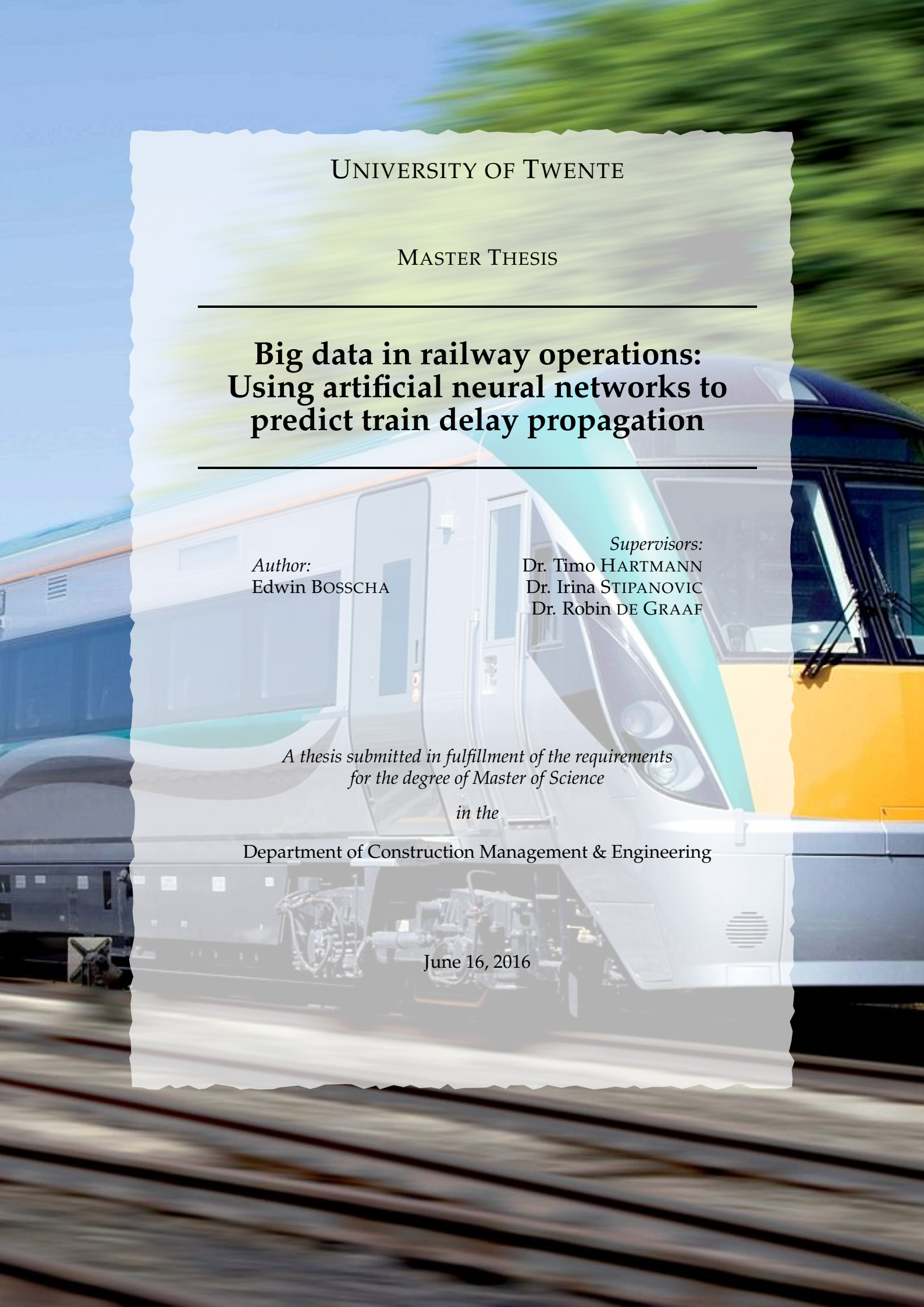

UNIVERSITY OF TWENTE

MASTER THESIS

Big data in railway operations: Using artificial neural networks to predict train delay propagation

Author:
Edwin BOSSCHA

Supervisors:
Dr. Timo HARTMANN
Dr. Irina STIPANOVIC
Dr. Robin DE GRAAF

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Construction Management & Engineering

June 16, 2016

UNIVERSITY OF TWENTE

Abstract

Faculteit Construerende Technische Wetenschappen
Department of Construction Management & Engineering

Master of Science

**Big data in railway operations:
Using artificial neural networks to predict train delay propagation**

by Edwin BOSSCHA

Advances in big data, data collection and machine learning have made it possible to apply new machine learning concepts to a wide array of problems. The aim of this thesis is to explore the possibility of predicting secondary delays in a railway network using a recurrent neural network. This can eventually be used to perform risk analysis and alternative evaluation combined with stochastic delay modelling. Empirical data from Irish Rail is used to test this method and verify the results.
First a RailML data model is constructed containing infrastructure, timetable and rolling stock information based on multiple data sources from Irish Rail. Significant features are identified and extracted from this data model for around 60.000 different delay combinations. Then a sequential approach is used incorporating a recurrent neural network to predict the total knock-on delay. This sequential approach allows for input data in variable lengths, avoiding information loss due to generalization of features. The model is trained with mini-batch gradient descent using the RMSprop algorithm on a large portion of the 60.000 training examples, and validated using the remainder of the example delay combinations.
A coefficient of determination of $R^2 = 0,7029$ is achieved, which is comparable to similar machine learning methods presented in literature. The resulting accuracy is in the same order of magnitude as similar research using support vector machines. While results are less accurate then the results that can be achieved with micro-simulation tools, a series of improvements of the proposed method are presented which might be able to elevate the results of this method to a higher level.

Acknowledgements

I would like to express my deepest gratitude to my supervisors Dr. Timo Hartmann and Dr. Irina Stipanovic for their support and guidance during this research. They've given me the freedom to set up my own research, while guiding me on critical parts. Their useful feedback on the research and reporting aspects have been critical in the execution of this research and the establishment of this thesis. Timo has sparked my interest in the computational side of construction management, and has been and inspiration for this research, as well as a larger part of my education.

I thank Cathal Mangan, Sharon Callanan and Michael Anderson as well as their colleagues at Irish Rail for being incredibly supportive and hospitable in the time I've spent there. They've went above and beyond to provide me with the necessary data required for my research, as well as to introduce me to how everything works at Irish Rail. In addition they've been incredibly pleasant to work with. In just a few months time they've managed to introduce me to many aspects of railway operations and how Irish Rail operates. This research would not have been possible without their support.

I thank Dr. Kenneth Gavin and his colleagues at Gavin and Doherty Geosolutions for providing me with a place to work during my time in Dublin, as well as providing me with initial support with my research. They've immediately welcomed me at their office with open arms, treating me as a valued colleague from the start.

Last of all I owe my gratitude to my family and friends for their support and encouragement during the process. Without you it would not have been possible to finish my master's degree.

Contents

Chapter 1

Introduction

Railway operators have always struggled with the right balance between railway capacity utilization and punctuality. With an increase in traffic density and capacity utilization, the amount of conflicts between trains is bound to increase, resulting in delays called knock-on delays.[1] Knock-on delays are delays which are induced by another train being delayed, instead of being induced by an exogenous event such as infrastructure or mechanical malfunctions. The prediction of these delays is extremely important to create a robust timetable, and for railway operators to be able to able to fully utilize their railway capacity. Aside from the creation of robust timetables, insight in delay causes and the propagation of delays can allow railway operators to manage their maintenance and new investments accordingly. Temporary speed restrictions are not uncommon with railway maintenance works or other works around the rails. Delayed trains are hard to avoid in these situations, but correcting the operational planning can resolve a lot of problems that might occur on busy railway corridors, and can prevent the delays to influence other parts of the railway network.

The prediction of knock-on delays can be applied to two main use-cases. The first is operational prediction, where delays have to be predicted in real-time to improve the capabilities of the railway operator to adjust to occurring delays. Real-time operational prediction of train delays can also be used to improve communication to commuters, by providing more insight in prospected delays in their travels. Real-time prediction of knock-on delays requires the prediction of train-specific delays to be usable for this use-case.

The second use case is simulation and analysis. Knock-on delay prediction can be used combined with stochastic modelling of primary delays to gain insight in the prospected total train delay occurrence for a location or for the total network. This can be transformed into monetary values for risk analysis strategical investment planning, or used to evaluate infrastructure or timetable alternatives in terms of punctuality and monetary loss. The evaluation of alternatives can then be used to find near-optimum solutions to maintenance problems or new investments.[2] This use-case requires a network-total delay minutes as an evaluative measure, and is where the focus of this thesis lies.

With the current rise in big-data, enabled by more advanced sensor systems, better capabilities in IT and data storage, and more computational power to process the accumulated data, new methods of train delay

prediction are now available. Multiple researches have been done using statistical regression to quantify the occurrence of train delays, and the occurrence of knock-on delays. However, most of these consider a single railway corridor with a relatively small sample set. Compared to traditional regressional methods, machine learning algorithms have proven useful in many fields of engineering, including computer vision, speech recognition, video captioning, object recognition and more. Machine learning algorithms have also been used sparsely in train delay prediction, producing results equal or better than those of traditional regressional methods.[3]

While some researches have indicated a correlation between certain input features and the occurrence of knock-on delays in railway operations, all these researches have been limited to single-line operations or to a specific small subset of lines and operational regimes. While this provides some insight in the parameters contributing to the occurrence of knock-on delays, it does not provide a fully generalizable model applicable to multiple scenarios besides the data-intensive micro-simulation method.

Based on established methods and literature, the main issues that can be identified with current methods are limitations in applicability, generalization and accuracy. Micro-simulation based methods often lack in applicability due to the large data overhead and required set-up time. traditional regressional models often lack generalization or accuracy, while they require less input and pre-processing. This leads to believe that there is much room for improvement in the prediction of knock-on delays.

Machine learning and neural networks have been applied in many fields of engineering and data science to be used as an improved regressional or classification model for a wide array of problems. However, the applications in railway operations and analysis are quite limited, not utilizing the full power of modern neural networks, while the possibilities are enormous. Modern neural network architectures and training methods are capable of far better performance in classification and regressional problems, and open up a whole new array of possibilities in computing and data analysis.

Artificial neural networks are algorithms which try to find the optimal transformation between input and output data based on smart learning algorithms. This makes it possible for artificial neural networks to find complex non-linear relationships in data, making it extremely applicable to the prediction of knock-on delays. Theoretically an artificial neural network is capable of representing every possible non-linear function, making them more versatile than multiple regression methods. Furthermore, being trained on purely empirical data allows the neural networks to establish the non-linear transformation between input and output based on actual data instead of theoretical simplifications, allowing them to account for hidden effects that would possibly be missed in analytical models.

This thesis describes a recurrent neural network based approach to predicting first-order knock-on delays in railway networks. Chapter 2

describes the general aspects of railway operations related to this research, after which Chapter 3 describes the basic theory of recurrent neural networks. The research approach is described in Chapter 4, discussing the proposed algorithm. Chapter 5 describes the application of the developed approach to a case study at the Irish Rail network using empirical delay data. Finally in Chapter 6 some conclusions will be drawn regarding the performance of the proposed method and a comparison between other methods described in literature.

Chapter 2

Railway Operations

Railway operations can be defined as the management of running trains throughout a railway network. It can be regarded as the result of infrastructure management, timetabling and other managerial aspects, and can be classified as a complex task. Numerous researches have been devoted to evaluation and optimization in railway operations, leaning on either the infrastructural, timetabling or strategic aspects.

From a basic operational standpoint, trains normally run on a timetable basis, which are designed to run (almost) conflict free. These conflicts however are bound to occur due to exogenous or unforeseen events. When considering a single train line, it can be stated that every train on that line physically takes up a portion of that track, because it is physically impossible for two trains to occupy the same position on a track. In addition to that, a certain distance, both in time and position, should be kept between trains to satisfy some safety conditions. Due to these safety considerations, trains usually have to be authorized to occupy a certain discrete track section. Traditionally, these sections are created by signal blocks or stopping points.

When operating at higher speeds, movement authority must sometimes be gained for multiple sequential track sections to satisfy the safety standards and to avoid unnecessary braking and accelerating. Thus it can be stated that a track section might be blocked longer than that the train actually occupies that track section. When two trains request movement authority for the same track section, or the track section is still blocked when a train requests movement authority, conflicts arise. Traffic controllers can decide to either reroute the train via other track sections, or halt the train until the requested sections are free again.

To evaluate railway operations, delay minutes are often a key indicator. The difference between scheduled and actual operations is deemed as the product of inadequate operations management and unforeseen circumstances. More delay minutes are often related to less reliable railway operations, which is noticeable for everyday railway users. But to use delay as an indicator for railway operations, multiple distinctions can be made. The first distinction is between arrival and departure delays. As the name indicates, the first is the delay when a train arrives at a station, and the latter is the delay when a train departs from a station. While these are highly related, most timetabling strategies incorporate some form of buffer, allowing trains to make up for some lost time.

A second distinction can be made between primary and secondary delays. Primary delays are delays that are caused by external events. Secondary delays, or often called 'knock-on delays', are delays induced by other primary delays. A delayed train can block a path for another train, inducing a knock-on delay for the train that has to wait. Due to this mechanism, delays are able to propagate throughout the railway network and timetable.

As described earlier, trains traditionally operate on an authority basis, which can cause conflicts between trains. While a timetable ideally is built not to have any conflicts, any deviations from the scheduled timetable can still cause conflicts. By increasing the headway between trains, which is the time between two sequential trains to pass a section, the probability of conflicts due to operational deviations is reduced.[4] Another method to prevent delay propagation is to induce buffer times, which can be used for trains to recover from delays. Buffer times mean that a train has more time at a stop than it actually needs. This allows trains with a slight arrival delay to still depart from that station as planned. One major consideration in creating a timetable is the extent to which buffer times and extra headway are available. With both coping mechanisms, the train runs sub-optimal, not utilizing the maximum potential of the infrastructure. However, both mechanisms reduce risk of knock-on delays within the network. This indicates that the timetables used are often capable of dealing with disruptions to a certain degree, depending on the capacity utilization. If a disruption would surpass the built in buffers, knock-on delays are bound to occur, and then will dissipate again based on the available headway and buffers.

2.1 Analysis methodologies

Railway operations classically are analyzed using either statistical regression methods or simulation methods. The first traditionally uses empirical data to establish regressional relationships between network parameters and the occurrence of operational deviations. The latter uses simulation and analytical models to encapsulate the operational logic and determine possible delays, and simulates the railway operations as they occur in a discrete-time manner.

Statistical Regression Statistical regression methods have been widely applied to railway operations and delays. Statistical regression give the possibility to approximate primary and secondary delays based on empirical data. It is the only method available to estimate the occurrence of primary delays based on simple regressional models.[5] Gibson introduces a regressional model estimating knock-on delays for a given primary delays based on regression.

An interesting subset of regressional algorithms are machine learning algorithms, using a large dataset of examples to train a more complex model. The main advantage is that relations do not have to be programmed explicitly, but the algorithm approximates these relations based on the

training data. A more in-depth review of machine learning and neural networks is given in Chapter 3. Machine learning has been applied to many different problems, including transport modeling and delay prediction. Most applications are focused on primary delays, such as the research by Markovic and Milinkovic, predicting primary delay occurences using support vector machines.[6] Robello and Balakrishnan have taken another approach, in his study to model delay propagation in airline networks. By using machine learning to classify the overall delay state of the network, and using clustering to these delay states, they can identify key characteristics for the most critical points in the American National Airspace System.[7] A more micro-simulation based research is proposed by Milinkovic and Markovic, predicting train delays using a neuro-fuzzy interface to encapsulate railway operational behavior.[8]

Lai, Huang and Chu present a comparison between traditional regressional and machine learning applied to railway capacity, related to knock-on delays.[3] Their findings conclude that even a relatively simple and small artificial neural network can outperform traditional regressional methods.

Simulation Simulation of railway traffic can traditionally be split into two methodologies; Micro-simulation and macro-simulation, differing in level of detail and general use-cases. Macro-simulation is generally not used to analyze and predict specific train performance, so it is not discussed further in this thesis.

Micro-simulation is a method where the movement of trains is simulated based on pre-programmed operational logic. It is used to analyze a wide range of operational problems on a microscopic level of detail. [2] A simulation as such requires a large amount of input data such as track layout, signaling locations and logic, speed profiles, timetable and exact rolling stock data. Micro-simulation models can be used to simulate train interactions to attain knock-on delays and recovery time based on stochastic occurrence of primary delays, or to evaluate conflicts within a proposed timetable. Micro-simulation models have proven extremely useful to predict railway capacity, knock-on delays and operational conflicts when all boundary conditions are known.[9, 2] However, parameters such as specific gradient profiles, traction force diagrams or brake performance can be hard to come by for an entire railway network, impeding the use of micro-simulation method for real use cases and fast alternative evaluation.

Chapter 3

Machine Learning

With the increase of available computational power and available data during the last decade, machine learning has been increasingly used as a classification and numerical prediction method. Machine learning is the study of self-learning algorithms using artificial intelligence. A clear distinction can be made between supervised and unsupervised machine learning, where the former maps an input to a predefined output, and the latter finds patterns in unlabeled data. This thesis focuses on supervised learning by using artificial neural networks.

3.1 Artificial Neural Networks

Artificial neural networks(ANN) are algorithms inspired by the human brain. As a machine learning algorithm they can be applied as both a classifier and a regressional algorithm, and form the basis for more advanced machine learning algorithms. Artificial neural networks revolve around a mathematical representation of a biological neuron. Every artificial neuron takes the weighted input of previous neurons, and outputs a single value based on a predefined activation function. While in theory this activation function could be any function, most applications suggest the use of simple functions such as a linear function, sigmoid function, hyperbolic tangent or Gaussian curve.[10]

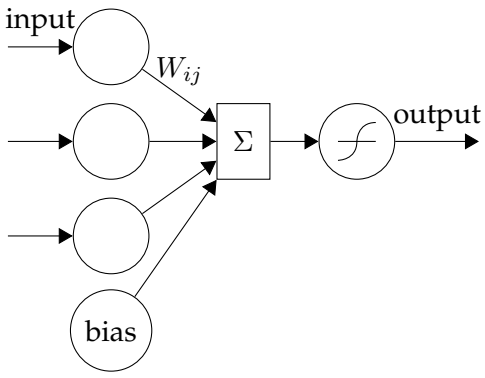

FIGURE 3.1: Model of artificial neuron.

While a single neuron is just a simple transformation function, creating a network of these neurons enables the ANN to reproduce complex non-linear functions. In theory an ANN could be created to represent any function, given it has enough neurons and connections.

The most basic variant of the neural network is a single hidden layer feed-forward neural network. This feed-forward neural network consists of three layers containing N_i neurons. The input layer represents all input variables, where every neuron in the layer outputs a specific input variable. The last layer, or output layer, represents the output values as numerical values. The middle layer, also called the hidden layer, represents a series of transformation functions applied to the input, and outputting the results to the output layer. Usually every neuron in a layer is connected to every neuron in the next layer, creating a fully-connected feed-forward neural network. But virtually any architecture, including more sparsely connected architectures are possible.

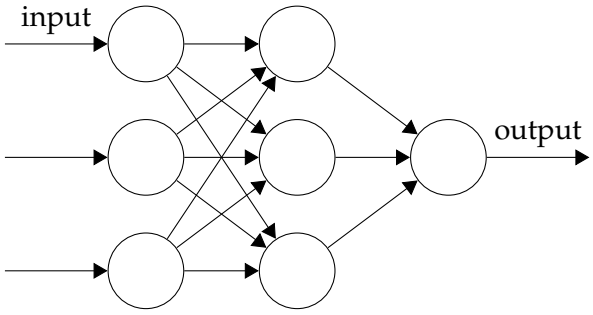

FIGURE 3.2: Feed-forward artificial neural network.

Introducing multiple hidden layers into the ANN enables the algorithm to represent more complex transformation functions. However, the cost of introducing more layers does increase the amount of connections between nodes, resulting in more parameters that have to be optimized in the learning process.

3.2 Recurrent Neural Network

To deal with the fixed-length input vector that is a major restriction for feed-forward neural networks, recurrent neural networks(RNN) have been introduced. RNNs function identically to a normal feed-forward neural network, with the addition of context nodes, which store the output of a previous time-step. By storing and reusing neuron output values in these context nodes, the algorithm is able to factor in data from the previous time-step. RNNs have proven extremely useful at solving sequential tasks such as text interpretation, speech recognition and function approximation. [11]

The simplest form of an RNN is the Elman pattern, where the hidden layer in a single-layer feed-forward neural network feeds its output value back to itself, as shown in Fig. 3.3.[12] This architecture can be extended by stacking recurrent layers, or feeding the output of one recurrent layer to another layer than itself.

The recurrent connections in the RNN allow the neural network to store information as if it has a memory to some degree. While this memory only acts as short-term memory, it can be effective for short sequences

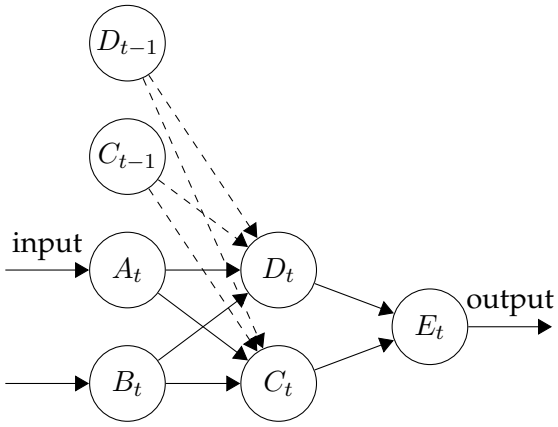

FIGURE 3.3: Elman recurrent neural network pattern. Dashed lines are recurrent connections.

(<50 elements).[13] For longer time-dependencies, gated recurrent neural networks such as long-short term memory(LSTM) neural networks perform significantly better.[14]

The recurrent neurons allow the algorithm to deal with sequential data, but also to deal with a variable length of input sequence. This allows for a many-to-one input-output mapping, where a sequence of data points is responsible for one singular output, instead of a continuous stream of outputs. This type of input-output mapping is depicted in Fig. 3.4.

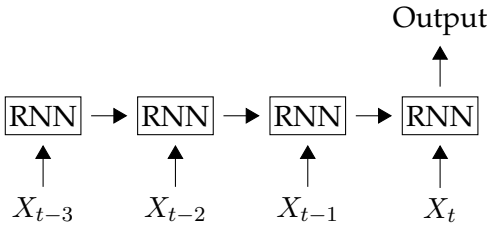

FIGURE 3.4: Many-to-one output mapping.

Le, Jaitlhy and Hinton present several manners to initialize the recurrent connections in an RNN, providing some toy experiments as a reference.[15] Their research suggests that initializing the recurrent weights of an RNN as an identity matrix highly improves performance, beating LSTM-networks on some experiments they provided. Initializing the recurrent weights as an identity matrix means that at the start of the learning process, each recurrent neuron only factors in it's own previous output. During the training process the recurrent neurons slowly add more connections with other recurrent neurons to improve their contextual behaviour.

3.3 Training

Training an ANN is the process where the structure and connections of the ANN are optimized to best predict data the neural network has not seen before. To accomplish this, a training dataset of coupled input and output values is used. If assumed that both the training and unseen data represent

roughly the same phenomenon, finding a network configuration that best represents transformation between the input and output of the training dataset will result in a good predictive capabilities for the unseen data. The ratio to which the algorithm is able to predict unseen data correctly is called generalization.

In most cases, the ANN architecture is defined beforehand, and only the weights of the connections are updated in the training process. The neural network architecture is mostly defined by trial and error, and the best suited architecture is highly dependent on the relationship between input and output data.

When the architecture and activation functions of the ANN are pre-defined, the training process becomes an optimization problem where the values of the connection weights $\vec{W}$ have to be optimized, as described in Eq. (3.1).

$$\min E(\vec{W}) \tag{3.1}$$

For regressional problems the mean square error can be used as the objective function $E(\vec{W})$. The optimization problem is then to minimize the total difference between the predicted and actual values.

$$E(\vec{W}) = \frac{1}{2n} \sum_{p} \sum_{o} (y_{p,o} - d_{p,o})^2 \tag{3.2}$$

The objective function is written out in Eq. (3.2), where p is an index over input-output pairs, o is an index over output values, y is the actual value of that output unit, d is the desired value of that output unit and n is the number of samples in the dataset.

3.3.1 Training Feed-forward neural networks

The most common training algorithm for feed-forward ANN is the backpropagation algorithm, on which most modern gradient-descent methods are based.[16] The method revolves around calculating the output error for each data pair, calculating the contribution of each neuron and weight to this error, determining a gradient based on the contribution to this error and finally updating the weights according to this error. Doing this process for the entire dataset can minimize the error between predicted and ideal output values, minimizing the mean square error for the predictive algorithm.

When considering the objective function stated in Eq. (3.2), the gradient is an array of partial derivatives for each weight. This partial derivative is the sum of the partial derivatives for each input-output pair. The partial derivative to a specific weight can be formulated as Eq. (3.3)

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta s_i} \frac{\delta s_i}{\delta net_i} \frac{\delta net_i}{\delta w_{ij}} \tag{3.3}$$

where E is the error function, s_i is the output of neuron i, net_i is the weighted sum of inputs of neuron i and w_{ij} is the weight between neuron

j and neuron i. These three components can be easily calculated, as $\frac{\delta s_i}{\delta net_i}$ is the derivative of the activation function of neuron i. $\frac{\delta net_i}{\delta w_{ij}}$ equals the output of neuron j. The remaining part of the equation, $\frac{\delta E}{\delta s_i}$, can be calculated by propagating the error back through the network using the chain rule. For the output layer, this is equal to the derivative of the error function, resulting in $\frac{\delta E}{\delta s_i} = y - d$, where y is the actual output and d is the desired output. Once the partial derivative of each weight is known, every weight can be updated according to the following rule:

$$w_{ij}(t+1) = w_{ij}(t) + \epsilon \frac{\delta E}{\delta w_{ij}}(t) \tag{3.4}$$

where ϵ is the learning rate, usually a value between 0 and 1. The value of this learning rate has an important effect on the convergence speed of the algorithm. With a lower learning rate, the algorithm converges slower, while a higher learning rate might cause the algorithm to oscillate, causing the algorithm unable to find a minimum value for the error.

3.3.2 Improved Backpropagation

To increase the effectiveness of the general backpropagation algorithm, many improvements have been proposed, such as the use of a momentum term, turning the weight update rule into

$$w_{ij}(t+1) = w_{ij}(t) - \epsilon \frac{\delta E}{\delta w_{ij}}(t) + \mu \Delta w_{ij}(t-1) \tag{3.5}$$

where μ is the momentum term. This tends to increase the stability of the learning process, and can theoretically help the algorithm bridge shallow regions of the error domain faster. Another often used improvement is the resilient propagation algorithm, which works solely on the sign of the gradients. Resilient propagation was introduced by Riedmiller and Bräun in 1993, and is known to be a significant speedup compared to the traditional backpropagation algorithm in a lot of cases.[17]

While the resilient propagation algorithm is a significant speedup in cases where the weights are updated after calculating gradients for the entire dataset, it loses effectiveness when using batched training, updating the weights after a certain amount of training examples. To overcome this problem, Tieleman and Hinton have introduced root-mean-squares propagation(RMSprop), an algorithm that scales the weight update value with the root mean square of the gradients.[18, 19] This method keeps a moving average of the mean squares of the gradients using Eq. (3.6).

$$MS_t = 0.9 MS_{t-1} + 0.1 \left(\frac{\delta E}{\delta W_{ij}} \right)^2 \tag{3.6}$$

This allows for the weight updates to be scaled individually based on the root mean squares of the gradients, resulting in the weight update equation provided in Eq. (3.7)

$$w_{ij}(t+1) = w_{ij}(t) - \frac{\epsilon}{\sqrt{MS(t)}} \frac{\delta E}{\delta w_{ij}}(t) \tag{3.7}$$

The advantage of RMSprop over regular backpropagation or resilient propagation is the fact that every individual weight update is scaled using the root mean square. Furthermore, keeping the moving average of the gradients allows for a scaling parameter that is not influenced by the use of mini-batches, where resilient propagation is not compatible with mini-batch gradient descent.

3.3.3 Training recurrent neural networks

While RNNs have a lot of similarities with feed-forward ANNs, backpropagation is usually not an effective solution for training a recurrent neural network, because of the dependencies throughout the sequential data. To account for the recurrent nodes in a recurrent neural network, the backpropagation algorithm has to be altered. Werbos has introduced an alteration of the regular backpropagation algorithm which is capable of dealing with these recurrent nodes.[20] The basis of this algorithm is unfolding the sequential recurrent neural network through time, creating a topology resembling a feedforward neural network, as displayed in Fig. 3.5. This allows for the errors to propagate back in time, and update the weights of the recurrent neural network accordingly. Specific weight update rules such as resilient propagation, momentum updates and RMSprop can be used to increase the efficiency of the basic backpropagation through time algorithm.

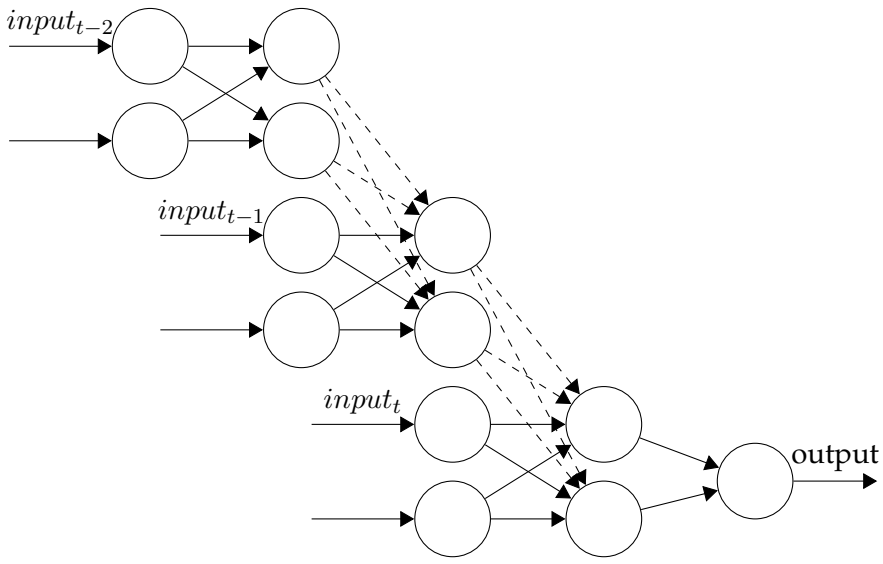

FIGURE 3.5: Schematic representation of unfolded RNN

Unfolding the neural network through time creates a deep neural network, susceptible to the problems occurring in deep neural networks, requiring extra care in the training process. These problems are explained in Section 3.3.4. The general procedure of the basic gradient descent methods is outlined in Algorithm 1.

3.3.4 Known Difficulties

While in theory these training methods are able to find a global optimum for the neural network, several difficulties arise when training

Algorithm 1 Outline of gradient descent learning methods

```
1:  while i < MaxEpochs do
2:      for all Minibatch in Dataset do
3:          for all Sequence in Minibatch do
4:              for all Datapoint in Sequence do
5:                  Calculate network output
6:                  Store neuron activation values
7:              end for
8:              for all Datapoint in Sequence do
9:                  Calculate partial derivatives for weights
10:             end for
11:             Reset recurrent activation values
12:         end for
13:         Update weight vector W⃗
14:     end for
15:     i ← i + 1
16: end while
```

feed-forward- and recurrent neural networks. First of all, the efficiency of the training method is highly dependent on the training parameters of the given training algorithm. Choosing the correct values for the training parameters is essential for the error rate to converge, but the choice is mostly limited to trial and error.

Vanishing gradients

A common problem in training neural networks using gradient descent methods is the vanishing gradients problem, as described by Hochreiter and Pascanu.[21, 13] Due to the error term being propagated through the neural network, and the derivative for most activation functions being lower than 1, the error term diminishes as the error is propagated through the network. This occurs especially in multi-layered structures. This is also a big problem for RNNs, as the backpropagation through time algorithm transforms the RNN into a deep feed-forward neural network. As the error propagates back through time, it is bound to vanish, inhibiting the capability of the network to learn from long-delayed dependencies.

One proposed solution to this known problem is the use of rectified linear units(ReLu). Linear rectifiers are activation functions which limit the output value at 0, creating the nonlinear function $o = max(0, x)$. The advantage of this type of activation function is that the derivative for positive numbers is 1. An added advantage is that ReLu neurons only activate when the input is higher than 0, creating sparser activations for all neurons. [22]

Exploding gradients

Exploding gradients is a problem mainly occurring in RNNs. The standard gradient descent methods are unable to account for a steep wall in the error curvature, which is likely to occur with RNNs. These kinds of walls can

severely disrupt the learning process when not dealt with appropriately, as described further by Pascanu, Mikolov and Bengio.[13] One proposed solution to deal with this problem is the use of a simple clipping function to clip the gradients once they tend to explode. While this does introduce a new parameter into the learning process, it is the simplest working approach to dealing with this problem, and has proven to be effective.[23, 24]

Overfitting

Overfitting is a problem that occurs due to the learning process working too well. When a machine learning model is overfitted, it represents the training data to an extent that is not generalizable to an unseen dataset. This is a huge limitation in machine learning, and regressional analysis overall. A first solution would be to use a second dataset, not used for training as a reference. Once the error rate on this training set goes up, the training algorithm should stop to prevent overfitting. However, this method means that the available data should be split up further, meaning less data is available for training.

A second method to avoid overfitting in neural networks is the dropout method, proposed by Srivastava et. al. This method manipulates the structure of the neural network to train a subsample of the entire neural network for each example. This can be done by randomly dropping out the activation of certain neurons or connections, leading to a randomly sparse neural network for each training example.[25] Due to the subsampling during the training procedure, the full neural network is less likely to overfit the data.

When applied to RNNs, Zaremba has identified some problems with dropout applied to recurrent connections, impairing the network from learning long-time dependencies.[26] They propose dropout to only be applied to the non-recurrent connections in the neural network. However, Moon et. al. propose a different approach to dropout application to recurrent connections, using a constant dropout mask for each sequence. This allows the dropout generalization advantages to be applied to recurrent connections as well.[27]

3.3.5 Alternative training methods

As specified earlier, the training process is in fact a mathematical optimization problem where a vector of weights has to be optimized to minimize an objective function. While the gradient descent methods described earlier usually do a good job at training a neural network, they are prone to getting stuck in local optima and subject to the problems described in Section 3.3.4. However, more general mathematical optimization techniques can also be used to optimize the objective function $E = F(\vec{W})$. General evolutionary algorithms such as the genetic algorithm can be applied, or social search algorithm such as particle swarm optimization. While these methods generally do require a lot more function

evaluations, where a function evaluation is the entire pass over a dataset, they do tend to be better at avoiding local optima.

Particle Swarm Optimization

Particle Swarm Optimization(PSO) is a social optimization method derived from the way groups of animals behave. Particle swarm optimization based methods have proven useful in optimizing non-convex problems, even if dimensionality is high.[28, 29, 30] The algorithm begins by initiating particles randomly within the search space. Each particle has a position, which is the vector of weights. Then a velocity is added to these particles to make them move through the search space. Each iteration, the position of the particle is updated according to this velocity, and the new position is evaluated. For each particle, as well as for the entire swarm, the best positions are being stored. The best performing position of that particle is called the cognitive attractor, and the best performing particle position in the swarm is called the social attractor.

Each iteration the velocity of the particles is updating using a set of simple rules. The velocity is based off three variables: the previous velocity, the cognitive attractor and the social attractor. The amount to which the velocity adheres to each of these variables is dependent on the learning parameters, and defines the swarm behavior. This results in a set of simple equations:

$$v_(t+1) = \mu v_t + A_1(x - a_C) + A_2(x - a_S) \tag{3.8}$$

where x is the particle position, μ is the momentum term, A_1 and A_2 are the cognitive and social learning rates, and a_C and a_S are the cognitive and social attractors. The values for the momentum and learning rate parameters define how well the swarm performs in the given search space. A higher momentum value generally results in more exploration, while a lower momentum value results in a more exploitative behavior. These values usually are set through trial and error, or meta-optimization techniques.

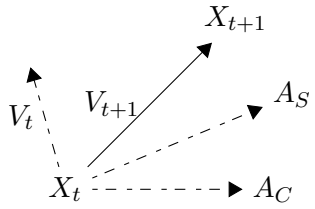

FIGURE 3.6: PSO particle update rule

The advantage of particle swarm optimization compared to the gradient descent methods described earlier is that it is less dependent on the gradient of the error surface. Particle swarm optimization disregards the difference between steep and less steep error surfaces and is capable to deal with non-convex error surfaces quite well. The cost of this performance is the high computational complexity requiring a significant increase in passes over the entire training dataset.

Chapter 4

Delay Prediction Method

In Chapter 1 two use-cases for knock-on delay prediction are presented. The operational prediction use-case requires a per-train approach to knock-on delays to be usable, while the evaluative use-case can be generalized to network-wide knock-on delays. This thesis focuses on the evaluative use-case. To model the knock-on delays occurring as an effect of a singular delayed train, several aspects have to be considered. The first of these is the required and available data that can be used to model this phenomenon. The second aspect to consider is how to use that data to approximate the delay propagation.

4.1 Railway data modeling

As established in Chapter 2, railway operations is a combination of infrastructure management, timetabling and equipment management. To model such a task and its implication, data about all three aspects should be in aggregated, and related to each other. When considering the railway network as a whole, it can be divided up into several routes, which possible intersect at some points. These routes cover a series of tracks and pass several stations, and all have a certain traffic behavior at a certain point at time. But because certain routes might use parts of the same infrastructure, delays can transfer from one route to another due to knock-on delays.

To create a related railway data model, a graph model can be created connecting intersections. Each edge and vertex in this graph can contain data about its respective route or intersection. This data behind each vertex and edge contains all infrastructure and timetable data, as illustrated in Fig. 4.1.

To improve communication between different types of software used in railway operations, the RailML exchange format has been developed.[31] RailML stands for Railway Markup Language, and is an exchange format based on the popular XML language. It is designed to contain infrastructural and operational data, and connecting these by referencing to each other. The basis of the RailML data model lies in three subclasses. These are infrastructure, describing topology, tracks and track elements; Rolling stock, describing all train formation features and timetable, describing the allocation of rolling stock to the infrastructure at a given time.

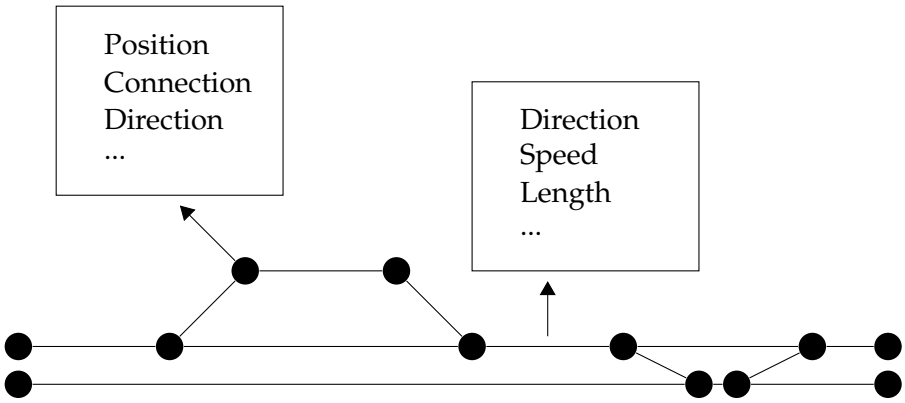

FIGURE 4.1: Illustration of a railway graph data model

The RailML data model permits some flexibility in level of detail, not restricting the user to a specified level of detail. It works by creating operational control points, which are referenced by the timetable module. Operational control points can be any location on the railway network, such as a signal, intersection or station. These operational control points are allocated along one or multiple tracks to define their relative location to other operational control points.

With the operational control points and track layouts known, a graph model can easily be created, where each operational control point is a vertex V. These vertices can be connected based on the track topology specified the RailML file. Each graph edge E represents a route between two connected vertices, containing information about the tracks connecting the vertices.

4.2 Proposed Algorithm

From the basic operational logic described in Chapter 2, we can derive that a transformation happens between the intended timetable and actual timetable once a disruption occurs. This transformation is dependent on infrastructure, rolling stock and timetable aspects, as well as the initial disruption. It is assumed that an artificial neural network would be able to represent this transformation, given the right network architecture and input parameters.

As established in Chapter 2, the first-order knock-on delay is collection of delays that directly result from a train with a primary delay. This means that every train that has a primary delay, or initiator train, can be linked to a value of total resulting first-order knock-on delays D. While it would be possible to approximate D using a feed-forward neural network using generalized features describing the movement of the initiator train, some problems would arise. Because of the fixed-length input requirement of a feed-forward neural network, input data would have to be generalized to a fixed-length input vector. In this feature generalization process, detailed information and nuisances would be lost. The process can be compared to sentence analysis and word recognition, where compression to a single

vector has resulted in limitations of the learning capabilities of the machine learning algorithms. To solve this issue, recurrent neural networks can be used to process variable-length sequences of input vectors, as described in Section 3.2.

When considering a railway graph G, the movement of the initiator train can be divided into a subset of movements between graph nodes. This creates a sequence S of movements m_i over graph edges. In this sequence, every movement m_i can be defined by a feature set F_i, which describe the infrastructure characteristics, rolling stock characteristics, timetable characteristics and the initial train delay. Assuming the premise that D is a result of infrastructure, rolling stock, timetable and initial event is correct, the sequence S with the correct feature set should contain enough information to approximate D.

In Chapter 3 it was established that recurrent neural networks excel at extracting meaning from sequential data with variable sequence length. Using a many-to-one recurrent architecture, an input sequence S can be mapped to an output value D, as displayed in Fig. 4.2.

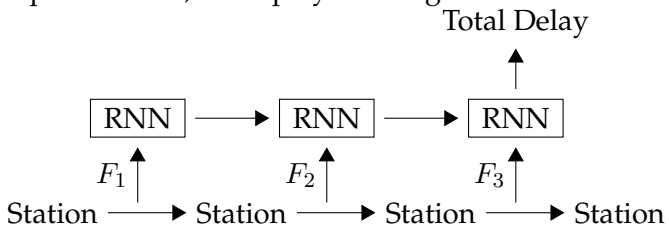

FIGURE 4.2: Schematical representation of RNN-based algorithm

The algorithm uses the input feature set for each point in the sequence, as well as the hidden state of the previous data point to update the new hidden state of the recurrent layer. This recurrent layer state is a vector of size n containing information about the sequence in an n-dimensional feature space. The generalized method is presented in Appendix A

Because of the highly non-linear behaviour, and the better generalization of deep neural networks, a multi-layered architecture is proposed. This allows the neural network to parse the input features to higher level features before the recurrent layers, increasing the potential performance of the algorithm. To solve the vanishing gradients problem for deep neural networks described in Chapter 3, ReLu activation layers are used as the activation function for all hidden layers.

To reduce the chance of overfitting, dropout is applied to each hidden layer as proposed by Moon et. al. and in Chapter 3.[27] Compared to evaluating an extra dataset, which increases computation time per epoch, the hidden and recurrent connections are dropped out with a specific dropout chance, allowing all processor power available to be directed towards gradient calculation during training. The proposed training algorithm is the RMSProp algorithm developed by Tieleman and explained

in Section 3.3.2.[18] This usually allows for faster convergence towards lower error rates and allows mini-batch training to save processing power.

4.3 Input Features

The input features of a neural network are the input variables derived from a dataset that are fed into the neural network. These features are represented as a real number, thus $F_i \in \mathbb{R}$. However, because of the initial weight initialization being semi-random, it is best to choose the input features to be any real number close to the $[-1, 1]$ range. Classes as an input variable can be dealt with by transforming it into a numerical parameter. This is usually done by binary encoding, creating a binary vector for the amount of classes, and allocating a 1 or 0 to the index of that class. For the predictive algorithm, the following input features were used:

Delay Times This is a two-value input, containing both the delay at the origin and the destination point at the respective graph edge. The value is presented in hours to keep it near the $[-1, 1]$ range.

Train Maximum Speed The train maximum speed is derived from the route speed restrictions and the train type. It is presented to the neural network in $\frac{100km}{h}$ to keep the value within the desired range.

Route Maximum Speed The route maximum speed is the general maximum speed for the considered graph edge, corrected for local speed restrictions. It is also presented in $\frac{100km}{h}$

Speed Homogeneity The speed homogeneity is the measure of homogeneity in the maximum speed of the trains passing that edge. It is calculated by calculating the standard deviation of train maximum speeds for each train passing that graph.

Route Departures The route departures is a measure for headway on the considered edge. This is divided into six values where the first three represent trains in the same direction as the initiator train, and the last three values represent trains in the opposite direction. The departure count is split into three values based on the time between the initiator train departure and the other train departure. The first value is the count of trains within 5 minutes difference, the second value for 5 to 15 minutes difference and the third value is for trains with a 15-30 minute difference.

Vertex Departures/Arrivals The vertex departures and arrivals is similar to the route departures feature, but for the origin and destination vertex of the initiator train. The first three values represent the origin vertex, and the last three values represent the destination vertex. The time difference grouping is equal to that of the route departures feature.

Edge Length The route length is the route distance between two vertices. It is calculated using the Dijkstra Algorithm applied to the RailML track definitions. The length is presented in $\frac{Miles}{10}$ to keep it within the desired range.

Percentage Double Track The percentage double track is fraction of distance of the route between two vertices that operates with a double track. This value is close to a binary value for most edges, with a few exceptions.

4.4 Implementation

The entire program including data management is written in C# using the .NET 4.6.1 framework. The recurrent neural network and training algorithm implementations are all based on the Encog 3.3 framework, which allows high customization while providing high-speed basic functions.[32] A custom implementation of the backpropagation through time algorithm was added based on the regular backpropagation code provided by Encog. The custom backpropagation through time implementation was optimized to run in parallel on multiple processor cores to attain the maximum performance out of the available hardware.

The RailML exchange format is converted into C# classes, to allow for greater speed and flexibility in the data manipulation and feature extraction. The RailML model of the infrastructure and timetable is serialized into these C# classes to extract data. A graph model is created from the track topology described in the RailML model. This graph model is then populated with the corresponding timetable for every delay combination. From this populated graph, a data sequence is then created for each delay combination in the training set using the input and output features specified in Section 4.3.

Chapter 5

Case Study

To train and test the algorithm described earlier, a case study has been conducted on the entire Irish Rail network. This chapter discusses some general characteristics of this network, the level of detail used in the data, the data collection process and the training process.

5.1 Network Characteristics

The Irish Rail network consists of a total of 2.400 km of tracks, spread out over the entire Republic of Ireland. It operates both urban and intercity traffic, making it a well diversified network for research purposes. The main lines; Dublin-Cork and Dublin-Belfast, as well as the urban DART service, operate fully on double track, while the rest of the network consists mostly out of single track lines.

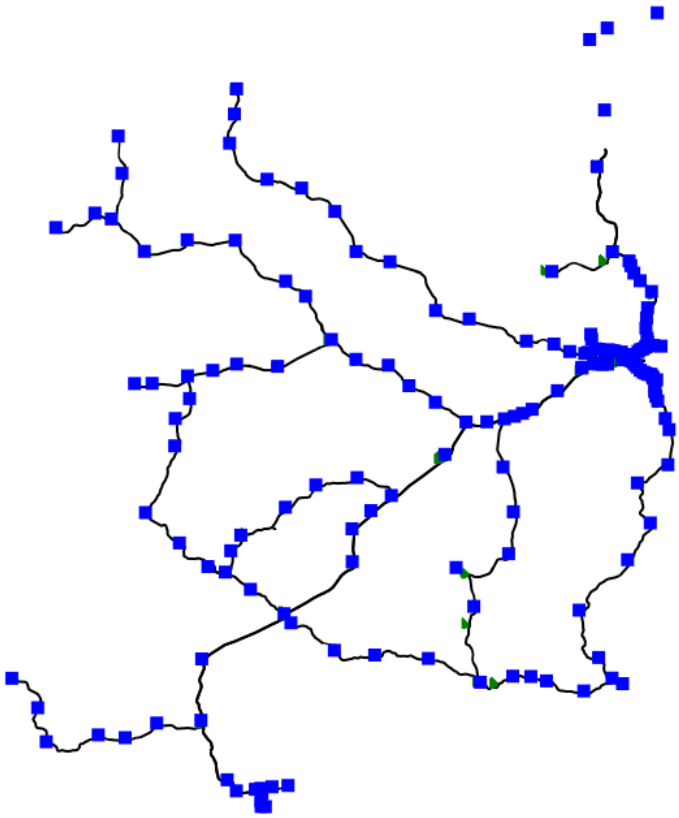

FIGURE 5.1: Topographical map of Irish Rail Network. Blue dots are operational control points.

Between 1-1-2010 and 1-9-2015 an average of 661 trains operate daily on the Irish Rail network, excluding shunting operations and empty train movements. This includes a small amount of freight movements per day, traveling from the Dublin or Rosslare Harbor to various locations throughout Ireland. Maximum operational speeds on the network range from 160 km/h on double tracks to 100 km/h on most single-track lines. Lower speed restrictions are in place in most urban areas or around level crossings.

Due to the Irish Rail network containing mostly single-track lines, the occurrence of knock-on delays are not uncommon. On average the total destination delay due to primary delays is 571.9 minutes per day between 1-1-2010 and 1-9-2015. The average amount of first-order knock-on delays is 93.9 minutes per day. On average 7% of all trains have primary delays higher than 5 minutes, out of which 18% cause a knock-on delay of more than 5 minutes.

5.2 Data Collection

The RailML data model described in Chapter 4 and the delay combinations were all established using databases utilized by Irish Rail. The infrastructure side of the RailML model was created using GIS data from Irish Rail. This data describes a set of infrastructure elements and their respective locations, as either points or polylines. This locational data is mostly based on GPS measurements or drawings. To transform this data into a valid topology, track polylines were connected based on switch locations and proximity. Using a brute-force method, the tracks closest to the switch location were queried, and they were connected using a RailML switch element. To ensure a valid topology, all switch locations were checked beforehand to avoid errors in the topology algorithm. To validate the topology, all track connections were manually checked for consistency. Operational control points such as stations were positioned along the tracks in a similar fashion, using GPS locations and a brute-force approach to find the nearest locations. Once the operational control points and topologies are established, train routes between operational control points can be calculated using the Dijkstra shortest path algorithm.[33]

Timetable and operational data was derived from a system developed in-house by Irish Rail to track train performance. This system registers the proposed timetable for every day, and registers the actual train operations using track circuitry. The system is regulated by delay clerks at the central traffic control, who allocate a delay code to every train delayed more than a certain threshold. This delay code contains information about the source of the delay, and whether it is a knock-on delay or primary delay. The source of the delay is established by the central traffic control as well as communications with the train driver.

The track circuitry based system was introduced gradually in 2009, and is still operational, providing around six years worth of operational data. To

avoid some initial start-up errors, this case study only regards data from 2010.

With the operational data and delay codes available, delay combinations can be easily formed for each primary delay. Trains that have suffered a knock-on delay can be allocated to the delay combination of their respective primary delay. This results in a total of around 60.000 delay combinations between 1-1-2010 and 1-9-2015.

5.3 Training Procedure

As described in Chapter 3 and Chapter 4, the RMSProp algorithm is the fastest and most widely used algorithm to train a similar neural network. Thus it is assumed that RMSProp is the initial best choice of training algorithm for this problem. To solve the problem that gradient descent algorithms are prone to local minima, a clause is built in to switch to particle swarm optimization when RMSProp algorithm gets stuck. Particle swarm optimization is initiated when the mean squared error has not decreased over 20 epochs. The learning rate of the RMSProp algorithm is exponentially lowered, as sensitivity to exploding gradients increases. This is done by introducing the formula $\epsilon_t = 0.98\epsilon_{t-1}$.

The training process is performed for multiple configurations, using different output activation functions, learning rates and other hyper-parameters. Due to the instability of recurrent neural networks and the wide range of output values, many configurations lead to an unstable training process where at some point the weights exploded, causing numerical errors. Due to the high required computational complexity, it was chosen not to optimize the training parameters in a computationally intensive manner. The training parameters were initiated on values suggested in literature for similar studies, after which they were fine-tuned based on which problems exhibited during the training process. From a set of successful training configurations, the best was chosen for further examination. Training this best performing configuration took a total of 17 hours on an AMD A6-3420 processor. All final training parameters for this specific configuration are given in Table 5.1

Parameter	**Abbreviation**	**Value**
Initial Learning Rate	ϵ	10^{-4}
Learning Rate Decrease	r_ϵ	0.98
Number of hidden layers	n_l	2
Number of neurons per hidden layer	n_n	50
Hidden layer activation function	f_{hidden}	ReLu
Output layer activation function	f_{output}	Sigmoid
Maximum weight norm	$\|w\|_{max}$	3
Dropout percentage	P_d	0.2
Weight initialization	w_0	$\mathcal{N}(0, 0.03)$
Batch Size	n_b	50
PSO social learning rate	A_1	2.0
PSO cognitive learning rate	A_2	2.0
PSO number of particles	n_{PSO}	75
PSO maximum velocity	v_{max}	0.2

TABLE 5.1: Training and neural network parameters

5.4 Results

To validate the performance of this method, a validation set of 1750 delay combinations is used. After the training procedure has stagnated, a mean square error for the predicted total knock-on delays in hours of 0.0206 has been obtained on the validation dataset. This is equal to a normalized mean square error of 0.2971, or the coefficient of determination $R^2 = 1 - NMSE = 0.7029$. The mean average error of the validation dataset is $MAE = \frac{\sum |x_i - y_i|}{n} = 0.0722h$. This means that the prediction is off by 4.33 minutes.

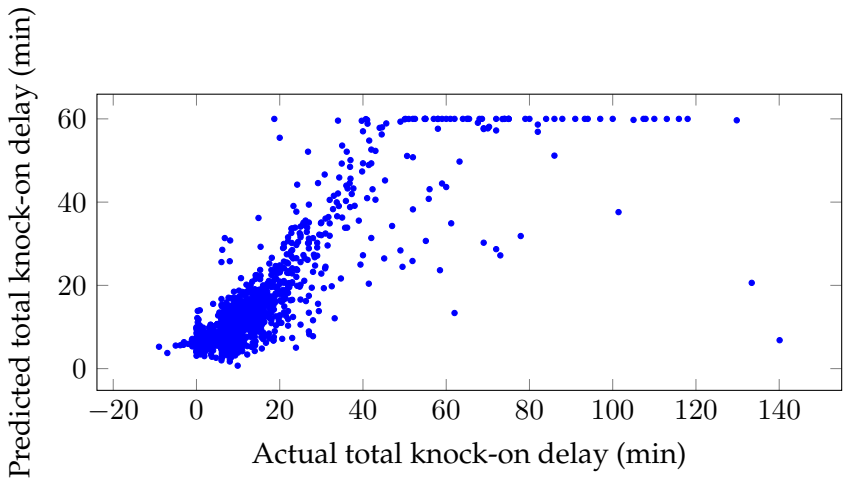

FIGURE 5.2: Histogram of outputs and ideal values

When looking at the individual values produced in the validation dataset, displayed in Fig. 5.2, the correlation between the predicted and actual total knock-on delay can be observed. The clear limitation at 0 and 1 hour where the sigmoid activation function is saturated can be observed. Other output activation functions as well as other output scales were tested, but resulted

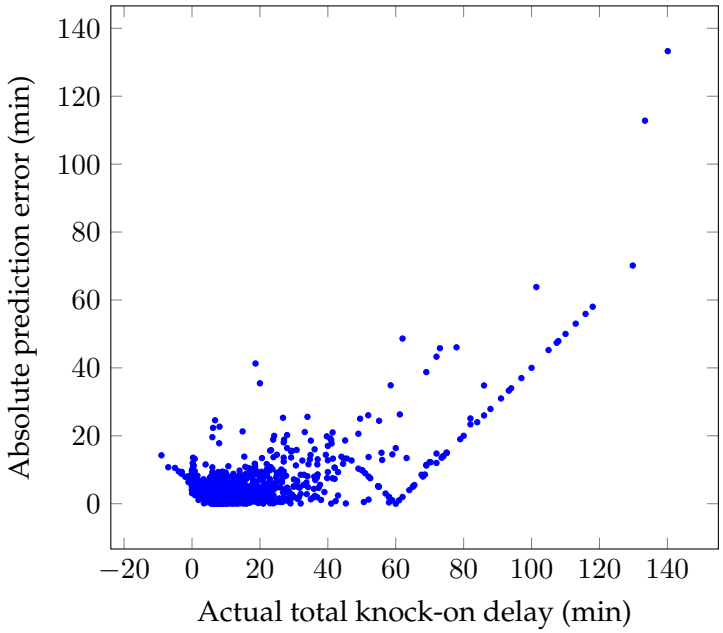

FIGURE 5.3: Histogram of absolute errors in predictions

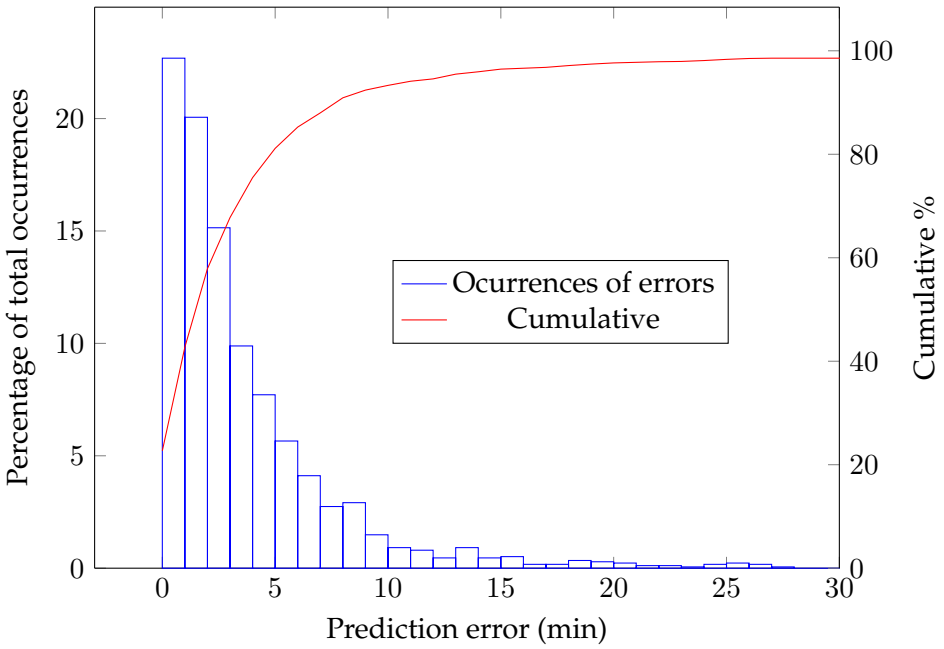

FIGURE 5.4: Occurrences of absolute errors

in an unstable training process or in far worse results. This suggests that an alternative output feature scaling or output activation function should be used for the algorithm to be more efficient at recognizing the wide array of possible output values.

In Fig. 5.3 the absolute prediction error is plotted against the actual knock-on delay for every validation example. This suggests that for total knock-on delays under 60 minutes, the bandwidth of the prediction error is almost equal. For examples with a total knock-on delay higher than 60 minutes, the absolute prediction error rises linearly, suggesting that most

high knock-on delay values are predicted as a 60-minute knock-on delay. Fig. 5.4 displays the occurrences of absolute error values of the predictive algorithm. It can be observed that a total of 1322 out of the 1750 validation examples are predicted within a 5 minute margin, corresponding to a total of 75.5% of the delays that are predicted within this 5 minute margin.

When the outliers with a total knock-on delay of more than 60 minutes are omitted, the corrected coefficient of determination is $R^2 = 0.7082$ over a set of 1698 observations. With these 52 samples omitted from the validation dataset, the mean absolute error is reduced to $MAE = 3.532min$. While this result seems more accurate, it means that the early saturation of the sigmoid output activation function hardly contributes to the coefficient of determination.

Chapter 6

Conclusion

The experiment described in this thesis did not reach a level of accuracy feasible for a real risk analysis of a railway network and timetable. The results did however prove the feasibility of recurrent neural networks in delay prediction and the general application to relational data to some extent. The attained regressional model by a recurrent neural network with two hidden layers results in a similar coefficient of determination as Marković et. al. in their method using support vector regression.[6] The expected increase in performance due to the retained detail in the sequential input data model was not observed. However, this study was conducted using more data examples attained from a complete, diverse railway network. Considering this does indicate better generalization capacity than other similar methods.

When looking at the attained model from an absolute accuracy standpoint, an average error of 4.33 minutes seems reasonably accurate. When comparing this to the study done by Marković et. al. where a MAE of 6 minutes was reported, the results from this experiment are a bit more favorable with a MAE of 4.33 minutes. However, when considering the range of outliers observed in the validation dataset in Fig. 5.3, misclassification is a big issue when looking at the reliability of the model. While the misclassifications observed in the validation dataset are statistically not incredibly significant, they can bias a risk analysis in the wrong direction. When considering the use-cases presented in Chapter 1, where the knock-on delay prediction combined with stochastic delay models are used to evaluate a multitude of alternatives, the wide bandwidth of errors can cause the analysis to favor otherwise highly unfavorable alternatives. This means that the level of confidence for the knock-on delay prediction has to be increased for it to be a usable model. This suggests that an alternative objective function than the mean squared error could prove more efficient for knock-on delay predictions applicable to real use-cases.

It can be argued that the artificial neural network trained with this method is only valid for situations that are almost identical to the examples it is trained on. This means that in order to attain an algorithm that is applicable to more cases, it has to be trained and validated on other situations. While compared to other machine learning based approaches presented in literature, the training data is already diversified, it is still based around the operations of one single railway network with certain characteristics. To extend the algorithm to other situations with differences

in traffic density, spacing between stations or other operational differences, the algorithm should be retrained on examples within that scope.

Compared to micro-simulation of railway operations, the accuracy attained by the recurrent neural network is significantly worse. There is no directly comparable study benchmarking the performance of knock-on delay prediction attained by micro-simulation, but it is assumed that micro-simulation models are capable of predicting train movements within a few minutes accurate.[2] Micro-simulation is also capable of predicting train-specific knock-on delays instead of the aggregated knock-on delays predicted using the recurrent neural network approach.

From a broader operational standpoint, micro-simulation methods are currently capable of providing insight in more operational metrics than the method presented in this paper. Occurrences of specific phenomena such as a train breaking down or maintenance work can be simulated in more detail to provide insight into operational bottlenecks. However, the machine learning approach presented in this thesis, as well as other machine learning applications in railway operations open up a whole new array of possibilities due to the high computational performance compared to micro-simulation methods. With this high computational performance, more alternatives or phenomena can be simulated in the same timeframe compared to micro-simulation methods, allowing for more brute-force oriented methods in timetable and alternative generation and evaluation.

From a machine learning and computing standpoint, it is speculated that the accuracy of this method can be pushed further using newer innovations in machine learning. Most techniques used such as RMSprop and ReLu activation functions have been developed in the last few years, and are still subject to constant improvement from the research community. The same goes for machine learning implementations in frameworks and the possibility of using GPU's (Graphical Processing Units) to speed up the learning process and neural network size. When considering the current speed of advancement made in machine learning and the availability of computational power, it is probable that more detailed regressional analysis will be possible to predict knock-on delays at higher accuracy and level of detail using machine learning. With the correct data and machine learning methods, the theoretical possibilities would supersede the current performance of micro-simulation in both accuracy and applicability, given that machine learning methods keep advancing at the current rate. The best performing artificial neural networks can have millions of connections and weight parameters, where this study only involved around 6000 connections. Theoretically the performance of artificial neural networks using more layers and more connections should improve significantly, suggesting that this method can ideally reach a better accuracy.

An obvious limitation of this study is the absence of signaling data in the training data and implementation. Signal placement, density and operation contributes significantly to the dynamics of knock-on delays, and thus are expected to have a positive influence on the potential prediction accuracy.

6.1 Recommendations

From the proposed artificial neural network, and the case study applied to the Irish Rail network, several recommendations can be done regarding a follow-up research, data management in railway operations and similar machine learning applications.

As described, the proposed method does not attain results accurate enough to be feasible for a realistic prediction yet. But it does demonstrate the potential of deep learning using a rather small and simple neural network. Further research into the application of recurrent neural networks to delay predictions using other input and output features will probably attain better and more accurate results.

Other objective functions could be considered to greatly reduce the bandwidth of errors in the resulting predictions. This will make the method significantly more useful in alternative evaluation, risk analysis and traffic assessment. Finding a solution for the saturation problems at 1 hour delays while also retaining training stability will also highly increase the usability of the predictive model. Increasing the resolution of the machine learning algorithm to signal block size instead of station-based could also improve the capabilities of the algorithm. This would require a higher resolution in historical operational data, which could be achieved by introducing more track circuitry or using GPS tracking on trains. Aside from the implications for this specific machine learning algorithm, more detailed delay data would also contribute to identifying bottlenecks more accurately with empirical data.

From a machine learning perspective, it would be wise to find ways to further push back the computational limitations by utilization of GPU's, dedicated FGPA (Field-programmable gate array) hardware or clusters of processors to unlock the potential of deep neural network structures. One big limitation in this study was the availability of computational power to process the data and calculate the gradients for such a big dataset. The best performing neural networks consist of significantly more layers and neurons per layer, meaning that more detail in information can propagate through the neural network.

A focus towards relational integrated data management instead of using the traditional table-based format would greatly increase the possibilities for other studies such as this one, and the general versatility of the data model. As proven in this study, the use of a relational data model such as a graph model enables for more in-depth studies into the dynamics of railway operations, and more general combinatorial analysis. The current practice to store all information in separate systems with minimal horizontal integration makes the execution of a similar study harder to start up. The possibilities for big data analysis that open up from utilizing a more integrated data model would have enormous potential for issues in asset management, maintenance planning, traffic control and long-term strategic planning.

Appendix A

Algorithm Pseudocode

Algorithm 2 Pseudocode of the machine learning algorithm

for all $DelayCombination$ in Set **do**
    $Sequence \leftarrow \{\}$
    for all $Station$ in $DelayCombination$ **do**
        Calculate input features F from data model for data point
        $Sequence \leftarrow Sequence + F$
    end for
    Calculate total primary knock-on delay D
    $Sequence \leftarrow Sequence + D$
    $DataSet \leftarrow DataSet + Sequence$
end for
while i < MaxEpochs **do**
    for all $Minibatch$ in $Dataset$ **do**
        for all $Sequence$ in $Minibatch$ **do**
            for all $Datapoint$ in $Sequence$ **do**
                Calculate network output
                Store neuron activation values
            end for
            for all $Datapoint$ in $Sequence$ **do**
                Calculate partial derivatives $\vec{G}$ using backpropagation through time
                $\vec{G} \leftarrow \frac{\delta E}{\delta \vec{W}}$
            end for
            Reset recurrent activation values
        end for
        Update weight vector $\vec{W}$
        $\vec{W} \leftarrow \vec{W} + -\frac{\epsilon}{\sqrt{MS(t)}}\vec{G}$
    end for
    $i \leftarrow i + 1$
end while

Bibliography

[1] Tijs Huisman and Richard J. Boucherie. "Running times on railway sections with heterogeneous train traffic". In: *Transportation Research Part B: Methodological* 35.3 (2001), pp. 271–292. ISSN: 01912615. DOI: 10.1016/S0191-2615(99)00051-X.

[2] Hans Sipilä. "Simulation of rail traffic delay modeling and infrastructure evaluation". PhD thesis. 2015.

[3] Yung-Cheng Lai, Yung-An Huang, and Hong-Yu Chu. "Estimation of rail capacity using regression and neural network". In: *Neural Computing and Applications* 25.7-8 (2014), pp. 2067–2077. ISSN: 0941-0643. DOI: 10.1007/s00521-014-1694-x. URL: http://link.springer.com/10.1007/s00521-014-1694-x.

[4] Anders Lindfeldt. "Congested railways". PhD thesis. Stockholm, 2012. ISBN: 9789185539901.

[5] Stephen Gibson, Grahame Cooper, and Brian Ball. "Developments in transport policy: The evolution of capacity charges on the UK rail network". In: *Journal of Transport Economics and Policy* 36.2 (2002), pp. 341–354. ISSN: 00225258.

[6] Nikola Marković et al. "Analyzing passenger train arrival delays with support vector regression". In: *Transportation Research Part C: Emerging Technologies* 56 (2015), pp. 251–262. ISSN: 0968090X. DOI: 10.1016/j.trc.2015.04.004. URL: http://linkinghub.elsevier.com/retrieve/pii/S0968090X1500145X.

[7] Juan Jose Rebollo and Hamsa Balakrishnan. "Characterization and Prediction of Air Traffic Delays". In: 2007 (2014), pp. 1–19.

[8] Sanjin Milinković et al. "A fuzzy Petri net model to estimate train delays". In: *Simulation Modelling Practice and Theory* 33 (2013), pp. 144–157. ISSN: 1569190X. DOI: 10.1016/j.simpat.2012.12.005.

[9] Kariyazaki Keiji, Hibino Naohiko, and Morichi Shigeru. "Simulation analysis of train operation to recover knock-on delay under high-frequency intervals". In: *Case Studies on Transport Policy* 3.1 (2015), pp. 92–98. ISSN: 2213-6258. DOI: 10.1016/j.cstp.2014.07.007. URL: http://dx.doi.org/10.1016/j.cstp.2014.07.007.

[10] Kevin L Priddy; Paul E Keller. *Artificial Neural Networks: An Introduction*. 2005. ISBN: 9780819478726. DOI: 10.1117/3.633187.

[11] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. "Advances in optimizing recurrent networks". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* (2013), pp. 8624–8628. ISSN: 15206149. DOI: 10.1109/ICASSP.2013.6639349. arXiv: arXiv:1212.0901v2.

[12] Cheng Yuan Liou, Jau Chi Huang, and Wen Chie Yang. "Modeling word perception using the Elman network". In: *Neurocomputing* 71.16-18 (2008), pp. 3150–3157. ISSN: 09252312. DOI: 10.1016/j.neucom.2008.04.030.

[13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *Proceedings of The 30th International Conference on Machine Learning* 2 (2012), pp. 1310–1318. ISSN: 1045-9227. DOI: 10.1109/72.279181. arXiv: arXiv:1211.5063v2. URL: http://jmlr.org/proceedings/papers/v28/pascanu13.pdf.

[14] Francoise Beaufays, Hasim Sak, and Andrew Senior. "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling Has". In: *Interspeech* September (2014), pp. 338–342. arXiv: arXiv:1402.1128v1.

[15] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units". In: *arXiv preprint arXiv:1504.00941* (2015), pp. 1–9. ISSN: 1045-9227. DOI: 10.1109/72.279181. arXiv: arXiv:1504.00941v1.

[16] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 0028-0836. DOI: 10.1038/323533a0.

[17] Martin Riedmiller and Heinrich Braun. "A direct adaptive method for faster backpropagation learning: The RPROP algorithm". In: *IEEE International Conference on Neural Networks - Conference Proceedings* 1993-Janua (1993), pp. 586–591. ISSN: 10987576. DOI: 10.1109/ICNN.1993.298623.

[18] Tijmen Tieleman and Geoffrey Hinton. *Lecture 6.5 - rmsprop*. 2012. URL: http://www.cs.toronto.edu/{~}tijmen/csc321/slides/lecture{_}slides{_}lec6.pdf.

[19] Tom Schaul, Ioannis Antonoglou, and David Silver. "Unit Tests for Stochastic Optimization". In: *Iclr* (2013), pp. 1–13. arXiv: 1312.6055. URL: http://arxiv.org/abs/1312.6055$\backslash$nhttp://www.arxiv.org/pdf/1312.6055.pdf.

[20] Paul J. Werbos. "Backpropagation Through Time: What It Does and How to Do It". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. ISSN: 15582256. DOI: 10.1109/5.58337.

[21] Sepp Hochreiter. "Recurrent neural net learning and vanishing gradient". In: *Int. Journal of Uncertainity, Fuzziness and Knowledge-Based Systems* 6.2 (1998), p. 8. ISSN: 0899-7667. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.7389{\&}rep=rep1{\&}type=pdf.

[22] Sachin S. Talathi and Aniket Vartak. "Improving performance of recurrent neural network with relu nonlinearity". In: *Under review of ICLR2016* (2015), pp. 1–11. arXiv: 1511.03771. URL: http://arxiv.org/abs/1511.03771.

[23] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. ISSN: 19410093. DOI: 10.1109/72.279181. arXiv: arXiv:1211.5063v2.

[24] Tomas Mikolov. "Statistical Language Models Based on Neural Networks". In: *PhD thesis* (2012), pp. 1–129. URL: http://www.fit.vutbr.cz/research/pubs/diss.php.cs?id=10158.

[25] Nitish Srivastava et al. "Dropout : A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research (JMLR)* 15 (2014), pp. 1929–1958. ISSN: 15337928. DOI: 10.1214/12-AOS1000. arXiv: 1102.4807.

[26] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. "Recurrent Neural Network Regularization". In: *arXiv:1409.2329 [cs]* (2014), pp. 1–8. arXiv: arXiv:1409.2329v3. URL: http://arxiv.org/abs/1409.2329$\backslash$nhttp://www.arxiv.org/pdf/1409.2329.pdf.

[27] Taesup; Moon et al. "RNNDROP : A NOVEL DROPOUT FOR RNNS". In: (2015), pp. 65–70.

[28] J Kennedy and R Eberhart. "Particle swarm optimization". In: *Neural Networks, 1995. Proceedings., IEEE International Conference on* 4 (1995), 1942–1948 vol.4. ISSN: 19353812. DOI: 10.1109/ICNN.1995.488968.

[29] V.G. Gudise and G.K. Venayagamoorthy. "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks". In: *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)* 2.1 (2003), pp. 110–117. ISSN: 13652125. DOI: 10.1109/SIS.2003.1202255. URL: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1202255$\backslash$nhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1202255.

[30] R C Eberhart and Yuhui Shi. "Particle swarm optimization: developments, applications and resources". In: *Evolutionary Computation, Proceedings of the 2001 Congress on* 1.FEBRUARY 2001 (2001), pp. 81–86. DOI: 10.1109/CEC.2001.934374.

[31] Andrew Nash et al. "RailML - A standard data interface for railroad applications". In: *Computers in railways IX* (2004), pp. 233–240. ISSN: 1462608X. URL: http://trid.trb.org/view.aspx?id=705282.

[32] Jeff Heaton. *Introduction to Neural Networks with C#*. Vol. 99. 2008, p. 430. ISBN: 1604390093.

[33] Luyao Chen et al. "Bidirectional dijkstra algorithm for best-routing of Urban traffic network". In: *Technology* 6754.2007 (2007), pp. 1–8. ISSN: 0277786X. DOI: 10.1117/12.764679.