universität innsbruck

Seminar Report

Neural Network Analysis - Weight Initialization

Quantitative Research Methods

Manuel Schneckenreither

manuel.schneckenreither@uibk.ac.at

5 June 2018

Abstract

Neural networks are used to approximate a functions. In the learning process the weights which map the input values to the output values are iteratively updated, such that the output of the neural network adapts to the desired values. It has been shown that correct weight initialization can reduce training times of neural networks tremendously. This seminar paper discusses and analyses recent literature for initialization of feedforward neural networks. Furthermore, we empirically prove these results by analyzing and comparing state-of-the-art weight initialization methods.

Contents

1 Introduction

Neural networks are becoming more and more widely used in manufacturing. Applications range from scheduling and load forecasting (e.g., [27, 34, 12, 17]), over the estimation of production costs (e.g., [3, 31]) to usage of neural networks as simulation metamodel outcomes (e.g., [15, 28]). However, generally speaking neural networks are always used to approximate functions. Thus it is important to understand the underlying ideas of neural networks to be able to apply the method thoroughly.

Neural network models for which the information flows in one direction without any feedback loops are called feedforward (neural) networks, or multilayer perceptrons, and are the quintessential deep learning neural networks. In the learning process the weights which map the input to the output values are iteratively updated, such that the output of the neural network adapts to the desired values. The number of layers used, as well as the type of the layers define the number of weights for a neural network. Additionally at the end of each layer there is an activation function which lifts the linear function of the actual layer to non-linearity. Commonly applied activation function are the rectifier linear unit ($ReLU$), the *softmax* function, or for instance the *tanh* function. According to the chosen activation functions the neural network is able to non-linearly map the input parameters to output values and thus non-linear functions can be approximated [9, p.164ff].

The training process, that is finding the correct weights, can be a tedious and sometimes it even fails completely, e.g. when the neural network is stuck in a local optimum. The most widely applied algorithm for finding the correct weights is backpropagation [13]. Backpropagation was originally introduced by Bryson and Ho in 1969 [2], forgotten and later independently rediscoverd [36, 24, 26]. This algorithm updates the weights of the layers according to the last known setting. Thus the correct initialization of the weights is crucial when learning an artificial neural network. It has been shown that correct weight initialization can reduce training times of neural networks tremendously, e.g. [33, 37, 20].

This seminar paper introduces, discusses and analyses state-of-the-art methods for

neural networks and the methodology for neural network analysis studies. Furthermore it puts a special focus on weight initialization and evaluates the presented techniques in an experimental setting. The rest of the paper is structured as follows. The next section introduces basic ideas of neural networks, before in Section 3 methods for performance evaluation of machine learning algorithms are highlighted. Section 4 and 5 present two articles which utilize these methods and analyse them according to the methods of Section 3. Section 6 provides an experiment for different weight initialization methods. We conclude in Section 7.

2 Basic Concepts in Neural Networks

A neural network consists of three types of layers. First the input layer, then the hidden layers and finally the output layer. There is exactly one input and one output layer for each neural network. The number of hidden layers can vary. The data flow is from the input layer through the hidden layers to the output layer. Each layer consists of several nodes (neurons), cf. Figure 1. In the simplest case the nodes of consecutive layers are fully connected, that is all nodes are linked as in Figure 1. Thus the model is associated with a directed acyclic graph. As the data flows from input to output the resulting function $f(\vec{x})$ can be described as a composition of functions, e.g. $f(\vec{x}) = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(\vec{x}))))$ for the example in Figure 1. Here the input layer is represented by the function $f^{(1)}$, the first hidden layer by $f^{(2)}$, the second by $f^{(3)}$, and the output layer by $f^{(4)}$. The length of this chain gives the depth of the model. An input z to a node y_i is given by the weight w_i corresponding to the N links to predecessors, the bias b, and a non-linear activation function g: $z(\vec{x}; \vec{w}) = g(\sum_{i=0}^{N} x_i \cdot w_i + b)$, cf. Figure 2. The aim is to learn the weights $\vec{w}$ for all layers such that the output of the neural network describes the desired value according to the given input.[9, 5].

The rest of this section is a short disambiguation for commonly used terms in neural network research.[9]

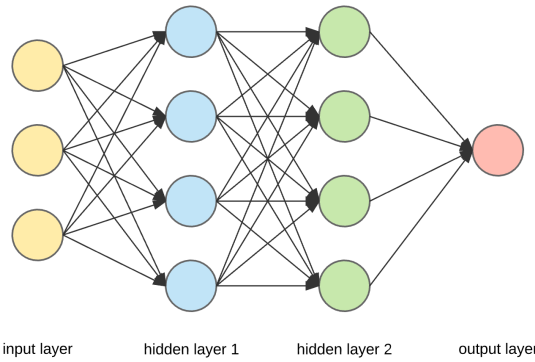

Figure 1: Schema for a feedforward neural networks [5].

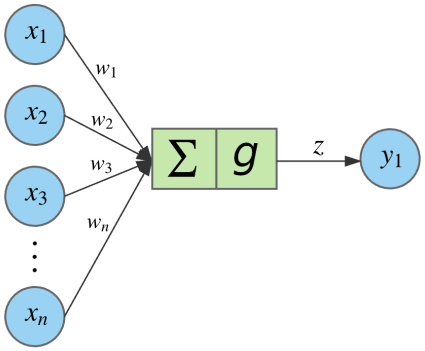

Figure 2: Connection to a node in a non-input layer (adapted from [5]).

Architecture. The architecture of a neural network describes the design of the model, that is the number and types of layers and the number of nodes, as well as the activation functions. The type of layers define the connection between the layers. In the simplest case this is a fully connected layer for which all nodes are connected. However, for instance in image classification convolutional layers, which reduce several nodes (pixel representations) to one node, are often used.

Weight initialization. Before starting the learning process the weights are initially declared. This is often done by random numbers.

Training settings. Due to the non-linearity the training process, that is learning the correct weights, is a complicated issue. The standard training algorithms are based on stochastic gradient decent backpropagation. These algorithms observe the difference of desired outcome and actual result of the neural network and update the weights starting by the output layer and then working its way to the input layer's weights.

Learning rate. The learning rate, usually a pre-determined parameter (hyperparameter), specifies the percentage of updating to new values, and is normally set to values between 10^{-2} and 10^{-4}.

Regularization/Weight decay. The L^2 parameter, also known as weight decay, is a regularization term that is applied in the learning process and drives the weights closer to the origin (of the search space). This is done by adding a regularization term to each weight update.

Momentum. Momentum is one of the most popular extensions to the backpropagation algorithm. It adds a term which can help to the network to get out of local optima [26].

Training length and stopping condition. To train a neural network the learning phase has to be determined. This can be either done by training the network a specified number of repetitions, or by defining a predicate as stopping condition, e.g. by an error rate.

3 Neural Network Research

The major decisions a neural network designer has to make include data preparation, also called feature extraction, input variable selection, the network architecture parameters such as the number of input, hidden and output nodes, node connection, training algorithm, transfer functions (activation functions) and many others.[38]

This section introduces methods for neural network research and is based upon the papers "Avoiding Pitfalls in Neural Network Research" by Zhang [38] and "Statistical

evaluation of neural network experiments: Minimum requirements and current practice" by Flexer [6]. Furthermore, various subsections are complemented with additional specific literature for the discussed subtopic.

The rest of this section introduces the main processes in neural network research and how these are executed. The processes are shown in the order of execution for a neural network study and include model building, data preparation, weight initialization, the training process, over- and underfitting, and model evaluation and comparison.

3.1 Model Building

As building a successful predictive neural network model is a hard task, it is important to be aware of the possibilities and follow a strategy to improve the network iteratively. However, the known rules of thumb work only for special problems or in special situations, and thus blindly applying them likely fails. Therefore, it is important to understand the underlying techniques to at least some extend. For instance, "many people believe that the functional form of the neural network model cannot be known precisely". This is not true. Their functional form is well known. The layers are connected linearly, while the activation functions are used to gain non-linear approximations. Furthermore, the results and behavior of artificial neural networks (ANN) can to some extend be understood and thus researchers should not simply treat neural networks as black boxes. If ignoring the internal working of ANNs the research could result in inappropriate treatment of neural networks and loss of the opportunity to gain insights from the established model.[38]

Determining an appropriate neural network architecture is one of the most important tasks when building an ANN model. According to the problem the input and output nodes can either be specified a priori learning, while for other applications these numbers have to be determined experimentally. The number of input nodes play a crucial role in the quality of the network. There is a tendency to specify a large number of variables to the model regardless of the relevance or redundancies. This can lead to overfitting and increased modeling time. Thus a correlation analysis of the possible input parameters could be done, before deciding which input parameters to present to the ANN.[38]

It is important to detail the model-building process in the article. Often papers lack of this kind of documentation and thus it is difficult or impossible to judge the appropriateness of the research and readers might not be able to replicate the results. This includes the architectures used, data splitting and preprocessing, training settings, weight initialization settings, learning rate, momentum, training length, stopping condition, algorithm used, regularization, and weight decay.[38]

3.2 Data Preparation

Data preprocessing is used to normalize the data, which speeds up the learning process [16]. However, it is unclear when data preprocessing is obligatory. For instance, Gorr [10] argues that neural networks should be able to detect both the nonlinear trend an the data seasonality.

The available data must be divided into two or three portions. The first portion is used for model-building, the second one for evaluation and if available the third one can be used for testing and parameter tuning. Using the whole data set for model building is considered a major pitfall in ANN learning. Similarly, it is a fallacy to train the model on more than one portion of data and select the best founded model. Thus the evaluation portion must always be used solely for evaluation. The size of the evaluation data should be at least 20% of the full data set. Nonetheless, data splitting is not about what proportion of data should be put in each subsample but rather about an adequate sample size.[6, 38]

The splitting of data ought to happen randomly to ensure each subsample is representative of the population. In case it is not possible to randomly select data instances, e.g. in time series applications, then the data splitting is done at the researches' discretion. However, it is important to ensure the portions reflect the characteristics of the data set.[38]

In case data transformation is applied then the results should be scaled to the original data range before evaluation to ensure interpreting is correct and methods can be compared. Many studies fail to indicate whether the performance measures are calculated

on the original or the transformed scale.[38]

3.3 Weight Initialization

These days there exist many initialization techniques. Often these methods are named after the authors of the paper. For instance in 2010 Xavier initialization after the first name of the first author in [8] was proposed. They analyse the activation values of the different layers and come to the conclusions that first of all the type of activation function is important and second that the defacto standard for the initialization for the weights $W_{i,j} \sim \mathcal{U}(-\frac{1}{\sqrt{n_l}}, \frac{1}{\sqrt{n_l}})$, where n_l is the number of nodes of layer l, causes the neural network to more likely fall into a local minimum and increase training times. According to their analytical analysis and underlying these ideas with experimental data they propose the initialization $W_{i,j} \sim \mathcal{U}(-\frac{\sqrt{6}}{\sqrt{n_l+n_{l+1}}}, \frac{\sqrt{6}}{\sqrt{n_l+n_{l+1}}})$. Furthermore, they propose to not use the *Sigmoid* activation function for the same reasons. However, in 2010, at the time this work was written, the activation function $ReLU$ was as not widely used and therefore not taken into consideration. Nonetheless, Xavier-initialization is based upon the idea the activations are linear while this is not valid for rectifier activations (e.g., $ReLU$).

Another standard initialization is to draw the weights from Gaussian distributions [11]. However, deep neural networks often have difficulties to converge when initialized with fixed standard deviations. Thus in 2015 He et al. [11] introduced another initialization scheme known as He et al-initialization nowadays. With i He et al. could surpass human-level performance for classification problems. The initialization method takes the often used activation function into account and also draws the weights from standard distribution. They propose a initialization with zero mean and standard deviation $\sqrt{2/c_l}$ where c_l is the number of connections of layer l. Thus in He et al. initialization $W_{i,j} \sim \mathcal{N}(0, \sqrt{2/n_l})$.

In the end all these initialization methods assume a specific neural network architecture for analytically deriving the initialization method. Thus it is well worth trying out several techniques for a specific problem and neural network setup when conducting a neural

network analysis study. However, Flexer [6] points out this random influences of weight initialization of which only a sample can be computed, must not be neglected and thus multiple runs of the same setting are required for a study.

3.4 Training Process

Due to the non-linearity the training process is a complicated issue. There are several different training algorithms available. The standard ones are backpropagation algorithms based on stochastic gradient decent. However, there are as well other approaches, like using a genetic algorithm to determine the network weights [23]. The basic backpropagation algorithm however, has many problems and should be avoided in favor of one of many optimizations.[38]

3.5 Overfitting and Underfitting

A wrong number of hidden layers can cause overfitting or underfitting problems. Unfortunately, there is no unique method in finding the right number of hidden layers. Thus these must be determined by trial and error.[38]

Accoring to Crawley and Talbet [4] overfitting can be solved by using regularization and early-stopping. Furthermore, overfitting can even bias the experimental evaluation by adapting the neural network to the data set of the inspected problem. They propose to conduct in each trial the model selection to prevent selection bias and because it reflects best practice in operational use.

3.6 Model Evaluation and Comparison

After modeling the performance of the model must be evaluated or validated using the data not used in the model-building stage. Flexer [6] proposes this as the minimum requirements for such evaluations after noting the lack of statistical evaluations in published studies.Prechelt [25] examines 190 papers on ANN learning published in leading neural network journals, finding that 78% are not evaluated thoroughly enough

to be acceptable, where a paper is called acceptable if it uses a minimum of two real or realistic problems and compares the results to those of at least one alternative algorithm.

First of all enough sample size needs to be allocated for the evaluation data set when splitting the data, otherwise the results might be due to chance. However, the appropriate size must be determined per application. Furthermore, an evaluation requires an appropriate comparison of ANN models with other models in statistics, machine learning, and data mining. Here it is important to examine the statistical significance of the models. Studies have shown that 33% of ANN articles have no comparison with other algorithms [25]. Adya and Collopy [1] propose three validation criteria for evaluation ANNs: (i) comparing to well-accepted models, (ii) using true out-of-samples, and (iii) ensuring enough sample size for evaluation. The use of confidence intervals provide means to asses the reliability of the model and its estimates.

The main goal for the statistical evaluation is to answer the question "whether there are effects of the variation of the independent variables (mainly type and certain paramter charactersitics of the data set in question) on the dependent variables (various perfomance measures like accuracy, root mean squared error or training time)" [6]. Flexer points out that it is crucial to have a training and a test set (about 1/3 of the data), and argues that multiple runs of the experiment must be conducted to ensure the actual performance of the algorithm is presented by reporting at least the mean and the variance. Furthermore, the in machine learning often used K-fold cross-validation is biased and should not be used.[6]

The parametric methods for statistical testing for which the normality assumption does not hold, instances can only be judged as "not significant" whereas otherwise they would have been judged "significant".[6]

Additionally instead of reporting the variance, it is better to show the corresponding confidence interval. Furthermore, the significance of the different between means should be computed via a t-test.[6]

4 Paper 1: Due-date assignment in wafer fabrication using artificial neural networks [14]

This section discusses the methodology used in the paper "Due-date assignment in wafer fabrication using artificial neural networks" by Hsu et al. [14] as it was introduced in Section 3. The paper examines if neural networks can outperform conventional and regression-based due-date assignment rules. The rest of the section discusses the paper in terms of model building, data preparation, weight initialization, the training process, over- and underfitting, and model evaluation and comparison.

4.1 Model Building

The authors conducted a statistical analysis (correlation analysis) on the data to determine the input variables. They found 42 variables to have significant correlation with the flow time and thus selected these as the input variables to the neural network. The whole modelling process was done for three different prediction models of which each was optimized. Thus they investigated the type of output and number of hidden layers for the neural network.

4.2 Data Preparation

They modeled a virtual wafer fabrication system based on a real wafer fabrication in Taiwan to generate data for the study. Therefore as many data as needed could be generated by the simulation. Furthermore, to ensure independence among the data only one in 10 outputs was randomly selected for the sample.

4.3 Weight Initialization

As with many studies also this study does not give any information on how the weights were initialized.

4.4 Training Process

As the study was published in 2004 it uses the standard backpropagation algorithm which uses the gradient-descent algorithm for updating the weights appropriately. This is clearly specified in the article.

4.5 Overfitting and Underfitting

"[The] learning process is repeated until the error between the actual and the desired output converges to a predefined threshold." Thus, the authors overcome over- and underfitting by setting the stopping criterion like this.

4.6 Model Evaluation and Comparison

The neural network results were compared to four other well-accepted models. As the data was generated using a simulation model, the sample size was also big enough and the data was not used for learning. Therefore, all minimum requirements were met. However, the authors do not report a variance and thus it is unclear whether multiple runs were conducted and if so, how many.

5 Paper 2: Job shop flow time prediction using neural networks [29]

This section discusses the paper "Job shop flow time prediction using neural networks" by Silva et al. [29] in terms of its methodology. The paper uses an artificial neural network to predict the flow time and, consequently, to estimate due dates of a hypothetical job shop. The rest of the section presents the paper in light of model building, data preparation, weight initialization, the training process, over- and underfitting, and model evaluation and comparison.

5.1 Model Building

The study uses a neural network with an input layer with 22 neurons, one hidden layer with 15 neurons, and an output layer with one node that returns the flow time prediction. Although, the article explains that the number of nodes for the hidden layer can be set empirically the authors do not state whether this was done. Furthermore it seems that the number of hidden layers was also not investigated. The objective of the algorithm is to minimize the difference between the actual and the expected flow time.

5.2 Data Preparation

Building upon a simulation, there is no need for data splitting, as the authors can produce as many independent and identically distributed data as they need. The authors explicitly state that the training and test data was independently generated.

5.3 Weight Initialization

The authors did not specify any information about the initialization of the neural network. They use a software package that seems to do this automatically.

5.4 Training Process

The study uses the Levenberg-Marquardt backpropagation algorithm as learning algorithm, which is an optimization of the standard backpropagation algorithm.

5.5 Overfitting and Underfitting

The stopping criterion for the training process is given by the amount of data that was generated for the training phase. This sums up to 200 jobs of data that were selected after removing the warm-up period.

5.6 Model Evaluation and Comparison

The evaluation data set was also composed of 200 jobs, however these were independently generated. Therefore, the sample data size was as big as the training set, which seems sufficient and the data set was independent from the training set. The results were compared to two other well-accepted models which is the minimum requirement. However, the authors do not present any variance and thus it is unclear whether multiple runs were performed, and if so if only the best result was reported. This would bias the reported performance and should not be done.

6 Weight Initialization Experiments

This section first introduces the considered problem, then the methodology used and finally the results of the experiment.

6.1 Problem description

The considered problem is simple. Consider a Cartesian coordinate system. The independent variables are x_1 and x_2, while the dependent variable is either class 0 or class 1. Whenever the input is in the circle given by the coordinates (0.33,0.33) with a radius of 0.33, or in the circle given by the coordinates (-0.33,-0.33) with a radius of 0.33 then the class is 1, otherwise it is 0. This is a classification problem, cf. Figure 3.

The artificial neural network is presented with 1000 random positions in the interval (-1,1) for both input variables and the according class. It then adapts the weight to such that the network adapts in the direction of the desired output. This procedure is supervised learning. The input and desired output is given to the algorithm in the training phase. In our case the artificial neural network shall learn the probability that a given position is class 1. The output of the neural network is a real number. Figure 4 shows a representation of the output of the tool. If the neural network outputs a value $\leqslant 0.2$ the cell is empty, if the output is $\leqslant 0.4$ the field is filled with a dot (.), if the output if $\leqslant 0.6$ the cell is filled with a minus sign (-), if the output is $\leqslant 0.8$ the field is filled with

an equality sign (=) and otherwise the cell is filled with a hash sign (#). The output was generated with 100000 rounds of learning, and therefore shows the

6.2 Methodology

We test three different weight initializations. First we initialize the weights uniformly (in the sequel called uniform initialization) with $W_{i,j} \sim \mathcal{U}(-\frac{1}{\sqrt{n_l}}, \frac{1}{\sqrt{n_l}})$, where n_l is the number of nodes of layer l, then we use Xavier initialization $W_{i,j} \sim \mathcal{U}(-\frac{\sqrt{6}}{\sqrt{n_l+n_{l+1}}}, \frac{\sqrt{6}}{\sqrt{n_l+n_{l+1}}})$, and finally we use He et al. initialization He et al. initialization $W_{i,j} \sim \mathcal{N}(0, \sqrt{2/n_l})$ as discussed in Section 3.3.

Neural Network Model We use the framework Grenade[1] to build the neural network[2]. The neural network consists of five fully connected layers with the number of neurons being 40-30-20-10-1 and the activation functions are *tanh-relu-relu-relu-logit*. The coordinates are specified as is, that is no data preparation is needed in this case. The number of layers is quite high. The purpose of this is to be able to observe different results for the weight initialization methods in this simple example.

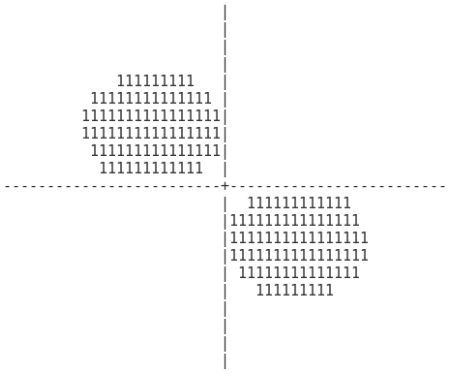

Figure 3: Classification problem. All positions for class 1 are marked.

[1] see `https://github.com/HuwCampbell/grenade`

[2] The code can be found at `https://github.com/schnecki/grenade/blob/master/examples/main/feedforward.hs`

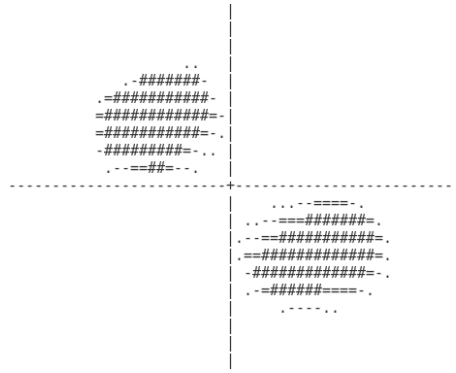

Figure 4: Example output of the tool.

Training Process For training we set the learning rate to 0.005 and the regularization l2 to 0.0005 We used no momentum. Thus the training algorithm is backpropagation with regularization. The training process lasts for 1000 inputs and afterwards the model is evaluated. The number of training rounds is quite low on purpose to be able to observe faster adapting setups.

Each setup was tested for 100 instances of the problem. After learning the evaluation tests for correct and incorrect classifications, as well as false negatives and false positives were calculated.

6.3 Results

Table 1 reports the results. On top of the table the means are reported, while at the bottom the standard deviations are listed.

He et al. initialization performs best and thus can correctly classify more test cases than Xavier or Uniform initialization. The ANOVA test reports significant differences between the groups. Furthermore the paired t-tests show that Xavier initialization is not significantly better than the Uniform initialization (p-value: 0.0765), but He et al. is highly significantly better than Xavier (p-value: <0.0001). Thus the fastest convergance can e achived with He et al. initialization.

In case of false positives Xavier initialization performs best, on average only reporting

Initialization	Correct	Incorrect	False Positives	False Negatives
Uniform	592.53	407.47	49.53	357.94
Xavier	604.73	395.27	7.77	387.50
He et al.	765.83	234.17	114.43	119.74
Uniform	60.80	60.80	164.65	108.50
Xavier	28.48	28.48	62.22	42.55
He et al.	145.45	145.45	158.74	120.22

Table 1: This table shows the means (top) and standard deviations (bottom) of the three tested initializations.

7.77 inputs as class 1, whereas the actual class was 0. The results are significant and direct comparisons show the significance of Xavier to Uniform with a p-value of 0.0186. Thus Xavier initialization works best when only little training instances are available but only little false positives should be produced by the model.

However, in case of false negatives He et al. initialization produces the smallest error rate. This group observation is highly significant and He et al. is highly significant better than Uniform (p-value <0.0001).

7 Conclusion

This seminar report presented a short introduction to (feed-forward) neural networks. It can be used as an entry point and for a disambiguation. Then in the main part important aspects for neural network research has been given and thoroughly discussed. A special focus was set to weight initialization, where state-of-the-art initialization methods were presented. Furthermore, we have given an overview of two neural network analysis papers and pointed out that neural network articles often lack of information. Finally, we experimentally showed that weight initialization can make a difference on a simple example. Therefore, for more complex solution spaces even better results with better weight initialization can be expected.

References

[1] M. Adya and F. Collopy. How effective are neural networks at forecasting and prediction? a review and evaluation. *J. Forecasting*, 17:481–495, 1998.

[2] A. Bryson and Y.-C. Ho. Applied optimal control, new york, blaisdell, 1969. *Google Scholar*.

[3] S. Cavalieri, P. Maccarrone, and R. Pinto. Parametric vs. neural network models for the estimation of production costs: A case study in the automotive industry. *International Journal of Production Economics*, 91(2):165 – 177, 2004.

[4] G. Cawley and N. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, July 2010.

[5] A. Dertat. Schema for a feedfoward neural networks. available online https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6, August 2017.

[6] A. Flexer. Statistical evaluation of neural network experiments: Minimum requirements and current practice. *Cybernetics and Systems Research*, pages 1005–1008, 1996.

[7] I. Giannoccaro. Complex systems methodologies for behavioural research in operations management: Nk fitness landscape. In *Behavioral Issues in Operations Management*, pages 23–47. Springer, 2013.

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[9] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] W. L. Gorr. Research prospective on neural network forecasting. *International Journal of Forecasting*, 10(1):1–4, 1994.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, Feb. 2015.

[12] W. He. Load forecasting via deep neural networks. *Procedia Computer Science*, 122:308 – 314, 2017. 5th International Conference on Information Technology and Quantitative Management, ITQM 2017.

[13] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[14] S. Y. Hsu and D. Y. Sha. Due date assignment using artificial neural networks under different shop floor control strategies. *International Journal of Production Research*, 42(9):1727–1745, 2004.

[15] R. D. Hurrion. An example of simulation optimisation using a neural network metamodel: finding the optimum number of kanbans in a manufacturing system. *Journal of the Operational Research Society*, 48(11):1105–1112, 1997.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] M.-T. Kechadi, K. S. Low, and G. Goncalves. Recurrent neural network approach for cyclic job shop scheduling problem. *Journal of Manufacturing Systems*, 32(4):689 – 699, 2013.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[19] H. Mather and G. W. Plossl. Priority fixation versus throughput planning. *Production and Inventory Management*, 19:27–51, 1978.

[20] D. Mishkin and J. Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[23] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989.

[24] D. Parker. Learning logic: technical report tr-87, center for computational research in economics and management science, 1985.

[25] L. Prechelt. A quantitative study of experimental evaluations of neural network learning algorithms: Current research practice. *Neural Networks*, 9(3):457–462, 1996.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[27] I. Sabuncuoglu and B. Gurgun. A neural network model for scheduling problems. *European Journal of Operational Research*, 93(2):288 – 299, 1996. Neural Networks and Operations Research/Management Science.

[28] M. Savsar and M. H. Choueiki. A neural network procedure for kanban allocation in jit production control systems. *International Journal of Production Research*, 38(14):3247–3265, 2000.

[29] C. Silva, V. Ribeiro, P. Coelho, V. Magalhães, and P. Neto. Job shop flow time prediction using neural networks. *Procedia Manufacturing*, 11:1767–1773, 2017.

[30] D. Simchi-Levi and Y. Zhao. The value of information sharing in a two-stage supply chain with production capacity constraints. *Naval Research Logistics (NRL)*, 50(8):888–916, 2003.

[31] A. E. Smith and A. K. Mason. Cost estimation predictive modeling: Regression versus neural network. *The Engineering Economist*, 42(2):137–161, 1997.

[32] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[33] S. Timotheou. A novel weight initialization method for the random neural network. *Neurocomputing*, 73(1):160 – 168, 2009. Timely Developments in Applied Neural Computing (EANN 2007) / Some Novel Analysis and Learning Methods for Neural Networks (ISNN 2008) / Pattern Recognition in Graphical Domains.

[34] B. Tripathy, S. Dash, and S. K. Padhy. Multiprocessor scheduling and neural network training methods using shuffled frog-leaping algorithm. *Computers Industrial Engineering*, 80:154 – 158, 2015.

[35] J. N. Tsitsiklis and B. Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[36] P. Werbos. Beyond regression: New tools for prediction and analysis in the behavior science. *Unpublished Doctoral Dissertation, Harvard University*, 1974.

[37] J. Y. Yam and T. W. Chow. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1):219 – 232, 2000.

[38] G. P. Zhang. Avoiding pitfalls in neural network research. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(1):3–16, 2007.

A Abstract of Dissertation: Reinforcement Learning and the Lead Time Syndrome

Decision makers in industry are confronted with huge amounts of data available through IT systems (e.g. Enterprise Resource Planning). Indeed, there are vast possibilities and savings introduced by the abundance of data and the sophisticated analysis of these data, especially with regard to information sharing in supply chains (see [30]). Although the access to actual and accurate data enables an efficient management of supply chains and its production planning, it may also lead to poor outcomes, especially if human decision makers are involved (e.g., due to biases such as the misperception of feedback; see, [7]). One example that originates from production planning is the so called "lead time syndrome (LTS)". This phenomenon was firstly described by [19] and represents a *vicious cycle* where the observed increase of the duration of orders through a production system (flow time) increases a planning parameter for the order duration (lead time) which leads to perpetually increasing input to the system resulting in increased flow times again.

At the same time machine learning algorithms are becoming more and more popular. Especially the fields of neural network analysis and reinforcement learning have drawn attention to this field of research. E.g. in 2015 Mnih et al. published an algorithm which could beat human players in various computer games [22]. They reported even better results after enhancing the algorithm [21]. Another example is the article by Lillicrap et al. in which they use a continous actions space[18]. These algorithms are based upon inferring statistical similarities by using huge amounts of data and use a neural network underneath.

In this work we show the existing problem of human planners according to the lead time syndrome and develop an algorithm which is able to prevent the lead time syndrome and automatically releases the orders into the production system. To do so we use reinforcement learning which uses an artificial neural network as function approximation. In reinforcement learning an agent chooses an action

A Abstract of Dissertation: Reinforcement Learning and the Lead Time Syndrome

In a nutshell reinforcement learning (Q-learning) algorithms work as follows. An agent explores the environment by consecutively taking an action and observing a immediate reward as it traverses from one state to another environmental state. According to the observed reward and new state the state-action tuple is assessed. This value is then stored before repeating the process. For a tabular representation of the state-action tuples the algorithm converges to the optimal value. However, in practise the extremely high number of state-action tuples prevents using a tabular representation. Therefore, often this function is approximated using a neural network. However it has been shown that this can lead to unstable or even diverging algorithms. [32, 35]

B Data

This section gives the data generated by the experiment for Uniform-, Xavier- and He et al. initialization.

B.1 Uniform Weight Initialization

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
1	618	382	0	382
2	398	602	602	0
3	672	328	26	302
4	611	389	0	389
5	604	396	0	396
6	612	388	0	388
7	379	621	621	0
8	628	372	0	372
9	610	390	0	390
10	618	382	0	382
11	623	377	0	377
12	607	393	0	393
13	585	415	0	415
14	609	391	0	391
15	599	401	0	401
16	628	372	0	372
17	387	613	613	0
18	609	391	0	391
19	615	385	0	385
20	392	608	608	0
21	635	365	0	365
22	400	600	600	0

B Data

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
23	645	355	0	355
24	579	421	0	421
25	628	372	0	372
26	619	381	0	381
27	588	412	0	412
28	603	397	0	397
29	602	398	0	398
30	615	385	0	385
31	619	381	0	381
32	609	391	0	391
33	622	378	0	378
34	612	388	0	388
35	587	413	0	413
36	400	600	600	0
37	607	393	0	393
38	597	403	0	403
39	610	390	0	390
40	602	398	0	398
41	592	408	0	408
42	629	371	0	371
43	582	418	0	418
44	579	421	0	421
45	615	385	0	385
46	627	373	0	373
47	614	386	0	386
48	593	407	0	407
49	417	583	583	0
50	595	405	0	405

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
51	584	416	0	416
52	612	388	0	388
53	611	389	0	389
54	600	400	0	400
55	587	413	0	413
56	605	395	0	395
57	601	399	0	399
58	395	605	605	0
59	618	382	0	382
60	612	388	0	388
61	596	404	0	404
62	636	364	0	364
63	593	407	0	407
64	620	380	0	380
65	593	407	0	407
66	586	414	0	414
67	590	410	0	410
68	598	402	0	402
69	627	373	0	373
70	616	384	0	384
71	634	366	0	366
72	604	396	0	396
73	608	392	0	392
74	609	391	0	391

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
75	605	395	0	395
76	619	381	0	381
77	570	430	0	430
78	592	408	0	408
79	600	400	0	400
80	622	378	0	378
81	594	406	0	406
82	609	391	0	391
83	625	375	0	375
84	601	399	0	399
85	627	373	0	373
86	622	378	0	378
87	593	407	0	407
88	585	415	0	415
89	619	381	0	381
90	610	390	0	390
91	628	372	0	372
92	601	399	0	399
93	609	391	0	391
94	612	388	0	388
95	581	419	0	419
96	618	382	0	382
97	607	393	0	393
98	659	341	95	246
99	627	373	0	373
100	657	343	0	343
	59253	40747	4953	35794

B.2 Xavier Weight Initialization

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
1	607	393	0	393
2	637	363	0	363
3	624	376	0	376
4	568	432	0	432
5	589	411	0	411
6	625	375	0	375
7	605	395	0	395
8	638	362	0	362
9	586	414	0	414
10	601	399	0	399
11	598	402	0	402
12	602	398	0	398
13	623	377	8	369
14	620	380	0	380
15	617	383	0	383
16	611	389	0	389
17	581	419	0	419
18	598	402	0	402
19	588	412	0	412
20	625	375	5	370
21	609	391	0	391
22	607	393	0	393
23	599	401	0	401
24	600	400	0	400
25	590	410	0	410

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
26	609	391	0	391
27	590	410	0	410
28	613	387	0	387
29	635	365	0	365
30	626	374	0	374
31	608	392	0	392
32	592	408	0	408
33	622	378	0	378
34	651	349	0	349
35	615	385	0	385
36	609	391	0	391
37	615	385	0	385
38	574	426	0	426
39	620	380	0	380
40	601	399	0	399
41	596	404	0	404
42	628	372	0	372
43	617	383	0	383
44	602	398	0	398
45	506	494	148	346
46	609	391	0	391
47	610	390	0	390
48	615	385	0	385
49	594	406	0	406
50	592	408	0	408

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
51	602	398	0	398
52	609	391	0	391
53	620	380	0	380
54	587	413	0	413
55	596	404	0	404
56	615	385	0	385
57	614	386	0	386
58	624	376	0	376
59	607	393	0	393
60	598	402	0	402
61	595	405	0	405
62	601	399	0	399
63	601	399	0	399
64	617	383	0	383
65	631	369	0	369
66	628	372	0	372
67	589	411	0	411
68	394	606	606	0
69	592	408	10	398
70	606	394	0	394
71	604	396	0	396
72	586	414	0	414
73	610	390	0	390
74	594	406	0	406
75	648	352	0	352

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
76	610	390	0	390
77	602	398	0	398
78	602	398	0	398
79	595	405	0	405
80	592	408	0	408
81	615	385	0	385
82	612	388	0	388
83	608	392	0	392
84	590	410	0	410
85	604	396	0	396
86	625	375	0	375
87	638	362	0	362
88	603	397	0	397
89	615	385	0	385
90	615	385	0	385
91	610	390	0	390
92	603	397	0	397
93	634	366	0	366
94	576	424	0	424
95	622	378	0	378
96	605	395	0	395
97	618	382	0	382
98	570	430	0	430
99	622	378	0	378
100	622	378	0	378
	60473	39527	777	38750

B.3 He et al. Weight Initialization

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
1	901	99	27	72
2	879	121	101	20
3	924	76	41	35
4	668	332	251	81
5	841	159	148	11
6	609	391	377	14
7	641	359	346	13
8	883	117	89	28
9	893	107	63	44
10	880	120	57	63
11	897	103	77	26
12	632	368	168	200
13	595	405	0	405
14	884	116	74	42
15	716	284	138	146
16	770	230	17	213
17	661	339	320	19
18	872	128	95	33
19	899	101	94	7
20	714	286	11	275
21	883	117	82	35
22	772	228	42	186
23	776	224	32	192
24	894	106	95	11
25	930	70	32	38

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
26	397	603	603	0
27	861	139	52	87
28	889	111	75	36
29	843	157	32	125
30	765	235	26	209
31	777	223	31	192
32	883	117	37	80
33	736	264	26	238
34	742	258	116	142
35	622	378	0	378
36	398	602	602	0
37	863	137	69	68
38	769	231	31	200
39	631	369	0	369
40	385	615	615	0
41	755	245	28	217
42	394	606	606	0
43	728	272	24	248
44	873	127	94	33
45	763	237	51	186
46	765	235	16	219
47	420	580	577	3
48	907	93	63	30
49	618	382	0	382
50	616	384	0	384

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
51	890	110	81	29
52	917	83	45	38
53	752	248	25	223
54	761	239	16	223
55	901	99	66	33
56	899	101	44	57
57	597	403	0	403
58	900	100	41	59
59	865	135	57	78
60	872	128	78	50
61	708	292	9	283
62	707	293	35	258
63	659	341	128	213
64	892	108	82	26
65	762	238	42	196
66	724	276	26	250
67	780	220	158	62
68	874	126	113	13
69	407	593	593	0
70	793	207	25	182
71	891	109	52	57
72	859	141	96	45
73	638	362	343	19
74	916	84	63	21
75	878	122	79	43

Nr	Correct	Incorrect	FalsePositives	FalseNegatives
76	895	105	49	56
77	898	102	76	26
78	881	119	64	55
79	889	111	108	3
80	847	153	64	89
81	410	590	590	0
82	886	114	40	74
83	903	97	84	13
84	596	404	0	404
85	874	126	92	34
86	780	220	178	42
87	894	106	43	63
88	893	107	100	7
89	627	373	347	26
90	758	242	17	225
91	717	283	49	234
92	897	103	28	75
93	910	90	48	42
94	714	286	16	270
95	611	389	0	389
96	903	97	76	21
97	673	327	12	315
98	762	238	42	196
99	636	364	0	364
100	473	527	472	55
	76583	23417	11443	11974