compare them to each other. These measurements compare their accuracy and generalization.

After having analyzed the results it can be concluded that Elman ANN is the most accurate ANN of all the architectures in the tests conducted. It must be kept in mind that the difference is small. The results also showed that the ANNs are pretty accurate. Therefore it can be said that ANNs tested here are good choice as being used as meta-models in this case. Feed-forward and generalized feed-forward ANNs generalize best according to the generalization measurements (AIC and MDL). The same goes here, as before, i.e. there is no remarkable difference between the architectures. Of course the ANNs that performed the best in terms of accuracy, i.e. Elman ANN, would be preferred even though the difference is small between the architectures. It can also be said that radial basis ANN should not be preferred because it performed little bit worse than the other architectures in most cases.

8.2 Future work

Current implementation of the ANNs included trial-and-error approach of choosing important ANN parameters such as number of hidden nodes, learning rate and momentum. Another approach of configuring these parameters is using genetic algorithms (GA). By using GAs in configuring these parameters the optimal settings could be found. It takes however very long time to complete as it requires multiple training sessions for the ANNs.

The optimization part is still missing. This final year project was only focused on comparing different ANN architectures. What is left is testing them in finding the optimal buffer sizes for the three most important buffer stocks in the production line.
A good idea would be to compare genetic algorithms against simulated annealing. When Altiparmak, Dengiz and Bulgak (2002) developed an ANN as a meta-model for asynchronous assembly system to search for optimal buffer sizes they used simulated annealing. They stated that part of a future work would be investigating different search algorithms in such problems.
9 References


Appendix – Training results for ANN architectures

Training results for feed-forward ANN

Feed-forward ANN test set results for cycle-time. Desired output versus ANN output

Feed-forward ANN training set results for cycle time. Desired output versus ANN output

Feed-forward ANN test set results for Work-In-Process. Desired output versus ANN output

Feed-forward ANN training set results for Work-In-Process. Desired output versus ANN output

Performance measures for feed-forward ANN

<table>
<thead>
<tr>
<th>Measure type</th>
<th>Test set performance</th>
<th>Training set performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.011283025</td>
<td>0.0091066</td>
</tr>
<tr>
<td>NMSE</td>
<td>0.094264129</td>
<td>0.0510327</td>
</tr>
<tr>
<td>r</td>
<td>0.949594014</td>
<td>0.9726273</td>
</tr>
<tr>
<td>%Error</td>
<td>3.146913746</td>
<td>2.9652241</td>
</tr>
<tr>
<td>AIC</td>
<td>Not relevant</td>
<td>-774.98108</td>
</tr>
<tr>
<td>MDL</td>
<td>Not relevant</td>
<td>-759.76479</td>
</tr>
</tbody>
</table>
Training results for generalized feed-forward ANN

Generalized Feed-forward ANN test set results for cycle-time. Desired output versus ANN output

Generalized Feed-forward ANN training set results for cycle-time. Desired output versus ANN output

Generalized Feed-forward ANN test set results for work-in-process. Desired output versus ANN output

Generalized Feed-forward ANN training set results for work-in-process. Desired output versus ANN output

Performance measures for generalized feed-forward ANN

<table>
<thead>
<tr>
<th>Measure type</th>
<th>Test set performance</th>
<th>Training set performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.010712</td>
<td>0.009278</td>
</tr>
<tr>
<td>NMSE</td>
<td>0.089492</td>
<td>0.051991</td>
</tr>
<tr>
<td>r</td>
<td>0.952071</td>
<td>0.9721</td>
</tr>
<tr>
<td>%Error</td>
<td>3.210191</td>
<td>3.088519</td>
</tr>
<tr>
<td>AIC</td>
<td>Not relevant</td>
<td>-759.707</td>
</tr>
<tr>
<td>MDL</td>
<td>Not relevant</td>
<td>-740.979</td>
</tr>
</tbody>
</table>
Training results for radial basis ANN

Radial basis ANN test set results for cycle-time. Desired output versus ANN output

Radial basis ANN training set results for cycle-time. Desired output versus ANN output

Radial basis ANN test set results for work-in-process. Desired output versus ANN output

Radial basis ANN training set results for work-in-process. Desired output versus ANN output

Performance measures for radial basis ANN

<table>
<thead>
<tr>
<th>Measure type</th>
<th>Test set performance</th>
<th>Training set performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.011149</td>
<td>0.010073</td>
</tr>
<tr>
<td>NMSE</td>
<td>0.093145</td>
<td>0.086448</td>
</tr>
<tr>
<td>r</td>
<td>0.949267</td>
<td>0.969609</td>
</tr>
<tr>
<td>%Error</td>
<td>3.448356</td>
<td>3.334707</td>
</tr>
<tr>
<td>AIC</td>
<td>Not relevant</td>
<td>-717.23</td>
</tr>
<tr>
<td>MDL</td>
<td>Not relevant</td>
<td>-690.309</td>
</tr>
</tbody>
</table>
Training results for Elman ANN

Elman ANN test set results for cycle-time. Desired output versus ANN output

Elman ANN training set results for cycle-time. Desired output versus ANN output

Elman ANN test set results for work-in-process. Desired output versus ANN output

Elman ANN training set results for work-in-process. Desired output versus ANN output

Performance measures for Elman ANN

<table>
<thead>
<tr>
<th>Measure type</th>
<th>Test set performance</th>
<th>Training set performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.010025</td>
<td>0.008376</td>
</tr>
<tr>
<td>NMSE</td>
<td>0.083753</td>
<td>0.046936</td>
</tr>
<tr>
<td>r</td>
<td>0.95264</td>
<td>0.974682</td>
</tr>
<tr>
<td>%Error</td>
<td>3.282849</td>
<td>2.843931</td>
</tr>
<tr>
<td>AIC</td>
<td>Not relevant</td>
<td>-749.708</td>
</tr>
<tr>
<td>MDL</td>
<td>Not relevant</td>
<td>-722.787</td>
</tr>
</tbody>
</table>
Training results for modular ANN

Modular ANN test set results for cycle-time. Desired output versus ANN output

Modular ANN training set results for cycle-time. Desired output versus ANN output

Modular ANN test set results for work-in-process. Desired output versus ANN output

Modular ANN test set results for work-in-process. Desired output versus ANN output

Performance measures for modular ANN

<table>
<thead>
<tr>
<th>Measure type</th>
<th>Test set performance</th>
<th>Training set performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.010636</td>
<td>0.009447</td>
</tr>
<tr>
<td>NMSE</td>
<td>0.088855</td>
<td>0.05294</td>
</tr>
<tr>
<td>r</td>
<td>0.952624</td>
<td>0.971767</td>
</tr>
<tr>
<td>%Error</td>
<td>3.039989</td>
<td>3.056463</td>
</tr>
<tr>
<td>AIC</td>
<td>Not relevant</td>
<td>-720.522</td>
</tr>
<tr>
<td>MDL</td>
<td>Not relevant</td>
<td>-691.26</td>
</tr>
</tbody>
</table>