DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
*STOCKHOLM, SWEDEN 2018*

# A self-normalizing neural network approach to bond liquidity classication

GUSTAV KIHLSTRÖM

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

# A self-normalizing neural network approach to bond liquidity classication

GUSTAV KIHLSTRÖM

Degree Projects in Mathematical Statistics (30 ECTS credits)
Degree Programme in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology year 2018
Supervisors at Kidbrooke Advisory: Björn Bergstrand, Edvard Sjögren
Supervisor at KTH: Henrik Hult
Examiner at KTH: Henrik Hult

*TRITA-SCI-GRU 2018:231*
*MAT-E 2018:39*

Royal Institute of Technology
*School of Engineering Sciences*
**KTH** SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

## Abstract

Bond liquidity risk is complex and something that every bond-investor needs to take into account. In this paper we investigate how well a self-normalizing neural network (SNN) can be used to classify bonds with respect to their liquidity, and compare the results with that of a simpler logistic regression. This is done by analyzing the two algorithms' predictive capabilities on the Swedish bond market. Performing this analysis we find that the perfomance of the SNN and the logistic regression are broadly on the same level. However, the substantive overfitting to the training data in the case of the SNN suggests that a better performing model could be created by applying regularization techniques. As such, the conclusion is formed as such that there is need of more research in order to determine whther neural networks are the premier method to modelling liquidity.

**Sammanfattning**

Likviditeten hos obligationer är komplicerad och ett fenomen som varje obligationsinvesterare måste ta itu med. I den här rapporten undersöks hur pass väl ett själv-normaliserande neuralt nätverk kan användas för att klassifiera obligationer med avseende på deras likviditet, samt jämförs detta resultat med när en simplare logistisk regrsession används. Detta görs genom att analysera de två algoritmernas prediktiva kapacitet på den svenska obligationsmarknaden. Efter genomförd undersökning finner vi att SNN och logistisk regression presterar på liknande nivåer. I fallet med SNN finns dock en stor overfit till träningsdatan, vilket indikerar att en bättre modell möjligtvis skulle kunna nås om vanliga regulariseringsmetoder skulle användas. Slutsatsen blir därmed att det finns behov av mer forskning på ämnet för att dra en konklusion huruvida neurala nätverk är den bäst lämpade samlingen av algoritmer för modellering av likviditet.

# Contents

# Introduction

Fixed income is a type of investment in which the investor receives a predetermined return either periodically or at some set time $T$ in the future. Depending on the type of fixed income financial derivative, the risk associated with the derivative can vary significantly. As an example, a SEKEUR3M (cross-currency) swap, an instrument in which two parties agrees to swap the interest rate attainable in SEK for the interest rate attainable in EUR every 3 months, and a US6M treasury bill (also known as T-bill) in which an investor agrees to pay $X$ to the US Treasury today in return for receiving $X + c$ in 6 months time, clearly have different levels of risk associated with them. Risk associated with fixed income is traditionally divided into two categories: market risk, and credit risk, where (unsurprisingly) market risk is the risk associated with the movements of the market(s) and credit risk is the risk associated with the creditworthiness of the counter party of an agreement. In the case of the currency swap and the T-bill described above, the value of the swap will change if (when) the interest rates attainable in the two currencies change. Thus, the swap is subject to market risk. The T-bill on the other hand will pay $X + c$, as long as the US Treasury does not default. This risk of default also applies to the counter party in the swap-agreement. As such, the T-bill is only subject to credit risk, whereas the swap is subject to both types of risk. Note that in the case of swaps, one party will often pay some extra basis points on top of the actual rate, so that the (expected) cash flows in both directions equal each other at the beginning of the swap's life, and as such the cost of entry into the agreement is zero for both parties.

The view on risk within fixed income presented above is broadly true, but limited. In reality it is more complicated. Market risk is of course not confined to changing interest rates, for example the risk of not being able to liquidate an asset is very real. We call this risk liquidity risk. In the case of bonds, as is the focus of this paper, there is reason to believe that the liquidity is affected by the credit rating of the issuer. This connection between liquidity and credit risk indicates the interconnectedness and complexity of understanding and controlling financial risk.

Given initiatives from the European Banking Authority [1] to regulate the capital requirements of financial institutions based on the level of liquidity of the institutions' assets, there has arisen a greater need for tools to evaluate liquidity. In their literature study on the subject Sommer and Pasquali [2] come to the conclusion that machine learning has potential to tackle this complex problem. The aim of this project is thus to create an algorithm that will be able to classify bonds into different levels of liquidity. This will be done by using machine learning methods on both fundamental and market characteristics. This task is interesting for two reasons; it will bring insights into which parameters/inputs effect the liquidity of the bonds, and perhaps more importantly it will provide a method to classify individual bonds, an action which could be relevant in performing other tasks, i.e. prediction where some or much data is missing. In order to tackle this problem neural networks will be used.

# Background

## Liquidity

There is no precise, generally accepted, definition of liquidity. However, imprecisely liquidity can be described as the ability for an investor to quickly buy (sell) a large quantity of a financial asset at (or close to) the asset's fair value. Liquidity is a major concern for financial institutions in the risk management of their financial liabilities and investments. Research in liquidity stretches back into mid 20th century. In 1968 Demsetz [3] was the first to conjecture the bid-ask spread as a consequence of (a lack of) liquidity and in 1970 Black [4] suggested that liquidity is the price above the fair value of an asset. Grossman and Miller [5] proposed that liquidity is the manifestation of the immediacy of the market (or rather it's participants). Harris [6] built upon this notion and defined liquidity as the width, depth, immediacy, and resilience of the market. In the last few years the importance of liquidity as a driving factor in times of crises has become more apparent (Mehrling [7]). This, has lead to new regulations from institutions such as the EBA and the Basel Committee [1], for example with the introduction of the high quality liquid asset (HQLA) classification. Even so, with or without regulation, liquidity is an issue that any institution-level investor needs to take seriously.

### Measures of liquidity

An obvious problem with working with, and thus regulating, liquidity is that it's not a measurable quantity, but rather needs to be approximated through a proxy. In the research on liquidity many such proxies have been proposed. In fact, the EBA [8], in their appendix *Annex 5: A survey of liquidity metrics*, they present 24 different metrics that aim to approximate liquidity, or at least some aspect of it. We will present some of these here.

Bid-Ask spread,

$$S(t) = p_{ask}(t) - p_{bid}(t), \tag{1}$$

is the difference in lowest asked price and the highest offered price of the bond at time $t$.

Relative spread,

$$S_{mid}(t) = \frac{S(t)}{p_{mid}(t)} = \frac{p_{ask}(t) - p_{bid}(t)}{p_{mid}(t)}, \tag{2}$$

is the bid-ask spread normalized with the mid-price.

Relative spread of log prices,

$$S_{log}(t) = \log(\frac{p_{bid}(t)}{p_{ask}(t)}), \tag{3}$$

is the natural logarithm of the bid price divided by the ask price.

## Drivers of liquidity

The measures mentioned above try to quantify liquidity, but do not explain what it is or what is driving it. The Basel Committee [9], on the other hand, does try to capture the essence of liquidity, where they suggest several characteristics that highly liquid assets possess. These characteristics are divided in two groups; fundamentals and market-associated. The fundamental characteristics are low risk, how easy it is to value the asset, low correlation with risky assets, and whether the asset is listed on a recognized exchange. The market-associated characteristics are whether the market is active or not, low volatility, and if the asset can be a "flight to quality"-asset. Similarily to the Basel Committee's effort, the EBA, as part in the process to define the Liquidity Coverage Ratio (LCR), suggested that the following features could be used to brand bonds as HQLA.

(a) minimum trade volume of the asset

(b) minimum outstanding volume of the asset

(c) transparent pricing and post-trade information

(d) credit rating

(e) price stability

(f) average volume traded and price

(g) maximum bid-ask spread

(h) time remaining to maturity

(i) minimum turnover ratio

Interestingly enough, these two major authorities on risk management for financial institutions only seem to share some of the characteristics of what constitutes a highly liquid bond. In addition, Friewald, et al [10] provides some further, bond specific, metrics affecting the liquidity of a bond, these include the amount issued, maturity, age, rating, bid-ask spread, and volume traded. In summary, there is no consensus in academia on exactly what effects and drives liquidity, however, the sources mentioned do agree on some metrics.

## Value at Risk

A common way in financial mathematics and risk management to measure the risk of some liability or asset that has a non-deterministic future value is to calculate the associated *value at risk* (VaR). Let $X$ be the stochastic asset, then from [11] the value at risk, at level $p$ is calculated as $VaR_p(X) = \min\{m : \mathbb{P}(mR_0 + X < 0) \leq p\}$, where $p \in (0,1)$ and $R_0$ is the discount rate associated with the period in question. Writing $X$ as $X = V_1 - V_0R_0$, the net value of the asset from the perspective of the investor, and letting $L = \frac{-X}{R_0}$ (in situations where a risk-free alternative investment is not applicable, $R_0 = 1$ is used), the VaR can be written as $VaR_p(X) = \min\{m : \mathbb{P}(L \leq m) \geq 1 - p\}$. Given that the distribution function of $L$, $F_L$, is continuous and strictly increasing we get that

$$VaR_p(X) = F_L^{-1}(1-p). \tag{4}$$

**Empirical distribution and quantiles**

Given a set of outcomes, $X = (X_1, X_2, ..., X_n)$, assumed to be independent and identically distributed, the *empirical distribution* is defined as $F_n(x) = \frac{1}{n}\sum_i^n \mathbf{I}(X_i < x)$, i.e. the fraction of points of the seen outcomes that are less than $x$. Mathematically, empirical quantiles are defined just as normal quantiles, $F_n^{-1}(p) = \min\{x : F_n(x) \geq p\}$. However, in the case of empirical quantiles, this can be rewritten as a formula in terms of the seen outcomes $X$. Let $X_{1,n}, X_{2,n}, ..., X_{n,n}$ be a perturbation of the set $X$ such that $X_{1,n} \geq X_{2,n} \geq ... \geq X_{n,n}$, then

$$F_n^{-1} = X_{[n(1-p)]+1,n}, \tag{5}$$

where $[\bullet]$ indicates the integer part of $\bullet$.

**Empirical VaR**

Often time the true distribution of a financial asset's or liability's return is unknown. In these situations it is common to either model the behaviour mathematically (theoretically) or using empirical methods (or a combination of both) in order to quantify the risk analysis. In situations where there is little or no prior information of the behaviour of the phenomenon one is trying understand, but there is historical data available, an empirical analysis is necessary. In such situations in quantitative finance the *empirical VaR*, or $\widehat{VaR}$, approach is often used. In the context of Eq. (4) and Eq. (5), $\widehat{VaR}$ is easily defined. Let $L_i = -\frac{X_i}{R_0}$, and $L_{1,n}, L_{2,n}, ..., L_{n,n}$ be a perturbation of $L_i$ for $i = 1, 2, ..., n$, such that $L_{1,n} \geq L_{2,n} \geq ... \geq L_{n,n}$, then

$$\widehat{VaR}_p(X) = L_{[np]+1,n}. \tag{6}$$

As in the case of normal VaR, in situations where one is modelling risk in which no return is present/expected we let $R_0 = 1$.

## Neural Networks Classifiers

A neural network classifier is a type of machine learning algorithm that tries to map a multi-dimensional input vector to a class. It can be viewed as a function $\mathbb{R}^{d_{real}} \times \mathbb{N}^{d_{cat}} \rightarrow \mathbb{N}^{d_{class}}$, where $d_{real} + d_{cat}$ and $d_{class}$ are the dimensions of the the input and the output respectively. Note that we can have both categorical, here of dimension $d_{cat}$, and continuous, here of dimension $d_{real}$, input to the same neural network.

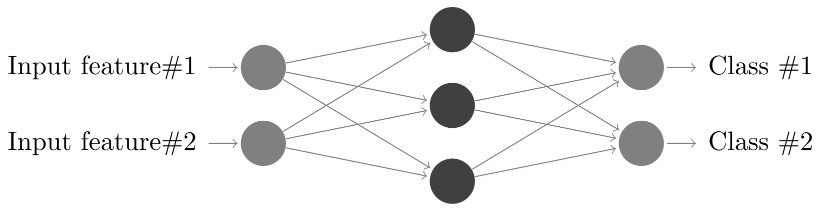


Figure 1: An artificial neural network with 2 inputs and 1 hidden layer, containing 3 nodes.

Neural networks (or rather *artificial* neural networks) get their name from the fact that they are inspired by biological neural networks (i.e. brains), having neurons sending electrical signals to each other in a vast network in order to process information. Figure (1) shows an example of a simple neural network with one *hidden layer* containing 3 nodes, and an input and output space of 2 dimensions. Letting every node in the network represent a neuron, and every edge signal-receptors, the analogy to biology becomes clear. In this example only one hidden layer is used, however, the choice of number of hidden layers as well as the number of nodes in these are arbitrary.

The transitions in Figure (1) can be described mathematically. If we let $x$ be the vector of inputs, the values of the hidden layer will be $a = \sigma(w^\top x + b)$, where $a$ is the vector of values for the hidden layer, $w$ is the weights associated with the first (and in this case, only) ”step” in the model, $b$ is the associated bias, and $\sigma$ is an non-linear *activation function.* The application of $\sigma$ turns the model non-linear, and hence gives it a much greater expressive power. In a more complex neural network, with more layers, $a$ would be propagated through the network and some $\tilde{a} = \tilde{\sigma}(\tilde{w}^\top a + \tilde{b})$ would be calculated, which in turn would be further propagated and so on and so forth, until the end of the network was reached.

### Backpropagation

In order for a neural network to have any predictive power, it needs to be trained on some data set. Training of the model is done via an optimization problem, namely we want to minimize a *loss function*, $L : \mathbb{R}^{N_{Out}} \to \mathbb{R}$, that takes the predicted probability of outputs and the actual outputs and calculates the error, where $N_{Out}$ is the number of classes used in the model. The simplest loss function is the mean square error (MSE); $L(a^{(N_L)}, \mathbf{y}) = \frac{1}{N_{data}N_y}\sum_{j=1}^{N_{data}}\sum_{i=1}^{N_y}(a_i^{(N_L)} - y_i^j)^2$ is used, where $\mathbf{y}^{(j)}$ is the $j$th training data point, $N_y$ is the dimension of the training data (and the last activation layer $a^{(N_L)}$), $N_L$ is the index of the last activation layer, and finally $N_{data}$ is the number of data points. More specifically, the weights, $w^{(l)}$, and biases, $b^{(l)}$, are what needs to be optimized. We start by investigating the weights for some layer $l$, $w^{(l)}$. Since we have that $a^{(l)} = \sigma^{(l)}(w^{(l)}a^{(l-1)} + b^{(l)})$, when using the MSE as loss function we get that the derivative $\frac{\partial L}{\partial w_{j,k}^{l}}$, where $(j,k)$ indicate that this (scalar) weight is the weight between activation $k$ in layer $l-1$ and activation $j$ in layer $l$, is given by;

$$\frac{\partial L}{w_{j,k}^{(l)}} = \frac{\partial z_j^{(l)}}{\partial w_{j,k}^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial L}{\partial a_j^{(l)}}.$$

The first two factors in the right hand side above can readily be computed as $\frac{\partial z_j^{(l)}}{\partial w_{j,k}^{(l)}} = a^{(l-1)}$ and $\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \sigma^{(l)\prime}(z_j^{(l)})$. We leave $\frac{\partial L}{\partial a_j^{(l)}}$ as it is for now. In a similar fashion we find that

$$\frac{\partial L}{b_j^{(l)}} = \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial L}{\partial a_j^{(l)}},$$

where again $\frac{\partial z_j^{(l)}}{\partial b_j^{(l)}}$ is easily computable as $\frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = 1$. Returning to $\frac{\partial L}{\partial a_j^{(l)}}$, we must take two cases into account, one in which $l = N_L$ and one on which $l < N_L$. If, assuming the simple mean square error as loss function, $l = N_L$, or in other words, $a^{(l)} = a^{(N_L)}$ is the last activation layer, then we have that

$$\frac{\partial L}{\partial a_j^{(l)}} = \frac{2}{N}\sum_{i=1}^{N}(a_j^{(l)} - y_j^i). \tag{7}$$

It is a bit more complicated when $l < N_L$. Unlike the weights and biases, each activation (or node in the network) has edges to each of the activations (nodes) in the next layer of the network. As such, when calculating the derivative we need to take these activations into account. Thus we get that,

$$\frac{\partial L}{a_j^{(l)}} = \sum_{i=0}^{n_{(l+1)}} \frac{\partial w_{(i,j)}^{(l+1)}}{\partial b_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial L}{\partial a_j^{(l)}}.$$

The backpropagation algorithm [12] computes the gradient for each activation, weight and bias, and updates these according to some optimization algorithm. Since the gradient activations are ”nested” the method firstly computes the gradient of the last layer activations, stores these values in a memoization matrix, then computes the second-to-last layer dynamically, etc.

### Cross entropy

The cross entropy, $H$, between two probability distributions, $p$ and $q$, defined on the same set of outcomes $\mathcal{X}$, is mathematically defined as $H(p||q) = -\int_{\mathcal{X}} p(x)\log(q(x))dx$. In the discrete case this is easily converted to $H(p||q) = \sum_{\mathcal{X}} p(x)\log(q(x))$. However, since $H$ is a non-negative, convex, function it can be reinterpreted as a cost function. Let $y$ be some true value represented by a vector containing 0s and one 1 (in the index representing the right class) and $a_j^{(N_L)}$ be the $j$th output neuron in a neural network. The cost function can then be defined as $L = -\sum_j \left( y_j \log(a_j^{N_L}) + (1-y_j)\log(1-a_j^{N_L})\right)$. It is clear then that, using a cross entropy loss function Eq. (7) can then instead be written as $\frac{\partial L}{\partial a_j^{N_L}} = -\sum_j \left(\frac{y_j}{a_j^{N_L}} - \frac{1-y_j}{1-a_j^{N_L}}\right)$.

### Softmax

The softmax function, often denoted by $\sigma$, maps an $N$-dimensional vector from $\mathbb{R}^{\mathrm{N}}$ to $[0,1]^N$, such that the sum of the vector components equal one, $\sum_{i=1}^{N} \sigma_i(\hat{x}) = 1$. This means that the outcomes from the softmax function can be interpreted as probabilities. As such, the softmax function is often used in the setting of classification using neural networks, where the last layer of activations of the network uses the softmax function. The last/output layer thus assigns the probability of each class given the input, and the data point is assigned as the class with the highest probability. The $i$th component of the resulting vector of probabilities is given by $\sigma_i(\hat{x}) = \frac{\exp \hat{x}_i}{\sum_{j=1}^{N} \exp \hat{x}_j}$. In order to utilize this for learning in the backpropagation setting the derivative is needed. To simplify notation, let $S_j = \frac{\exp\{a_j\}}{\sum_k \exp\{a_k\}}$, i.e. the $j$th entry in the "probability vector" generated by using the softmax function. Then, $\frac{\partial S_j}{\partial a_j} = \frac{\exp\{a_j\}\sum_k \exp\{a_k\} - \exp\{2a_j\}}{(\sum_k \exp\{a_k\})^2} = \frac{\exp\{a_j\}}{\sum_k \exp\{a_k\}}\big(1 - \frac{\exp\{a_j\}}{\sum_k \exp\{a_k\}}\big) = S_j(1-S_j)$, and analogously $\frac{\partial S_j}{\partial a_i} = -\frac{\exp\{a_j\}\exp\{a_i\}}{(\sum_k \exp\{a_k\})^2} = -\frac{\exp\{a_j\}}{\sum_k \exp\{a_k\}}\frac{\exp\{a_i\}}{\sum_k \exp\{a_k\}} = -S_j S_i$. It is now simple to implement the softmax function into the neural network training setting described earlier.

### Self-normalizing neural networks

Self-normalizing neural networks (SNNs) are a special type of feed-forward neural networks introduced with the aim to reduce overfitting in deeper neural networks. SNNs were developed as a response to the need to specially design a network structure for feed-forward networks with limited overfitting without using standard regularization techniques, such as discussed above. The main difference between an SNN and any other type of feed-forward neural network is the scaled exponential linear unit (SELU) activation function [14].

$$SELU(x) = \lambda \begin{cases} x & if x > 0 \\ \alpha \exp\{x\} - \alpha & if x \leq 0, \end{cases} \tag{8}$$

where $\alpha$ and $\lambda$ are scalar hyperparameters. SNNs use the same idea of keeping weights and biases in the network small in order to prevent overfitting, as weight-penalties and dropout. SNNs, however, achieves this simply by using the SELU function as the activation function for each hidden layer (with the possible exception of the last layer). This works because, given that the mean and variance of the previous activation are within a predefined fixed interval $[\mu_{min}, \mu_{max}], [\sigma^2_{min}, \sigma^2_{max}]$, the SELU activation function will map the mean and variance of the current activation back into these intervals. This means that, given that the first and second moment of the first activation belongs to these intervals, every consecutive first, second moment-pair will also belong to them, which in turn means that all weights and biases will be limited in norm. This leads to a similar effect to that of normal regularization. Using $\alpha = 1.673, \lambda = 1.050$, the intervals become $\mu_{min} = 0.03106, \mu_{max} = 0.06773$ and $\sigma^2_{min} = 0.80009, \sigma^2_{max} = 1.48617$.

## Stochastic gradient descent

As described above, backpropagation provides the logical infrastructure necessary to train a neural network. However, there is still a need to determine precisely how to improve the accuracy of the network, given the calculated gradients, i.e. there is need for a numerical optimization algorithm. The most basic of the optimization algorithms is stochastic gradient descent (SGD). Given a vector of parameters $\theta$, standard gradient descent uses the following update-equation: $\theta_t = \theta_{t-1} - \eta G_{t-1}$, where $G_{t-1}$ is the gradient of $\theta_{t-1}$ based on all data available, and $\eta$ is the *learning rate*, a scalar hyperparameter. Stochastic gradient descent, on the other hand, uses only a fraction of the data to compute the gradient, $\mathbf{E}[G^{(i)}_{t-1}] \approx G_{t-1}$. Here $G^{(i)}_{t-1}$ is the gradient based on a subset of all available data. This approximation of the true gradient is necessary due to the computational time needed when there is a large amount of information to process.

## Adam optimizer

There are however more advanced optimization algorithms than SGD. The adaptive moment estimation algorithm [13], frequently called the Adam optimizer, is a numerical optimization algorithm. As the name suggests, Adam takes the first and second moment into account when calculating the update equation. In doing so, it also uses adaptive learning rates, i.e. it uses the momentum of the trajectory of the values. Let $m_t$ and $v_t$ be the mean and variance estimations at step $t$, and $G_t$ the gradient of the objective function $\theta$ at this step. The update equations for $m_t$ and $v_t$ are $m_t = \beta_1 m_{t-1} + (1-\beta_1)G_t$, $v_t = \beta_2 v_{t-1} + (1-\beta_2)G_t^2$. In order to have unbiased estimations one instead uses $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$, $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$. The final update step is then to set $\theta_t = \theta_{t-1} - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$. Thus a full update step looks as following:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)G_t, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)G_t^2, \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}, \\ \theta_t &= \theta_{t-1} - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \end{aligned} \tag{9}$$

where $\eta$ is the learning rate, $0 < \epsilon << 1$ is used to ensure that the last fraction is well-defined, and $\beta_1$, $\beta_2 \in (0,1)$ are learning parameters.

## Model fit

There are many reasons as to why a machine learning algorithm may fail in modelling data. Two major such categories are over- and underfitting. Underfitting, as the name suggests, is the lack of signal found by the algorithm, i.e. that the algorithm fails to map the input features to the right output. If possible, this is usually mitigated by choosing a different learning algorithm, or in the case of neural networks, the number of hidden layers and nodes could be increased in hope of increasing the explanatory power of the model. Overfitting, on the other hand, is the phenomenon of a model that performs well on a set of inputs but generalizes poorly to other data sets. In machine learning classification one method in order to combat overfitting, is to divide the data in to three subsets; a *training set*, a *test set* and a *validation set*. The algorithm is trained using the training set, and the trained model that performs best on the test set is chosen as the optimal model. The validation set is then used in order to ensure that the model is not overfitted on the test set.

In the setting of neural networks, two common approaches to limiting overfitting is to use *regularization* and *dropout*. Regularization is the technique of adding a function $f(\theta)$ to the loss function, $L + \lambda f(\theta)$, where $\lambda$ is a scalar hyperparameter, and $f(\lambda)$ is a non-negative function based on the weights $\theta$, typically the $L^2$-norm. $f$ serves as a penalty with the aim of preventing any single weight in the network from becoming too large and as such decreasing the risk of the network becoming too reliant on any one neuron. Dropout uses the same idea of keeping all neurons relevant for the predictive ability of the network, however instead of adding a penalty, the dropout technique "drops" neurons, during the learning process, from the network with a probability $p$. This forces the learning to handle cases where some neurons are missing, and in turn keeps the network from being too reliant on any one neuron. According to Klambauer, et al [14] these types of techniques work well on recurrent- and convolutional neural networks, but less so on feed-forward networks.

## Logistic regression

Logistic regression, unlike the name infers, is a model used for binary classification. The model is based on using the sigmoid function to determine the probability of a data point belonging to a class given some input features, much like neural network classifiers. The sigmoid function is defined as $sigm(x|\theta) = \frac{1}{\exp\{-\theta^\top x\}}$, where $x$ is the input features, and $\theta$ are the parameters. In the simplest case $\theta^\top \bar{x} = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$, however increasing the complexity of $\theta^\top x$ increases the expressive power of the model, but also the risk of overfitting to the training data. Once $\theta = \hat{\theta}$ has been trained to fit a data set, new data points, $\tilde{x}$, are classified as belonging to class 1 if $sigm(\tilde{x}|\hat{\theta}) \geq 0.5$, and classified as class 0 if $sigm(\tilde{x}|\hat{\theta}) < 0.5$. The simple binary case can be extended to multi-class classification by using the *all vs. one* logistic regression. In the all vs. one method one binary logistic regression model is trained for each class $j$, such that each model $j$ models the probability of the data belonging to class $j$ or not. Once all models have been trained, new data points are tried on each model and the data is classified as $\max_k\{sigm(\tilde{x})|\hat{\theta}_k\}$, i.e. the model $k$ that generates the highest probability.

# Method

## Experiment

The aim of this thesis is to investigate how well neural networks can capture the liquidity of bonds. To do this a neural network will be trained on some fundamental aspects of bonds in order to classify them by different liquidity measures. Since one of the use-cases of such a classification model is to use the time series of bid/ask/prices of similar bonds for bonds where these are unavailable, any data items based on these will be discarded as input variables. This means that even though for example the bond yield most likely has a significant explanatory power as to predict a bonds liquidity, since the yield is based on the price of the bond it will not be used in this analysis. In order to set the results of the neural network into context, a logistic regression will also be performed and the performance of the fairly complex neural network and the simpler logistic regression will be compared.

## Data preparation

The data is taken from the Swedish bond market. The data is gathered using a Thomson-Reuters Eikon terminal, which has information on circa 4000 Swedish bonds with a maturity after 2018-01-01, and an issue date before 2017-08-30. Out of these only 824 bonds had enough data (most often the bid-ask time series were missing) to perform a relevant analysis. In Table (1) the data points that were downloaded can be seen. The issue date and the maturity date are thus far considered as categorical data points. However, this hides the underlying natural order of dates. As such these are converted to continuous variables in a manner such that the issue date was transformed into the number of days from the issue date to 2018-01-01, and the maturity date is transformed to the number of days from 2018-01-01 to the maturity date. Still, a majority of these data items are of a categorical nature and need to be handled in some way. In this paper, the choice taken is to use one-hot vectors to represent the categorical data. This means that every categorical data item is split into several data items (as many data items as the number of categories in the original data item) that only take on the values 0 or 1. Thus, as an example, a data item $D$ with 3 categories $D \in (0, 1, 2)$ is turned in to 3 data items $D_1, D_2, D_3$, with 2 categories $D_i \in (0, 1)$, such that $D_i = 1 \implies D_{\not{i}} = 0$. In order to keep the input features to similar proportions, the original amount issued and the coupon were both normalized by the largest value in each category.

In addition to the data presented in Table (1), one year (or , if the bond was younger than a year, all available historical values) of daily bid- and ask prices were downloaded for each bond. Using these time series the following liquidity measures were computed: average bid-ask spread, the (empirical) $\mathrm{VaR}_{0.95}$ of the bid-ask spread, the average mid bid-ask spread, and the average log-spread. For

| Data item | Description |
|---|---|
| Currency | The currency that the bond was issued in. |
| Industry Sector Description | A description of the industry sector that the emitting does business in. |
| Coupon | Size of the coupon. |
| Coupon Type Description | Description of bond structure. |
| Maturity date | The date that the bond matures. |
| Is Perpetual Security | If the bond has a maturity date or not. |
| Issuer Rating | S&P long-term rating of the emitting company. |
| Day Count | How time is calculated for coupons. |
| Industry Sub Sector Description | A more precise description of the emitting company's business sector. |
| Issue Date | The date the bond was emitted. |
| Issuer Domicile | The country in which the issuer of the bond is based. |
| Asset Category Description | The type of fixed-income asset. |
| Asset Type | The type of bond. |
| Asset Sub Type | Further classification of the type of bond. |
| Capital Tier | The capital tier under Basel III that the bond falls under. |
| Issue Country Code | Code of the country the bond was issued in. |
| Coupon Class | The high-level type of coupon (e.g. fixed, floating, etc.) |
| Original Amount Issued | Amount originally issued of the bond. |
| Seniority Type | The classification of the bond under the LCR framework. |
| Worst Redemption Event | Worst redemption event (in terms of the investor). |
| Country of Risk | The country in which the risk is taken. |
| Flag Bullet | Flag if the bond is a bullet-bond. |
| Flag Extend | Flag if the bond is extendable. |
| Flag Refundable | Flag if the bond is refundable. |
| Reversed Convertible | Flag if the bond is reversed convertible. |
| Type of Redemption | Expected type of redemption of the bond. |
| Redemption Put | Flag if the bond is puttable. |
| Redemption Call | Flag if the bond is callable. |

Table 1: Data items used in the analysis.

each of these measures the total data set was sorted on the liquidity measure and divided into 2,3,4, and 5 classes of equal size, depending solely on the liquidity measure. I.e. $4 \times 4 = 16$ new (labeled) data sets were created.

## Model setup

These newly created data sets is the data that the algorithms will train and be evaluated on. The neural network that is used is a fully connected self-normalizing neural network with 10 hidden layers. The hidden layers have 500, 400, 300, 300, 350, 250, 125, 75, 30 nodes respectively. Since an SNN is used, all hidden layers have the SELU-activation function, however in the output layer the softmax function is used. In addition the cross entropy function is used as loss-function. The model is built, trained, and tested using Google's machine learning API *TensorFlow*, which handles backpropagation and implementation of numerical optimization. To train the network the Adam Optimizer is used, using $\beta_1 = 1 - \eta$, $\beta_2 = 0.9999$, and $\epsilon = 10^{-8}$. The learning rate $\eta$ is dynamically updated every 50 learning iteration according to the following rule $\eta = 0.99^{i/50}\eta_0$, where $\eta_0 = 0.001$ and $i$ is the current iteration.

The data is shuffled and split into a training set, a test set, and a validation set, composed of 70%, 20%, and 10% of the data set respectively. For each data set the model is then trained using 125000 training iterations. The model that achieves the minimum classification error on the test set during these 125000 iterations is saved, along with the corresponding accuracy rate for the training and validation sets, and the analysis is done on this model.

In addition to this, a logistic regression model is trained. Due to the lack of need of a validation set in the training process for logistic regression models, the data sets is split into two parts, one training set consisting of 70% of the data and a test set consisting of the other 30%. The parameters that is trained is $\theta$ such that $\theta^\top \bar{x} = \theta_0 + \sum_{i=1}^{n} \Big(\theta_{1,i} x_i + \theta_{2,i} x_i^2\Big)$. To reduce overfitting, an $L2$-penalty based on the parameters is added to the loss-function.

## Evaluation

In order to get get an understanding of whether the classification of liquidity in the way as done in this paper is commercially viable or not, a small example evaluation/simulation is made. In this evaluation, an event in which a portfolio of bonds need to be sold (or bought) is simulated and the effect of using the classification, as previously described, compared to the actual values observed in the market. In these kinds of events, where a position is needed to be liquidized quickly, one is generally required to cross the current bid-ask spread, which illustrates the importance of liquidity risk management. The evaluation is performed on the *out-of-sample* sets, i.e. on the validation set in the SNN case and on the test set in the logistic regression setting. The evaluation is performed on every liquidity measure and every output class model. It consist of using the models to classify the liquidity measure of all out-of-sample data points. These bonds are then assigned the mean of the liquidity measure of the corresponding class in the training data set. These assigned values are then subtracted by the actual values. Finally, the mean of these values is presented. Thus it can be summarized as;

$$e = \frac{1}{n} \sum_{i=0}^{n} \big(\hat{s}_i - s_i\big), \tag{10}$$

where $e$ is the evaluation factor, $s_i$ is the actual value of the liquidity measure of bond / data point $i$, and $\hat{s}_i$ is the assigned measure for bond $i$. In the case of using the average bid-ask spread as measure, $e$ becomes the average additional cost of crossing the bid-ask spread for the portfolio of bonds, from using the model as opposed to the actual values. When using the bid-ask spread $\mathrm{VaR}_{0.95}$ measure, this evaluation amounts to the average additional cost of holding the $\mathrm{VaR}_{0.95}$ based on the models classifications compared to reality. In the case of the log-spread we change the $s_i$ and $\hat{s}_i$ in Equation (10) to $\exp\{s_i\}$ and $\exp\{\hat{s}_i\}$. From Equation (3) we see that $\exp\{s\} = \exp\{\log(\frac{P_{bid}}{p_{ask}})\} = \frac{P_{bid}}{p_{ask}}$, and as such the evaluation on the log-spread measure becomes the average difference in the estimated and the actual bid-ask quotient. Finally, for the relative bid-ask spread, which is less tangibly interpretable, the evaluation simply becomes the average difference in the estimated relative spread and the actual relative spread.

## Results

### Self-normalizing Network

Tables (2) - (5) shows the results obtained by training and evaluating the SNN described in the Method section.

| Classes | Training Accuracy | Test Accuracy | Validation Accuracy |
|---|---|---|---|
| 2 | 0.99830794 | 0.8816568 | 0.8352941 |
| 3 | 0.9915398 | 0.8165681 | 0.67058825 |
| 4 | 0.9915398 | 0.74556214 | 0.64705884 |
| 5 | 0.98815566 | 0.6745562 | 0.5411765 |

Table 2: Results obtained by classifying using the SNN on average bid-ask spread.

| Classes | Training Accuracy | Test Accuracy | Validation Accuracy |
|---|---|---|---|
| 2 | 0.97461927 | 0.80473375 | 0.7647059 |
| 3 | 0.9966159 | 0.7928994 | 0.69411767 |
| 4 | 0.97969544 | 0.6449704 | 0.5882353 |
| 5 | 0.6497462 | 0.591716 | 0.5411765 |

Table 3: Results obtained by classifying using the SNN on bid-ask $\text{VaR}_{0.95}$.

| Classes | Training Accuracy | Test Accuracy | Validation Accuracy |
|---|---|---|---|
| 2 | 0.99492383 | 0.8994083 | 0.8117647 |
| 3 | 0.99492383 | 0.8165681 | 0.63529414 |
| 4 | 0.99323183 | 0.73964494 | 0.63529414 |
| 5 | 0.98307955 | 0.6804743 | 0.5411765 |

Table 4: Results obtained by classifying using the SNN on the average relative bid-ask.

| Classes | Training Accuracy | Test Accuracy | Validation Accuracy |
|---|---|---|---|
| 2 | 0.99319726 | 0.875 | 0.85705883 |
| 3 | 0.99489796 | 0.79761904 | 0.7176471 |
| 4 | 0.99319726 | 0.74404764 | 0.7294118 |
| 5 | 0.9880952 | 0.7083333 | 0.63529414 |

Table 5: Results obtained by classifying using the SNN on average log bid-ask.

## Logistic Regression

Tables (6) - (9) shows the results obtained by training and evaluating the logistic regression model described in the Method section.

| Classes | Training Accuracy | Test Accuracy |
|---|---|---|
| 2 | 0.876480 | 0.889763 |
| 3 | 0.805414 | 0.791338 |
| 4 | 0.739424 | 0.685039 |
| 5 | 0.690355 | 0.543307 |

Table 6: Results obtained by classifying using the logistic regression on the average bid-ask spread.

| Classes | Training Accuracy | Test Accuracy |
|---|---|---|
| 2 | 0.822335 | 0.775590 |
| 3 | 0.744500 | 0.767716 |
| 4 | 0.663282 | 0.637795 |
| 5 | 0.661590 | 0.507874 |

Table 7: Results obtained by classifying using the logistic regression on the average bid-ask $\text{VaR}_{0.95}$.

| Classes | Training Accuracy | Test Accuracy |
|---|---|---|
| 2 | 0.886632 | 0.846456 |
| 3 | 0.820642 | 0.759842 |
| 4 | 0.761421 | 0.669291 |
| 5 | 0.680203 | 0.535433 |

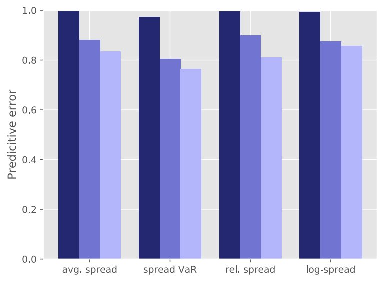
Table 8: Results obtained by classifying using the logistic regression on the average relative bid-ask.

| Classes | Training Accuracy | Test Accuracy |
|---|---|---|
| 2 | 0.884907 | 0.826771 |
| 3 | 0.813874 | 0.779527 |
| 4 | 0.751269 | 0.673228 |
| 5 | 0.692047 | 0.539370 |

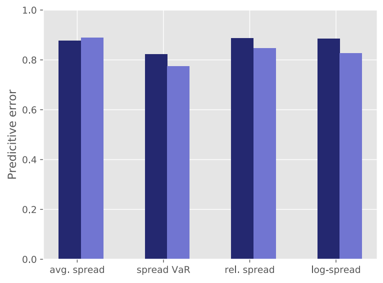
Table 9: Results obtained by classifying using the logistic regression on the average log bid-ask.

## Comparative results

For the following figures a dark blue column represents training accuracy, a blue column represents test accuracy, and a light blue column represents validation accuracy. Figures (2) - (5) displays bar charts over the performance of the different measures grouped by the number of classes.

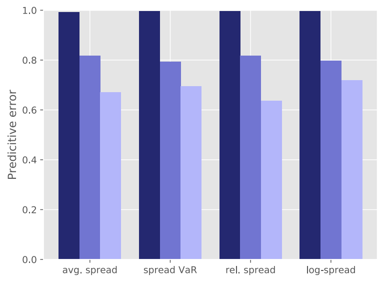


(a)

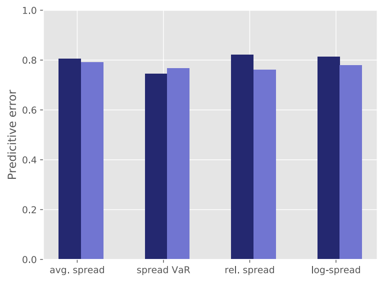


(b)

Figure 2: Bar chart over results using 2 liquidity classes, using (a) SNN and (b) logistic regression.

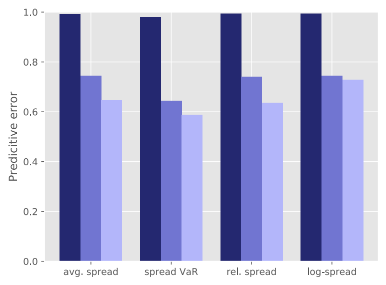


(a)

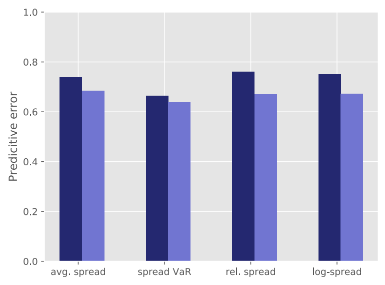


(b)

Figure 3: Bar chart over results using 3 liquidity classes, using (a) SNN and (b) logistic regression.

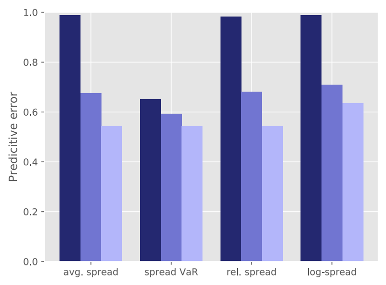


(a)

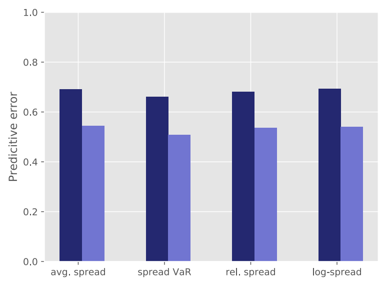


(b)

Figure 4: Bar chart over results using 4 liquidity classes, using (a) SNN and (b) logistic regression.



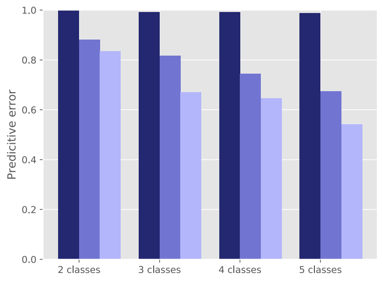
(a)



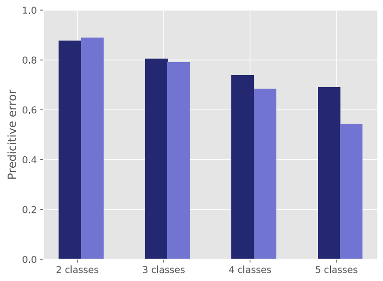
(b)

Figure 5: Bar chart over results using 5 liquidity classes, using (a) SNN and (b) logistic regression.

Figures (6) - (9) displays bar charts over the performance of the different number of classes grouped by the liquidity measure.

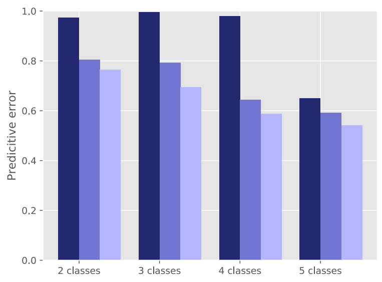


(a)

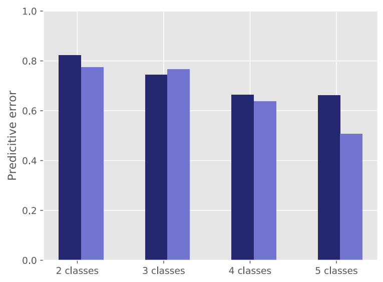


(b)

Figure 6: Bar chart over results using the average bid-ask spread as liquidity measure for different number of classes, using (a) SNN and (b) logistic regression.

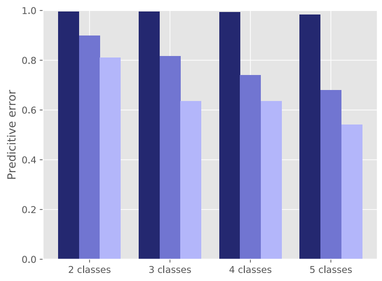


(a)

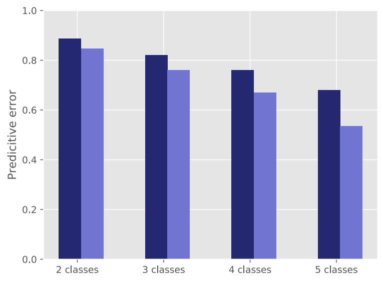


(b)

Figure 7: Bar chart over results using the bid-ask $\mathrm{VaR}_{0.95}$ spread as liquidity measure for different number of classes, using (a) SNN and (b) logistic regression.

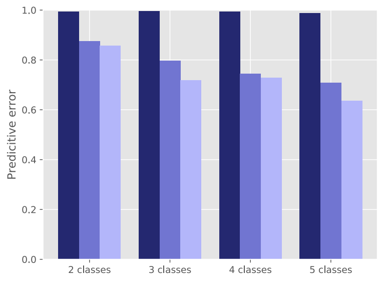


(a)

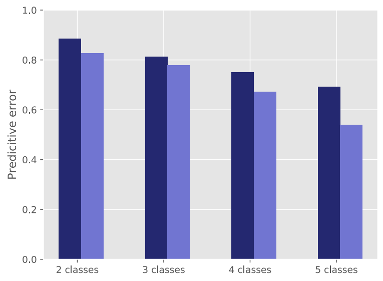


(b)

Figure 8: Bar chart over results using the relative bid-ask spread as liquidity measure for different number of classes, using (a) SNN and (b) logistic regression.



(a)



(b)

Figure 9: Bar chart over results using the log bid-ask spread as liquidity measure for different number of classes, using (a) SNN and (b) logistic regression.

Tables (10) and (11) shows the results from evaluation methods described in the Method section.

| Measure | 2 classes | 3 classes | 4 classes | 5 classes |
|---|---|---|---|---|
| Avg. spread | 0.10952 | -0.063831 | 0.0832397 | 0.075179 |
| Spread VaR$_{0.95}$ | 0.109554 | 0.018493 | 0.033545 | 0.086480 |
| Rel. spread | 0.001394 | 0.015139 | 0.000717 | 0.000051 |
| Log-spread | -0.1458285 | -0.0952148 | 0.0213707 | 0.1467561 |

Table 10: Results of the evaluation of the SNN models.

| Measure | 2 classes | 3 classes | 4 classes | 5 classes |
|---|---|---|---|---|
| Avg. spread | -0.01680 | -0.09643 | -0.01480 | 0.01072 |
| Spread VaR$_{0.95}$ | -0.07046 | -0.05077 | -0.64550 | -0.00546 |
| Rel. spread | -0.00038 | -0.00450 | 0.00010 | -0.00082 |
| Log-spread | -0.51628 | -0.50933 | -0.37075 | -0.28596 |

Table 11: Results of the evaluation of the logistic regression models.

| Avg. spread | Spread VaR$_{0.95}$ | Rel. spread | Log-spread |
|---|---|---|---|
| 0.012866 | 0.080301 | 0.001110 | -0.257585 |

Table 12: Results of the evaluation using a one-class-model.

## Discussion

### Self-normalizing neural network

Inspecting Tables (2) - (5), it is clear that there are certain patterns that span all liquidity measures. As one would have thought, the accuracy generally decreases with an increasing number of output classes. The exceptions to this rule are some of the training set results, in which the accuracy is high for all classes and liquidity measures. Hence, small fluctuations can cause the training set accuracy of a model with more output classes to be higher than that of a model with fewer. It is clear that the model with best performance, in respect of the accuracy, are the models built upon the log-spread liquidity measure. By simple inspection, it is however difficult to quickly order the performances of the average bid-ask spread, the relative spread, and the spread $\mathrm{VaR}_{0.95}$ measures.

Since the outcomes are discrete it is easy to benchmark the models. A model with only two output classes that performs better than a 0.5 error rate would be deemed to have found a signal. If the model achieved an accuracy rate around 0.5, it would be equal to simply guessing, and an error rate less than 0.5 would be performing worse than a monkey throwing darts at a dartboard. Obviously, the corresponding rates for a 3, 4, and 5 class model would be 0.33.., 0.25, and 0.2 respectively. The validation column of Tables (2) - (5) suggests that every single class - measure combination performs at a rate significantly better than this theoretical, no-signal, benchmark level.

The overall best performing model is the 2 class model predicting the bid-ask log-spread, having a validation error rate of approximately 0.86. The second best 2 class model was the one predicting the average bid-ask spread, followed by the relative bid-ask spread, and finally the bid-ask spread $\mathrm{VaR}_{0.95}$ spread. In the 3 class case, the log-spread model also performed the best, followed by the bid-ask spread $\mathrm{VaR}_{0.95}$, then the average bid-ask spread, and the relative spread. The 4 class performance order is identical to the 2 class case. Lastly, for the 5 class case, the log-spread once again has the best performance. The other liquidity measures had the same accuracy on the validation set, meaning they managed to correctly classify exactly the same amount of data points. This ordering is summarized in Table (13).

| | 2 classes | 3 classes | 4 classes | 5 classes |
|---|---|---|---|---|
| 1 | log-spread | log-spread | log-spread | log-spread |
| 2 | average spread | spread $\mathrm{VaR}_{0.95}$ | average spread | |
| 3 | relative spread | average spread | relative spread | |
| 4 | spread $\mathrm{VaR}_{0.95}$ | relative spread | spread $\mathrm{VaR}_{0.95}$ | avg. /rel. / $\mathrm{VaR}_{0.95}$ |

Table 13: Results using SNN, ordered by performance.

By inspecting the tables, it is clear that there is a significant difference in the accuracy rate between the training accuracy and the test accuracy of all trained models. The training accuracy for most models are at a level of 0.95 or better, whereas the test accuracy for most models are within the span of $0.6 - 0.9$. This indicates that the models suffer from overfitting to the training data, and hence does not generalize to the test data. It is also noticeable that the same phenomenon occurs when looking at the test error in respect to the validation error. The difference between these error are, however, smaller; in general having a difference of about $0.05 - 0.15$. As such there is also some overfitting with respect to the test data. These tables also show that the test error grows at a much higher rate than the training error as more output classes are added, which is evidence that the higher number of output class models are more prone to overfitting. There does not seem to be a similar pattern for the test / validation accuracies.

## Logistic regression

Tables (6) - (9) display the results gained by training and and testing the logistic regression model on the data. By inspecting these tables one realizes that there are some similarities to the results obtained by the SNN models. The tables show that, much like in the case of the SNN, that the error grows with the number of classes the model outputs, which of course is to be expected. Also as in the case of SNN, every model manages to find a signal such that the training error is better than randomly choosing a class as classification. These results differ, however, in terms of the relative performances. The measure that has the greatest overall performance is, unlike the SNN case, the average bid-ask spread, which has the highest accuracy for all number of output class cases. For the 2 through 5 class models, the spread $\text{VaR}_{0.95}$ measure places fourth, third, fourth, and fourth respectively and hence is the by far worst performing measure. The relative spread correspondingly places second, fourth, third, and third. The log-spread, finally, places third, second, second, and second; generally performing better than the relative spread. These ordered results are summarized in Table (14).

| | 2 classes | 3 classes | 4 classes | 5 classes |
|---|---|---|---|---|
| 1 | average spread | average spread | average spread | average spread |
| 2 | relative spread | log-spread | log-spread | log-spread |
| 3 | log-spread | spread $\text{VaR}_{0.95}$ | relative spread | relative spread |
| 4 | spread $\text{VaR}_{0.95}$ | relative spread | spread $\text{VaR}_{0.95}$ | spread $\text{VaR}_{0.95}$ |

Table 14: Results using logistic regression, ordered by performance.

According to the results, for most liquidity measures, the difference in training accuracy and test accuracy is less than 0.1, with most lying in the interval of $0.04 - 0.1$. The liquidity measure with the overall lowest difference, and hence lowest overfitting, seems to be the relative spread; having the models with 2 and 3 output classes obtaining a greater accuracy on the test set than on the training set, while the model with 4 and 5 output classes performs just slightly worse on the test set than on the training set.

## Model comparison

In comparing the performance of the SNN models and the logistic regression models, it is not entirely unambiguous as to which accuracy rates one should compare. Since the logistic regression models are easier (in terms of finding the minimum) to train than the SNN, there is no need for a validation set, due to the fact that the model is not optimized for the test set in the same was as the SNN models are. To avoid any boosted results due to comparing overfitted results from the SNN models with out-of-sample results from the logistic regression models, in this analysis we compare the test set results of the logistic regression with the results from the SNN validation set.

From inspection of Figure (2) - (9) it becomes clear that the SNN model consistently, across all liquidity measures and all number of output classes, obtains a better performance on the training set than the logistic regression models. However, when comparing the SNN validation set performance with the logistic regression test set performance, there is no such easy pattern; with the logistic regression model having a greater accuracy for all numbers of output classes using the average bid-ask spread, but only the 3 class output case for the log-spread models. In the case of the spread $\mathrm{VaR}_{0.95}$, the logistic regression models outperform the SNN models in the 3 and 4 output class cases. Finally, in the relative spread models, the logistic regression had the greater performances in the 3, 4 and 5 output class cases. These performance results are summarized in Table (15).

| Classes | bid-ask spread | spread $\mathrm{VaR}_{0.95}$ | relative spread | log-spread |
|---|---|---|---|---|
| 2 | ✶ | | ✶ | |
| 3 | ✶ | ✶ | ✶ | ✶ |
| 4 | ✶ | ✶ | ✶ | |
| 5 | ✶ | | | |

Table 15: Schema over the measure - class pairs that the logistic regression model outperformed the SNN model.

By looking at Figures (2) - (9) it is evident that the SNN models suffer more from overfitting than that of the logistic regression models. In fact, there is a significant drop in accuracy between the SNN training and validation sets, whereas some of the logistic regression models obtain a higher accuracy on the test set than on the training set. These better results on the test set are as opposed to the training set results, of course, only slightly better and an indication that the models do not suffer from any observable overfitting, and generalize well to out-of-sample data. It seems that the overfitting, in the case of the SNNs, increases as the number of output classes increases. One possibility to explain this is that the SNNs perform well on the training set, obtaining an accuracy of around 0.95, and as the number of output classes increases, the difficulty in "guessing" the right class increases as well. However, since there is a similar phenomenon happening with the logistic regression models, albeit on a smaller scale, this could be seen as evidence that the overfitting increases as the model complexity (in this case the number of output classes) increases.

Tables (10) and (11) shows the results from the "simulation"/evaluation described in the Method section. A positive value would indicate that the model on average overestimates the true value, and a negative value indicates an underestimation. By looking at the tables it is clear that the SNN, for most models, overestimate the true value and that the logistic regression models more often underestimate it. Whether an over - or underestimated value is of preference is however unclear, and is highly depending on one's aversion to risk and whether one is on the buying or selling end. Of greater interest is how far from the true value the estimates are, i.e. the absolute value of the values in Tables (10) and (11). These, averaged over class and liquidity measure are presented in Tables (16) and (17).

| Method | bid-ask spread | spread $\text{VaR}_{0.95}$ | relative spread | log-spread |
|---|---|---|---|---|
| SNN | 0.082942675 | 0.062118 | 0.00432525 | 0.102292525 |
| Log. Reg. | 0.0346875 | 0.1930475 | 0.00145 | 0.42058 |
| $\Delta$ | 0.048255175 | -0.1309295 | 0.00287525 | -0.318287475 |

Table 16: Absolute values from Tables (10) and (11), averaged by liquidity measure.

| Method | 2 classes | 3 classes | 4 classes | 5 classes |
|---|---|---|---|---|
| SNN | 0.091674375 | 0.04816945 | 0.0347181 | 0.077116525 |
| Log. Reg. | 0.15098 | 0.17018 | 0.2577875 | 0.07574 |
| $\Delta$ | -0.059305625 | -0.20961805 | -0.2230694 | 0.001376525 |

Table 17: Absolute values from Tables (10) and (11), averaged by number of classes.

In Tables (16) and (17) a positive $\Delta$ indicates that the SNN models had a higher average error and thus that the logistic regression models performed better. The reverse relationship follows analogously. Table (16) indicates that the logistic regression models had a soundly better performance on the average spread and the relative spread, whereas the SNN models performed significantly better on the spread $\mathrm{VaR}_{0.95}$ and on the log-spread. Table (17), on the other hand shows that the SNN models generally had a better performance when averaging by class; obtaining a better performance on the 2 class output models, a significantly better performance on the 3 and 4 output models, and having a just slightly worse performance than the logistic regression models on the 5 output class models.

By looking at Table (12) we can compare the evaluation models with the situation of using a one-class model, i.e. assigning the same estimate of the liquidity measure to all bonds. Using the relative spread, we can see how using the model can effect an investment. If we assume that the cost of a transaction is $\frac{1}{2}(\frac{p_{ask}-p_{bid}}{p_{mid}})$ per bond bought (sold), using the one-class model for a 2000000 SEK investment the additional cost in reality (compared to what the model predicted) would then be $2000000 \times 0.5 \times 0.001110 = 1100$ SEK. Using the 2 class model would amount to a transaction cost of $2000000 \times 0.5 \times 0.001394 = 1395$ SEK, analogously the 3, 4 and 5 class models would yield, 15139 SEK, 717 SEK, and 51 SEK in cost respectively.

## Conclusion

By taking the content of the Results and Discussion section into account, there are a few conclusions to be made. When using the SNN to model liquidity measure, the most accurate measure seems to be the log-spread measure. It is however difficult to determine the order of the remaining liquidity measures in terms of accuracy; it could be argued that the spread $\text{VaR}_{0.95}$ has the poorest performance, but this is not definitive. It is however quite clear that running the SNN algorithm to train a model on this particular data set results in overfitting to the training data. There also seems to be a somewhat overfitting to the test data in respect to the validation data, albeit on a relatively small scale compared to the training data / test data relationship.

Similarly as in the case of the SNN models, there is a best performing liquidity measure for the logistic regression models as well. It is, however, the average bid-ask spread, obtaining the greatest accuracy on all number of output classes. It could once again be argued that the spread $\text{VaR}_{0.95}$ is the worst performing measure, and that the log-spread is better performing than the relative spread. This is however, not obvious. The logistic regression models also, in general, suffer from overfitting to the training data, although this overfitting is significantly less than in the case of the SNN. As such we can draw the conclusion that an SNN model of the depth and breadth used in this paper is much more prone to overfitting to the training data than the logistic regression.

Even though analyzing overfitting is interesting, what is more important form a practical point of view is the performance of the algorithms. Here again, some conclusions can be drawn. The logistic regression models outperform the SNN models for the average bid-ask spread measure. In the case spread $\text{VaR}_{0.95}$ it is unclear which, or if any, of the two models performs better. It could be argued that logistic regression outperforms SNN on the relative spread measure, and that SNN outperforms logistic regression on the log-spread measure. However, when analyzing the performance we are not just interested in the accuracy on the out-of-sample sets, but also the "simulation" of a selling off bonds. Interestingly, when averaging over the number of output classes, the logistic regression models generally obtain a better performance on the simulation, but when averaging over the liquidity measures it is not as clear. The logistic regression models obtain a better performance than the SNN models for the average bid-ask and relative spread, and the reverse is true for the spread $\text{VaR}_{0.95}$.and log-spread.

The evaluation in the relative spread case shows that there could be some value added by using machine learning approaches to model liquidity. Although the 2 and 3 class models performed worse than the one-class model, the 4 and 5 class models had a significant improvement in minimizing the costs.

Having these reflections and conclusions in mind the remaining question is whether it is worth using the complex SNN over the simpler logistic regression when modelling bond liquidity. In the exact setup used in this paper, the time and computational resources needed to train the SNN, combined with the similar or even lesser performance compared to the logistic regression, it clearly is not. With this being said, there are several aspects and parameters that have been left outside the scope of this paper that could effect the results. As such, the final conclusion is that when modelling bond liquidity and a decently working model is sufficient, the SNN is most likely not the best choice. However, if time and resources are available and a highly competitive model is needed, placing effort into optimizing the neural network could be a fruitful approach.

## Extensions

As mentioned, there are certain aspects that this paper only partly touches upon or discards entirely. These are things that could be investigated if one were to try to create a model that models bond liquidity as well as possible, or in other words, things that could lead to an improvement in the accuracy of the model. The most obvious thing to try is to change the model. The idea behind the self-normalizing neural network is that it should self-normalize, i.e. keep all weights in the network to a small value in order to act as a form of regularization. By looking at the results from the SNN it is clear that the models overfitted to the training data. Thus in further research one could try to add other types of regularization in order to bring down the level of overfitting. This could, for example, be to add dropout to the training or adding a penalty to the weights.

Another likely reason as to why the model overfitted is the relatively small dataset used to train the model. Since all of the Swedish bond market was used in the training, there are no more *real* data points to be added. One way to handle this could be to add data points that are not exactly from the same dataset but ones that we expect to have similar behavior, for example one could use the entire Scandinavian/Nordic bond market in the training process. Another approach to extending the training data is to use *data augmentation*/bootstrapping. As an example, one could randomly copy $N$ data points from the training set, add a small random noise to the continuous features and then add this newly created data point to the training data in order to synthetically expand the training data.

Furthermore, to increase the accuracy one could try to actually optimize the structure in the neural network in terms of the accuracy on the validation set. In this paper the number of hidden layers, and the number of nodes in each hidden layer, was set to be the same for all models. In reality it is most likely that these hyperparameters have an impact on the performance of the models; what works well for one model might not work well for some other model and vice versa. Another aspect that most probably influences the performance of all models is the way one creates the different classes. In this paper the data set was ordered with respect to the value of the liquidity measure and then partitioned into subsets of equal cardinality. It is not obvious that this is the optimal approach, in fact by looking at Figures (10) - (13) from *Appendix A*, one would expect this partition to create "unnatural" classes. This unreasonable expectation that the ordered measures would grow linearly, might lead to a situation where data points that are very close in measure may end up in different classes. One way to improve this partitioning would be to instead look at the range of the measure and divide this into an equally sized partition, and place the data points whose measures' fall within the same subset belong to the same class. Alternatively, one could use a clustering algorithm on the measures in order to form more "natural" classes.

Finally, an interesting extension to this project would be to, instead of using a classifier, use a regression approach. This, can of course also be done using a neural network. An issue is how one would determine the accuracy of the model, but the overall performance could be analyzed using the "simulation" used in this project (which in fact would be better suited for the regression case). In fact this paper has served as a kind of pre-study of whether neural networks can be applied to predicting liquidity and if one were to find a way to eliminate the overfit, the natural next step would be to use the same methods for the regression case, which would be better suited for use in industry.

## References

[1] European Banking Authority, *Report on appropriate uniform definitions of extremely high quality liquid assets (extremely HQLA) and high quality liquid assets (HQLA) and on operational requirements for liquid assets under Article 509(3) and (5) CRR*, 2013.

[2] Somer, P., Pasquali, S., *Liquidity - How to capture a multidimensional beast*, Journal of Trading, Volume 11, Number 2, 2016.

[3] Demsetz, H., *The Cost of Transaction*, The Quarterly Journal of Economics, Massachusetts, Volume 82, issue 1, 1968.

[4] Black, F., *Fundamentals of Liquidity* , Journal of Financial Economics, 1970

[5] Grossman, S.J., Miller, M.H., *Liquidity and Market Structure*, Journal of Finance, Volume 43, Number 3, 1988

[6] Harris, L.E, *Liquidity, Trading Rules, and Electronic Trading Systems*, New York University Salomon Center Monograph Series in Finance, New York, 1990

[7] Mehrling, P., *The New Lombard Street: How the Fed Became the Dealer of Last Resort*, Princeton University Press, Princeton, NJ, 2010

[8] European Banking Authority, *Discussion Paper: On Defining Liquid Assets in the LCR under the draft CRR*, 2013.

[9] Basel Comittee on Banking Supervision, *Basel III: The Liquidity Coverage Ratio and liquidity risk monitoring tools*, Bank For International Settlements, 2013.

[10] Friewald, N., Jankowitch, R., Subramanyam, M.G., *Illiquidity or Credit Deterioration: A study in the US corporate bond market during financial crises*, Journal of Financial Economcis, Volume 105, 2012.

[11] Hult, H., Lindskog, F., Hammarlind, O. Rehn, C.J., *Risk and Portfolio Anlysis: Principles and Methods*, Springer, 2012.

[12] LeCun, Y., *A Theoretical Framework for Back-Propagation*, Proceedings of the 1988 Connectionist Summer School, Pittsburgh, PA, 1988.

[13] Kingma, D., P., Ba, J., *Adam: A Method for Stochastic Optimization*, Conference Paper at the 3rd International Conference for Learning Representations, 2015.

[14] Klambauer, G., Unterthiner, T., Mayr, A. Hochreiter, S., *Self-Normalizing Neural Networks*, Advances in Neural Information Processing Systems (NIPS), 2017.

## Appendix A

Figures (10) - (13) show histograms over the different liquidity measures.

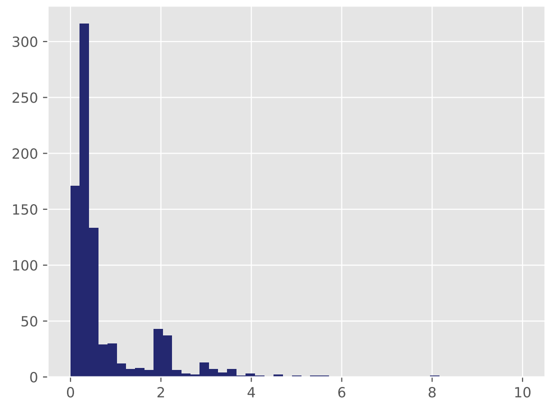


Figure 10: Histogram over the average spread for all data points.

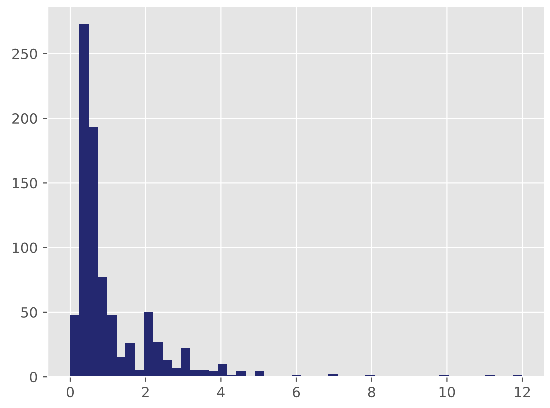


Figure 11: Histogram over the average spread $\mathrm{VaR}_{0.95}$ for all data points.

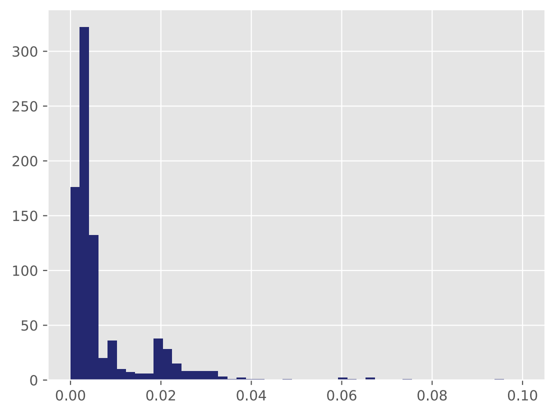


Figure 12: Histogram over the mid spread for all data points.

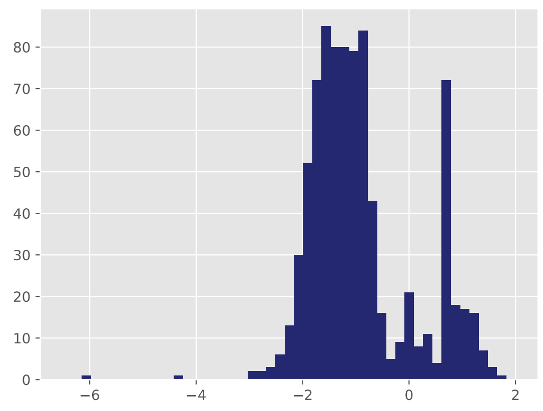


Figure 13: Histogram over the log spread for all data points.

TRITA -SCI-GRU 2018:231