# Intelligent Machines in Economics

- A macroeconomic application of the LSTM neural network

Master Thesis

Rasmus Schier
- 20136441 -

Aalborg University
MSc Economics

AALBORG UNIVERSITY
DENMARK

**Economics**
Aalborg University
http://www.aau.dk

**Title:**
Intelligent Machines in Economics - A macroeconomic application of the LSTM neural network

**Author:**
Rasmus Schier

May 30, 2018

**Supervisors:**
Roman Jurowetzki
Hamid Raza

**Page Numbers:** 66

**Acknowledgement**

Thanks to my supervisors Roman Jurowetzki and Hamid Raza for helping and guiding me through this thesis.

## 0.1 Abstract

Macroeconomic data often consist of datasets with a large amount of variables containing only relatively few observations. This is disadvantageous when dealing with statistical consistency and degrees of freedom in complex models requiring lots of explanatory variables.
The arrival of so-called "big data" has created a surge of new and different types of data that makes more information available to those who can extract it. With econometric models often being asymptotically estimated, the marginal information gain does, however, decrease when the sample size grows to a certain size. The decreasing econometric efficiency for large datasets makes more powerful extraction methods interesting. This has raised the popularity and general interest for machine learning methods, as these are able to extract information somewhat automatically and efficiently. The Long Short-Term Memory neural network is one of these models. It imitates the biological brain function of synaptic activity in an artificial neural network. With this it is able to form patterns based on input data. This type of network solves problems of earlier types, by including the ability to retain information, but also forgetting it when it becomes irrelevant. The patterns learned are used to produce predictions based on inputs. Common usages of this network is for classification purposes of different inputs, often to extract information. By classifying inputs the network can essentially be used as data generating machines.

Economic applications of these learning networks are scarce and often limited to univariate financial time series.
The LSTM in this thesis is fed 19 different danish macro indicators in an attempt to predict the continuous value of growth in danish GDP. The indicators are common to GDP predictions and selected manually based on economic literature of GDP "nowcasting". For benchmarking purposes a principal component regression factor model with five latent factors is constructed. The validation methodology follows the supervised machine learning approach of splitting the dataset into a train- and test-set. The models are trained and estimated on the train-split, while the test-split is used for out-of-sample prediction accuracy. The LSTM show better performance in the setup presented, with a lower squared error, RMSE and MAE. The Diebold-Mariano forecast test does however suggest that none of the models predicts significantly better than the other.
The results suggest that the LSTM may improve accuracy with more available data, and that the network can be used in a macroeconomic setup. It does however have limited structural inference capabilities, which hurts the trustworthiness of the predictions and limits the networks forecasting capabilities. The networks provide no real benefit over the principal component regression presented. This places the

LSTM as a powerful predictive tool, but still a tool that is best used right now as a powerful data generating method, allowing econometrics to handle the economic interpretation.
Exciting possibilities does however exists for the networks in economic applications. Better predictive accuracy may be possible with the variational auto-encoder, where a LSTM encodes information into a condensed representation of the inputs received. The goal is to reconstruct the inputs as closely as possible from the condensed information. Much like the latent factors in the factor models. The condensed information can be used for prediction, but structural inference is still not possible.
The LIME framework allows for transparency in the networks by providing local interpretations of inputs. In theory opening for the possibility of structural inference. The framework is mainly developed and tested in classification applications, so further research is necessary to conclude the usefulness in the macroeconomic regression setup.

# Contents

# Chapter 1

# Intelligent Learning and Economic Predictions

While the idea of "Artificial Intelligence" has been around since it was first discussed in 1943 (McCulloch & Pitts, 1943), it has recently experienced a surge of popularity. With a significant increase in the amount and quality of all kinds of data, the buzzword "big data" has emerged to describe a new plethora of information available. This new kind of data consists not only of numbers, but all sorts of information like tracked behaviour, images and text. It has allowed computer scientists to imitate seemingly "intelligent" learning in machines, with the use of powerful neural networks, squeezing even more information out of available data. The Long Short-Term Memory (LSTM) is such a network, for example driving the "brain" behind self driving cars and face- and voice recognition. While older feed-forward neural networks can learn underlying patterns in data, the LSTM introduces memory to the networks allowing it to selectively remember earlier useful patterns.
In economics different attitudes exists towards the use of data. Many nuances and increments exists, but in general there are those who use it descriptively to support theory and those who "let the data speak". For those actively relying on data and model estimation, this advancement in data and technology provides interesting opportunities in an econometric context. The usefulness and relevance is however heavily reliant on the actual goal of the application. Two main directions arise when discussing machine learning and econometrics; econometrics versus machine learning, and econometrics combined with machine learning.

### 1.0.1 Econometrics combined with Machine Learning

The approach of combining machine learning and econometrics is essentially a two step process: 1) Machine learning is used to extract as much relevant data as pos-

sible, which can then 2) be fed to any desired model.
The general disregard for the type of data in machine learning has some clear advatages in extracting and clustering data. The network in this thesis is presented sequences of data in which it has to find patterns. The sequences does not necessarily have to be of economic nature. A typical application of this type of network is in image recognition, where the network categorise a sequence of pixels based on predefined labels. (Henderson et al., 2012) used satellite images in an econometric framework, where changes in emitted light captured by satellites is used to reflect economic activity and growth in GDP. This is highly relevant for low-income countries with poor data. The intensity of light is measured from the images in seven levels of intensity and is combined with density of income holding population; low density areas will have a lot more unlit pixels than high density areas. The premise is therefore that high density areas with lower intensity, experience less growth than high density areas with high intensity. The framework presented requires lots of manual data processing to fit the data. Implementation of an automatic method of generating economically categorized image data, would be highly useful in this case.
Another use of satellite images is presented in (Lobell, 2013), where satellite images of crops are used to explain and predict crop yield and deviation. The article is again suffering from manual extraction of useful information, in which the automatic extraction would be useful.
Another way to implement the automatic data generation is in language processing, both spoken and text. By automatically labelling the attitude of headlines and comments on different subjects, sentiment analysis can be incorporated into prediction of stock movement. (Antweiler & Frank, 2004) applies this idea in a regression framework to model volatility based on attitude. The difficulties of this is, again, labelling data. With the LSTM's ability to capture long term dependencies by for example connecting words appearing different places in a sentence, the attitude of the message can be extracted. By presenting the network with different sequences of words, the attitude can be categorized as i.e. "positive", "negative" or "neutral". This method can be extended to classify not only messages directly addressing specific subjects, but also volatility appearing after messages from influential agents. All these examples are classification problems, which is then applied to a continuous problem.

Automatic generation and extraction of relevant information is useful in a data driven economic context. But when applying the machine learning algorithms as data generating methods, the discussion of implementation quickly becomes about the capabilities of the algorithms. The economic interpretation is added by the surrounding econometric problem and structure. Exploring machine learning capabilities in a framework like this changes the nature of the discussion to a computer

science problem of classifying data most efficiently. The economic value added by extracting even more information, is simply the increase in available data.

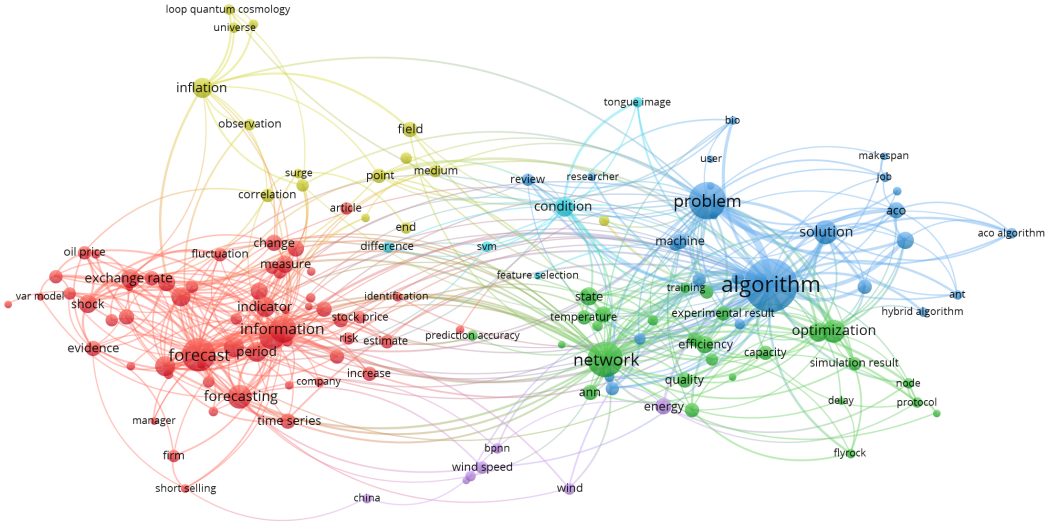
### 1.0.2 Econometrics versus Machine Learning

The other main implementation is to test whether machine learning methods can outperform econometric methods in prediction. An application like this utilises machine learning methods to solve the econometric problem, instead of solving what is essentially data problems happening to be of an economic character. In this way, the discussion is kept in the economic domain. This approach is discussed and applied in this thesis.
Explicitly trying to solve econometric problems with "intelligent learning machines" in the form of neural networks, and the LSTM in particular, is a relatively new endeavour. (Herbrich et al., 1999) discussed economic applications of neural networks in 1999 and the subject evolved up through the 2000's, where it was also discussed in prominent econometric textbooks (Enders, 2015) as an alternative non-linear method. It did, however, never gain much traction because of the limited amount of relevant economic data. No significant improvements where made over already existing econometric methods on available data. Partly because the search algorithm behind the networks require lots of observations, which limits applications to financial series containing many observations.
(Enders, 2015) discusses a standard neural networks in a univariate financial application in his textbook. (Zhang et al., 2010) used a feed-forward network in a multivariate prediction of the possibility of recession in China, based on leading indicators. (Varian, 2014) talked about different machine learning approaches from tree structures to LASSO-regressions. (Fischer & Krauss, 2017) applied the LSTM to multivariate financial time series in an exploration of its performance compared to many of these methods mentioned by (Varian, 2014) in an economic context. The network showed better performance in predicting time series movement compared to both the popular Random Forest method, LASSO-regression and simple feed-forward neural networks.
The constructed bibliographic network of clustered keywords in Figure 1.1, shows a separation of machine learning and economics in the literature. It also suggest a lack of literature connecting macroeconomics to neural networks. The map is created by linking keywords in multiple layers of references[1], starting from (Kuang et al., 2017). This article discuss the use of a modified neural network to forecast different financial series, but also inflation rate. In this sense the article provides a good starting point for exploring the links in the literature revolving around the problems and methods of using neural networks to solve econometric problems. A link between two keywords suggests use of both keywords in the articles. The

[1]Created with Web Of Science v.5.29

size of clusters grows with the number of occurrences. Distance between clusters is a signal of which type of article the keywords are used in.
The problems and methods in the left cluster have an economic character, while the right is concerned with machine learning methods and problems. The red cluster is focused around forecasting, while the yellow moves into keywords related to the details of econometric methodology. The green cluster revolves around neural networks, while the blue is keywords used in discussion of machine learning methodology. The relevant keywords are those specific to the field it is used in, for example *"network"* and *"forecast"*. Keywords such as *"problem"* or *"information"* is less relevant as they are too general, only connected to a cluster because of a higher number of mentions.

**Figure 1.1:** Bibliographic Network Clustered by Keywords and Linked by Common References

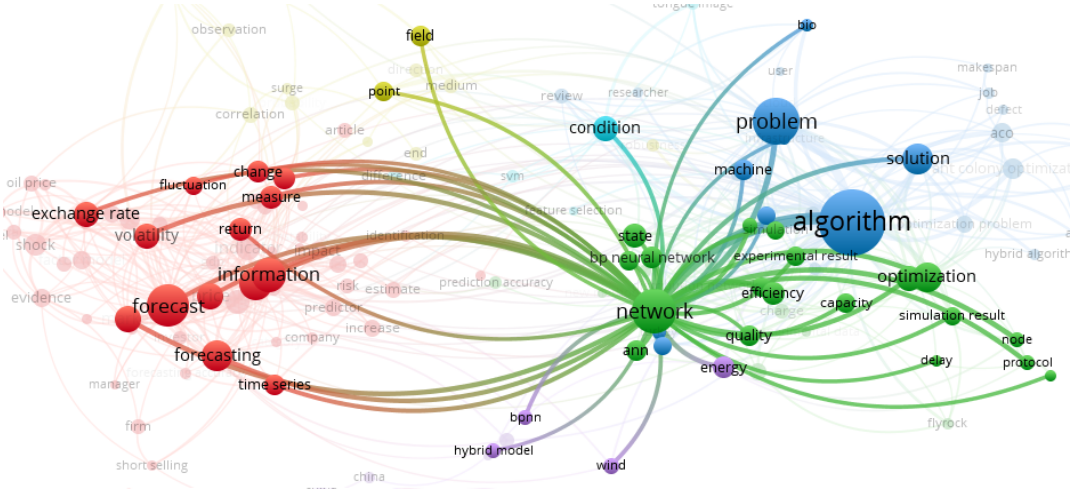



*Source: Web of Science v.5.29*

Looking at the links of the *"Network"* keyword in Figure 1.2 the connection to economic literature is focused on financial forecasting of exchange rates, volatilty and return. Following the keyword *"Indicator"* in Figure 1.3 it does have some ties to *"algorithm"*, but primarily connects to other economic keywords. The *"GDP"* and *"recession"* keywords connects only to economic keywords.
So while the neural networks have been applied to financial time series, this overview suggests that macroeconomic applications are limited.

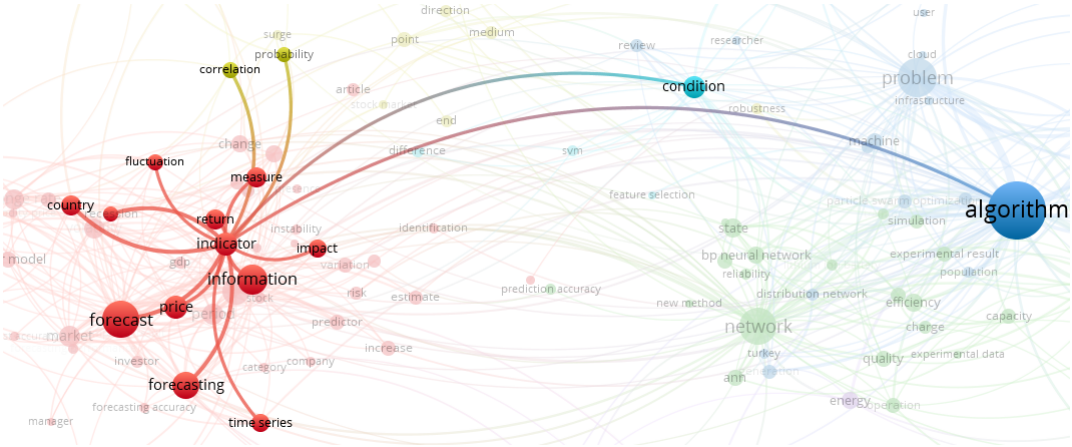
*This thesis therefore contributes with an application of the more powerful LSTM in a traditional multivariate macroeconomic perspective. The goal is to explore how well the LSTM predicts growth in GDP, based on macroeconomic indicators for the danish economy.*

**Figure 1.2:** Links for "Network" Keyword




*Source: Web of Science v.5.29*

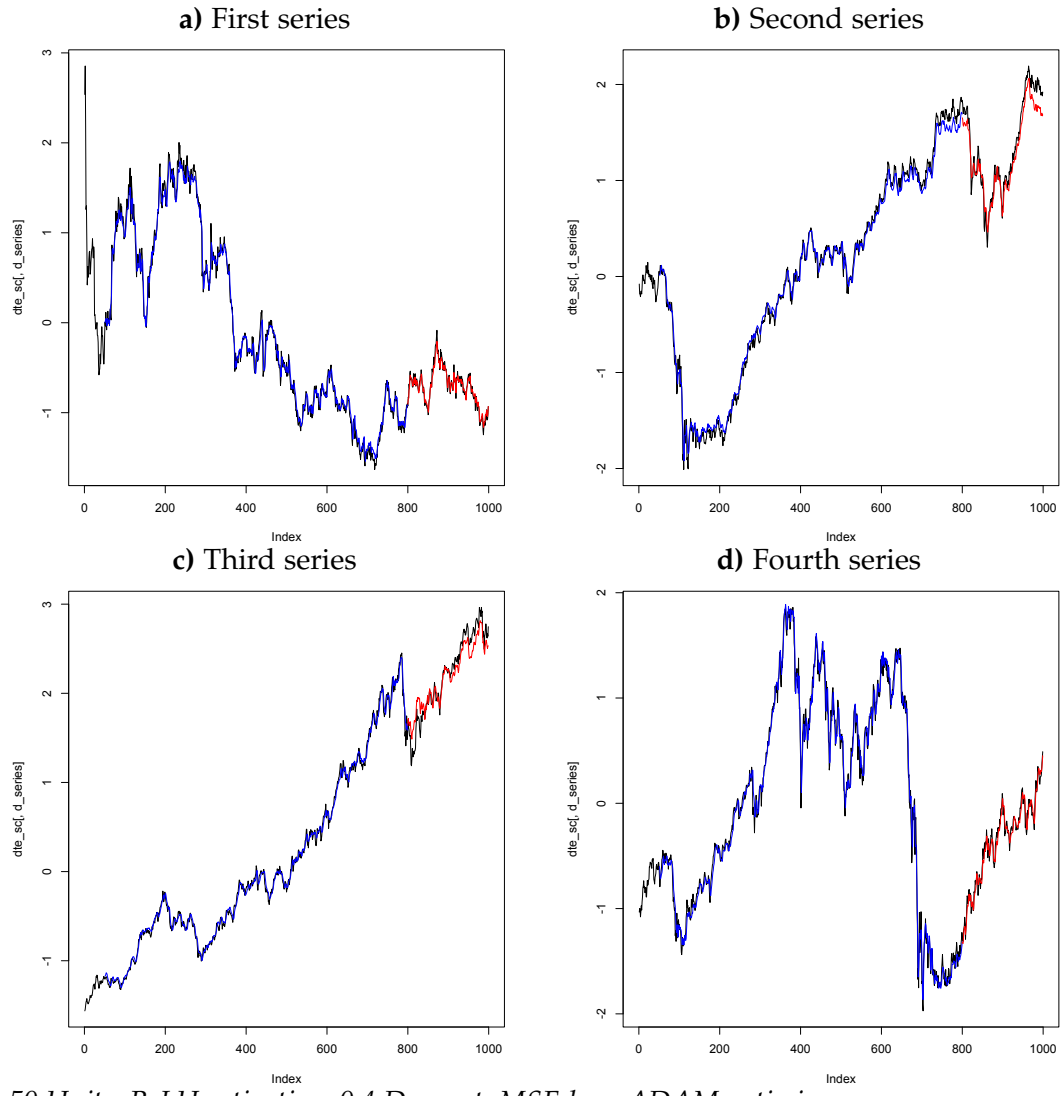
**Figure 1.3:** Links for "Indicator" Keyword




*Source: Web of Science v. 5.29*

A showcase of the pattern learning ability of the LSTM is presented in Figure 1.4, where it predicts four univariate time series. The series are unnamed daily series, randomly chosen from Spyros Makridakis M4 Forecasting Competition (Makridaki, n.d.). The networks are constructed in a sliding window fashion, creating sequences of 50 observations. The first 50 observations are used to predict observation 51 and the window is then moved one-step ahead, using observation $[2 : 51]$ to predict observation 52, and so on. The tuned parameters are shown under the figures, but explanation of these are saved for later. The data is split

**Figure 1.4:** LSTM of M4-Competition Time Series




*50 Units, ReLU activation, 0.4 Dropout, MSE loss, ADAM optimizer*

into a training- (blue) and test-set (red) for the series. The network is only fed the

training set, and then has to predict the out-of-sample red part, based on the patterns learned from the training. As it can be seen, the network shows good fitment in both training and prediction. No specific tuning of parameters is done for the individual series. This is why the prediction of some series are better than others, as the network configuration works better for some series. This seemingly strong ability to extract patterns, is what this thesis applies to a multivariate macroeconomic setup.

### 1.0.3 The Macroeconomic Problem

*"Taking the temperature"* of an economy is highly useful in decision making. It can, however, also be a complex task to perform. Growth in GDP reflects the cyclical movements in an economy, but the information is ultimately retrospective. This is a disadvantage to those using this information, so prediction of GDP growth is useful.
A simple univariate autoregressive forecast with the ARIMA model could be used, but in the case of GDP, a multivariate approach has its benefits. Mainly because it opens for the possibility of modelling the complex system that an economy is and basing the prediction on leading indicators for the economy. Vector autoregressive models can create such structural equations, but allows only for a limited number of indicators to be included. They are therefore useful for impulse response analysis between variables, but not necessarily predicting GDP from a large vector of indicators. Furthermore macroeconomic data is often "wider" than it is "long", meaning that a lot of different variables exists but each containing relatively few observations. The econometric solution are Dynamic Factor Models, which solves some of these problems, as they allow for many indicators to be included in the prediction. A principal component dynamic factor model is used as benchmark for the LSTM.
The method of applying an LSTM to macroeconomic data with many indicators is somewhat straight forward, as the structure and method of the network simply maps multiple inputs to an output. In theory it should be as simple as plugging in a number of indicators and a desired output, such as key economic indicators and GDP. In a sense this thesis therefore also explores if many series with relatively few observations, can be used in place of long time series, when using the LSTM.

The thesis is divided into six chapters:

- The first chapter constitutes this introduction.
- The second chapter places machine learning in the "Artificial Intelligence" terminology, presents the biological motivation for constructing artificial neural networks in the first place and discusses the methodology behind training

and validation in machine learning. The chapter further introduces neural networks in detail ending in the LSTM. An economic context is provided where possible.

- The third chapter presents a quick run down of the theory behind the Principal Component Dynamic Factor Model used for benchmarking and why it can handle high-dimension dataset with many macroeconomic indicators.
- The fourth chapter compares the results of GDP growth prediction, from both models.
- The fifth chapter discusses some exciting possibilities for the LSTM in an economic context.
- The sixth chapter provides a conclusion.

# Chapter 2

# Machine Learning Approach

For the sake of clarity this chapter starts out with an overview of the key elements associated with the term *"Artificial Intelligence"*. This is mainly done because this thesis only relates to the discipline of machine learning and ultimately the application of the LSTM in an economic context. This is necessary because *machine learning* and *artificial intelligence* often are used interchangeably. A biological motivation for using artificial neural networks is presented, and the field of machine learning is visualized.
Next the evolution of artificial neural networks is presented in detail in a modular framework ending in a presentation of the Long-Short-Term Memory (LSTM) network. The aim is to introduce neural networks to those unfamiliar with machine learning. This covers the mathematical method behind the networks, the arising problems of the method used and a presentation of the LSTM, that solves some of the complications earlier network algorithms suffered from.

## 2.1 Overview of Artificial Intelligence

As stated in the introduction the term artificial intelligence is far from new. The idea of modelling artificial computational learning started in 1943 (McCulloch & Pitts, 1943), but the term *"Artificial Intelligence"* is said to have emerged from a Darthmouth workshop in 1956, participated by several prominent researchers. It was concluded that true artificial intelligence only exists when *"A self-improving machine (or program) is created. When computers are able to learn by reading existing natural language text and when an AI has problem solving capacity equivalent to the computer science community"* (Hall, 2011, p. 174). In other words a machine that mimics the abilities of the human brain.
Ray J. Solomonoff, one of the attendees at the Darthmouth Workshop, later discussed flaws in developed "learning machines" and associated computational al-

gorithms, which excludes them from reaching human like learning- and problem solving capabilities. These algorithms, even though highly useful tools and essential to machine computation, still needs a sourrounding cognitive architecture. This architecture has to call upon instances of machine learning algorithms, in order to be classified as true "artifical intelligence" (Solomonoff, 2009) (Hall, 2011). The overview can therefore be presented as Figure 2.1
As it can be seen in Section 1, a functioning AI is required to mimic the collaboration of different cognitive functions. It has to be able to process natural language in text and speech, display reasoning, problem solving and planning, and be able to represent knowledge. On a physical level it also has to mimic motoric functions and be able to display perception, often through sensors (Desai, 2017). The term artificial intelligence can be seen as a somewhat more theoretic and philosophical concept, while machine learning is the practical computer science instantiation of the concept, handling the different tasks. This is why the focus of this section of the thesis is directed towards the application of artificial intelligence, being machine learning, rather than artificial intelligence as a whole. The LSTM network used in this thesis is placed in the recurrent network category trained in a supervised fashion where both input and output is presented to the network.

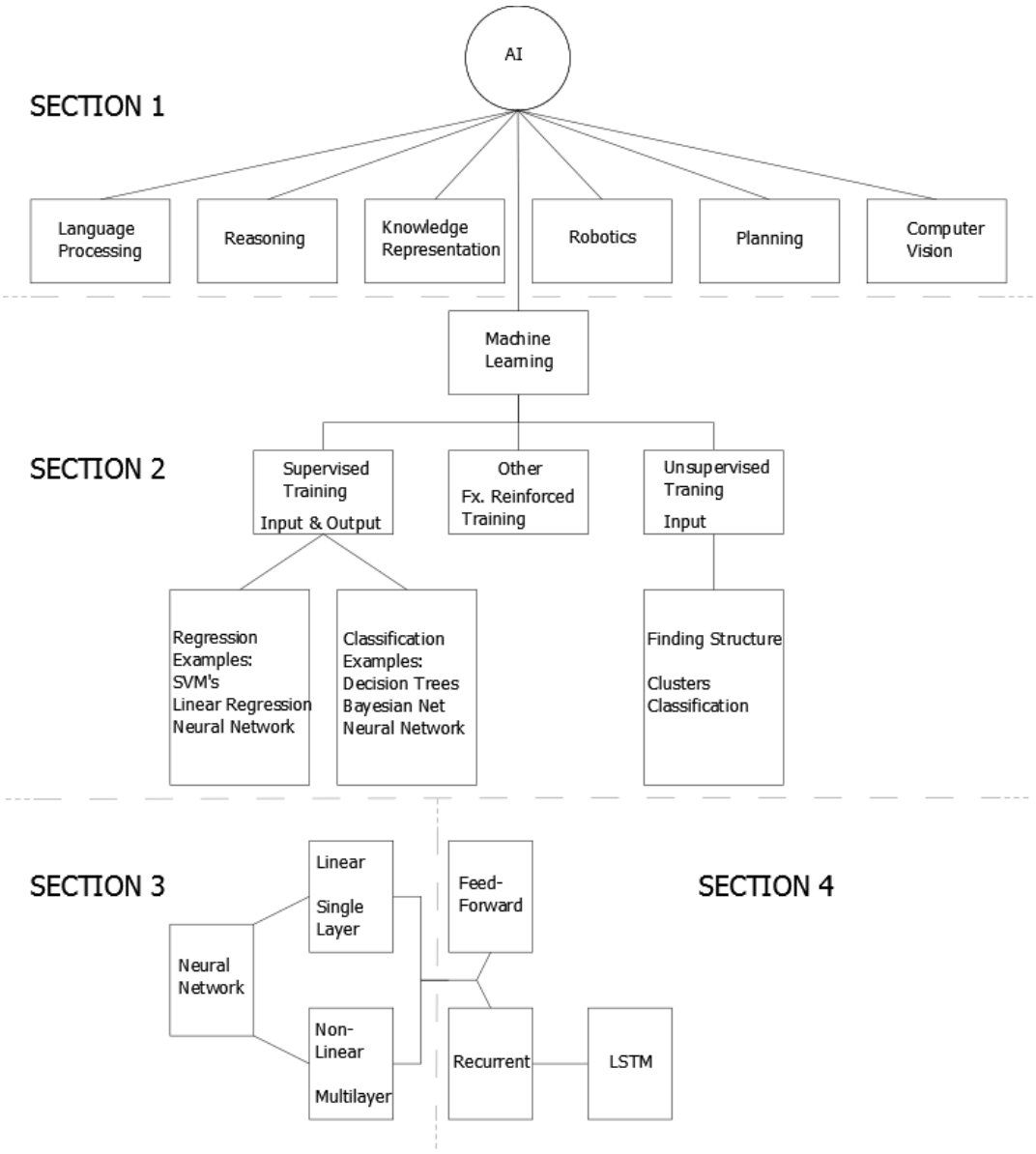
## 2.2 Machine Learning Methodology

Tom M. Mitchell summarised the field of machine learning as *"the attempt to construct computer programs that automatically improve with experience."* (Mitchell, 1997, p. xv) and defined computer learning broadly with:

*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*
- (Mitchell, 1997, p. 2)

In short: a computer learns by experience if it improves on its performance doing a task. A broad definition like this captures the interdisciplinary nature of machine learning, and shows its relevancy in economics. Machines that learn underlying patterns and improve these with experience, essentially solves optimization problems.
As Section 2 show in Figure 2.1 the plethora of different training algorithms can in general be split into different classes. The relevant in this case being the supervised training (Ayodele, 2010b).
During supervised training, the algorithm is presented with pairs of inputs and outputs from a training dataset. The goal is for the learner to learn an underlying function by mapping the input to output. Essentially learning that a special

**Figure 2.1:** Overview of Artificial Intelligence




*Source: Section 1 (Desai, 2017), Section 2 (Ayodele, 2010a), Section 3 (Mitchell, 1997), Section 4 (Elman, 1990)*

configuration of input values results in a given output. For example macroeconomic indicators to growth in GDP. With this underlying function it should be able to correctly predict an output value from any input. A continuous problem like predicting the growth in GDP would result in a continuous value, while a classification problem like predicting the state of a binary recession indicator would result in an assigned state.
Supervised training requires some form of data-preprocessing to work properly, often including some sort of feature/attribute subset selection and feature/attribute construction or transformation to reduce dimensionality of data, while still keeping complexity (Ayodele, 2010b). This relates much to the selection of key indicators in an econometric approach. Private consumption would in economic theory for example be a feature of GDP, as it is assumed to explain GDP. In this sense the selection of economic inputs to the model, relies as much on these assumptions, as econometric approaches.
With proper data, the learning algorithm recurrently adjusts its parameters to minimize the error between desired output and predicted output. The specific training algorithm is discussed in section 4.2.
The validation of machine learning models under supervised training, is based on a train- test split of the data. The set is split into two: one training set and one test set for out-of-sample validation of the predictions. It is further split into an input subset containing all input series, and output subset. In this case the inputs are the key indicators while the output data is the GDP. The result is four different datasets: an input and output for training and an input and output for out-of-sample validation. The argument behind using out-of-sample validation is that the algorithm is optimized until the network produces the lowest error possible for the prediction. Not necessarily the lowest error of fitment to training data.

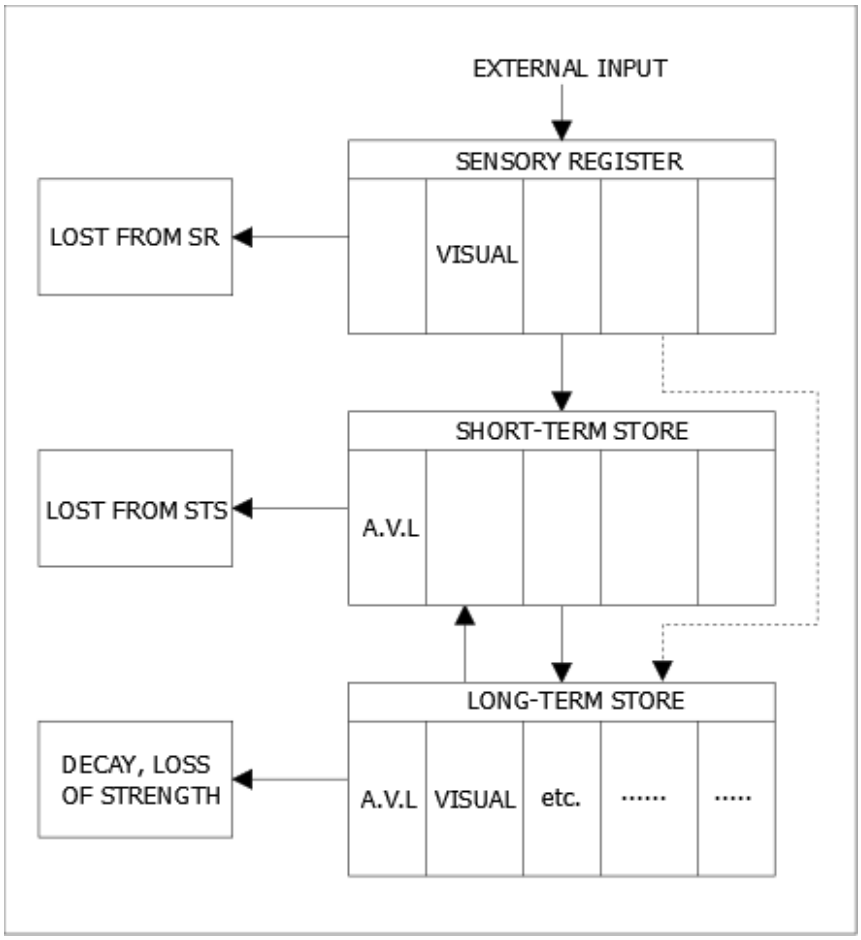
Given the structure of the algorithm, and as it will become evident in the theoretic explanation in section 2.4.1, the neural network have one main focus: prediction. This sets the methodology and philosophy behind machine learning apart from that of econometrics. While econometrics often revolves around parameter estimation, the machine learning approach is build around prediction of output (Mullainathan & Spiess, 2017). In essence focusing on prediction of $\hat{y}$ rather than on estimation of $\hat{\beta}$. This sets a limit on the relevant cases for which to apply neural networks, as structural inference and response analysis has no direct implementation in the networks. This could be one reason for the relatively few number of macroeconomic applications. With the missing ability for structural inference, relevant applications are pushed towards those who do not require long run estimation. Practitioners are thus not necessarily concerned with considerations such as cointegration and error correction.
The network is however still a highly useful tool in the right applications because

of its high level of automation in learning and recognising patterns from data. Econometric practitioners are aided by significance levels in the distribution of the data, but recognizing patterns such as relevant lags or regressors are still ultimately the choice of the practitioner. The LSTM learns these patterns on its own by pure error minimization, ultimately removing the manual aspect of extracting relevant information from the data by automating the process. In this sense the "intelligent learning" is incorporated into the economic modelling.
The inner workings of the network and the automated learning is explained in the next sections, starting with the biological inspiration for forming artificial neural networks in the first place.

## 2.3 The Biological Motivation For Artificial Neural Networks

Pattern recognition is vital to the function of humans and animals. From a practical point of view of recognizing poisonous from edible and friends from foes, but also from a functionality perspective of, for example, categorizing sound waves into understandable language. Cognitive neuroscience describes this process as the brain matching information from stimulus with information from memory (Eysenck & Keane, 2005). Building on this, memory and in extension learning, is crucial to the overall progression of the individual.
The formation of memory is in general credited to synaptic activity in the brain. The brain consists of billions of neurons which each connect to other neurons through neurotransmitters and receptors forming a synapse. The transfer of electrical charges through synapses is what makes up synaptic activity (Bliss & Collingridge, 1993). In 1968 Atkinson and Shiffrin proposed a theoretic framework that categorized this activity into three stages: sensory register, short-term store and long-term store (Atkinson & Shiffrin, 1968). The framework is somewhat limited to the scope of stimulus to the visual system, as it was well understood at the time while understanding of other sensory dimensions where limited. Figure 2.2 shows the different stages of synaptic activity, starting with an instant initial registration of visual external input and a *"serveral hundreds miliseconds decay"* before it is forgotten. Before the sensory registration has decayed the input is transferred to the short-term stage, which has a longer associated decay. This stage is also referred to as the *"Working Memory"*, and acts as a filtering mechanism transferring only some of the sensory input into long-term storage. Atkinson & Shiffrin argues based on other studies, that the transfer from short-term storage to long-term storage may in some cases be associated with conscious effort to remember and others with involuntary repeated exposure to the input. The efficiency of recognizing these patterns of synapse activity and decay, makes the basis for forming memory in the brain. This efficiency may however be hard to estimate. The framework revolves around the authors defined *"auditory-verbal-linguistic"* (shortened a-v-l)

**Figure 2.2:** Atkinson & Shiffrins Memory Model




*Source: Recreation of Figure 1 - Structure of the memory system (Atkinson & Shiffrin, 1968)*

storing mechanism, because one mode of sensory input may be stored in short-term as another mode. Using the example from the framework, a written word is initially visually registered, but may be stored in auditory short-term storage because of phonetical association. This modal difference makes the estimation of efficiency hard.
The specific neurological events making up the transfer from short-term storage into long-term storage is later on discussed and improved by Beddeley & Hitch in 1974 (Baddeley & Hitch, 1974). The conceptual simplicity of the framework presented by Atkinson & Shiffrin is however relevant when applying the transfer of input through a neural network.
The study of biological learning systems has greatly inspired computer scientists to model artificial neural networks in order to both aid biological research, but also recreate artificial systems making machines capable of learning.
To warrant the exploration of artificial learning systems with machines, one textbook on machine learning presents the claim that raw computational power of computers is superior to the power of the human brain (Mitchell, 1997). The fastest on/off switching time of neurons in the brain is estimated at around $10^{-3}$ seconds, while the switching time of transistors in computer processors is estimated to be around $10^{-10}$ seconds (Mitchell, 1997, p. 82). As Mitchell however also states, serveral problems arise from this claim. The idea of adopting a similar structure is valid, but as Atkinson & Shriffin stated, the efficiency of the transfer function from sensory input to working memory and long-term storage is unknown because of modal switching between the stages. This is exactly why the a.v.l composite is created, because it is unknown how much weight is placed on each sensory input. An artificial neuron outputs a single value while the synapses in the brain exchange a set of information. This invokes an epistemological problem in the assumption that neuron activity in the brain is comparable to the simple on/off switching of transistors in computer processors (Chalmers, 1993).
Because of this the field of machine learning can be split in two paths

- One that attempts to model and mimic the neural activity in the human brain as accurately as possible.
- One that focuses on the computational implementation by taking advantage of processor power, while sacrificing realism.

In the context of this thesis the latter path is chosen as focus is directed towards the algorithmic computation with neural network and not necessarily at true to life modelling. This is also the reason for using the structure of memory formation proposed by Atkinson & Shiffrin, as reference when making biological analogies.

## 2.4 Artificial Neural Networks

Stephen Grossberg (Grossberg, 1980) and Rumelhart & McClelland (Rumelhart et al., 1986) are major contributors to the computational modelling of memory formation in artifical neural networks, building on the biological motivation. Rumelhart and McClelland presented in 1986 a framework for parallel distributed processing (PDP) taking the form of an artificial neural network. Grossberg greatly expanded on the non-linear capabilities of the PDP networks, by implementing non-linear activation functions.
The authors identify eight components of a PDP model:

- A set of processing units
- A state of activation
- An output function for each unit
- A pattern of connectivity among units
- A propagation rule for propagating patterns of activity through the network
- An activation function for combining inputs impinging on a unit with a current state, to produce a new level of activation
- A learning rule whereby patterns are modified by experience
- An environment within which the system must operate

- (Rumelhart et al., 1986)
This section of the thesis presents this framework.
Secondly the general components of learning in a network is presented in-depth, deriving the *delta rule* proposed by Rumelhart & McClelland, and the application of said rule in the *back-propagation* algorithm. Different types of neural networks exists. The most relevant in the context of this thesis being the feed-forward network and the recurrent network.

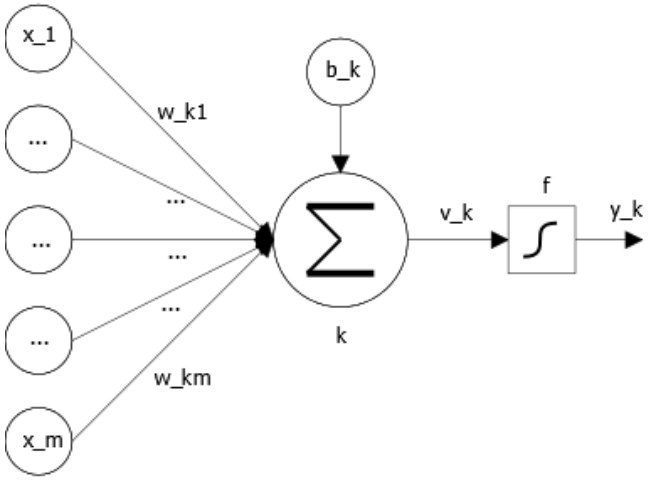
By quickly reflecting on this framework the economic relevancy becomes clear. The network takes a number of economic series as input and sends them through processing units. These units are activated according to a specified rule and transformed into a new output which is then send through other connected units. In the end of the network an output is produced; the growth in GDP. Using the out-of-sample methodology this is then compared to the real value of GDP growth, and the connections between the neurons are then modified to produce an output closer to the target GDP growth. The learning rule specifies how the algorithm does this on its own, requiring minimal interference from the designer when the data is provided.

### 2.4.1 A Modular Network Architecture

#### The Feed-Forward Network

The most basic component of a neural network is the neuron (Rumelhart et al., 1986).
Referring to Figure 2.3, a neuron, $k$ takes a number of inputs $x_m$. The input is

**Figure 2.3:** Representation Of A General Neuron




*Source: Figure 2.1 Single-Layer Neuron*

multiplied by a weight $w_{km}$, connecting the input to a summation function. A bias is added here, $(b_k)$. This gives an input $v_k$ which is passed to an *activation function* $f$. This produces the neuron output $y_k$ send to the next neuron. The output of a neuron is thereby:

$$y_k = f_k(w_{km}x_m + b_k)$$

$m$ refers to a specific input, $k$ to the specific neuron in a layer of more than one neuron. $v_k$ is a local field that will later on be filled with a gradient.
The parameters $w_{km}$ and $b_k$ can be adjusted by the network according to a learning rule, while the activation function is chosen by the designer at initial setup (Hagan et al., 2014).
Following the biological analogy, the weight $w_{km}$ is the strength of the signal coming from a synapse, the summation and activation function is the body of the neuron, while the output is the signal send to a synapse connected to another neuron.
This is also how a network can have multiple layers; an *input* layer, a *hidden* layer

and an *output* layer. With multiple inputs $x_m$ and multiple neurons, $k$, multiple weights exists. These form the weight matrix, $W$, with elements of $w_{1,1}, w_{1,2}, w_{2,1}, w_{2,2}..., w_{k,m}$.

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \dots & w_{k,m} \end{bmatrix}$$

This weight matrix is multiplied on the inputs. The matrix also resembles a simple single-layer network with multiple inputs. The output from each neuron is therefore given as:

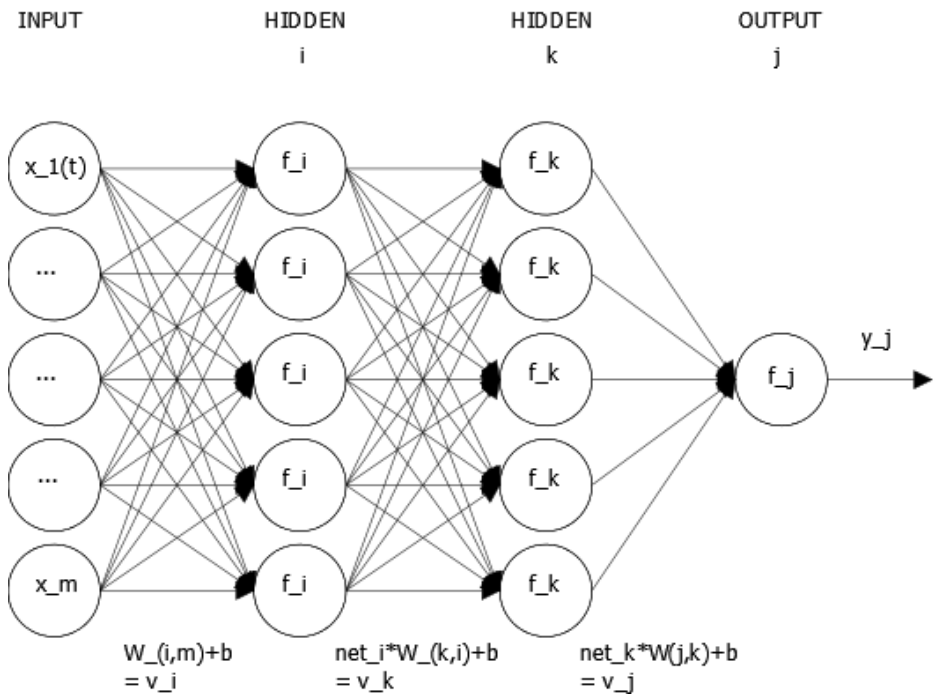
$$y_k = f_k(Wx_m + b_k) \tag{2.1}$$

A multi-layer network includes an input layer, a number of hidden layers and an output layer. With multiple hidden layers and an output layer. $k$ denotes neurons in the layer to the right of neurons $i$, and $j$ to the right of layer $k$, giving the order:

$$i \rightarrow k \rightarrow j$$

This is shown in figure 2.4 The flow of information is the same as in the simple

**Figure 2.4:** Multi-layer Network




*Source: Figure 2.10 Multi-Layer Network*

neuron. The input $x_m$ is multiplied by weight matrix $W$ creating the local field

$v_k$, assuming an initial condition of $b_k = 1$. This field $v_k$ is used by the activation function $f_k$. The output from the activation function, $y_k$, is then passed to the next layer $j$, acting as input to these neurons. To better show the flow of information, the output $y_k$ from a hidden layer is referred to as net input to the next layer, denoted as:

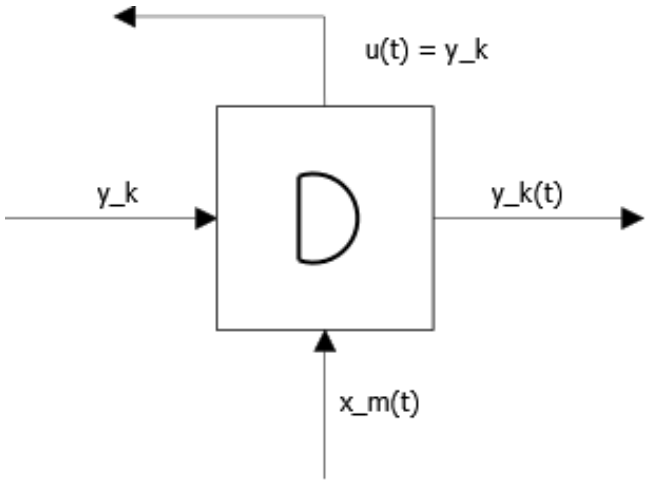
$$net_k = y_k$$

A dedicated output layer presents the final output of the network, $y_j$.
The network presented is also known as a feed-forward network, with topology of a directed acyclic graph, and the neuron constructed as a *perceptron*. This kind of network is often used in classification problems. Take for example (Zhang et al., 2010) as an economic application. A three-layer feed-forward network is used to classify the probability of a recession. By mapping values of different economic indicators to a binary recession indicator in a training period, the network can predict a probability of recession based on unseen test values of the same indicators. The network receives the indicator values in the input layer. The recession indicator, with a value of $[0 \text{ or } 1]$, is set as the output. The network iteratively updates the weight matrix, $W$, transforming the values of the different indicators flowing through the network and producing a recession probability. The out-of-sample values of the indicators are then fed to the network and transformed by the weights to a resulting output. This output is a probability of recession between 0 and 1.

**Recurrent Neural Networks**

A time dimension can be added by using a recurrent network structure, that facilitates a *Delay* component to the structure (Hagan et al., 2014). This blocks makes it possible for earlier outputs of the network, to influence future outputs and thereby opens for the possibility of lagged dependencies in economic data.
The delay module shown in Figure 2.5 receives $x_m(t)$ as initial input. Simply meaning that the delay operation does not have any influence on the output in the first iteration. A copy of the output $y_k$, referred to as $u(t)$, is sent back to the network and used as input for the next iteration. The copy that is sent back becomes $u(t) = y_k(t-1)$ opening for the possibility of basing the output of the neuron, not only on $x_m(t)$, but also $y_k(t-1)$.
The recurrent structure can be constructed as presented in Figure 2.6.
It is shown that the input $x_m(t)$ is send directly to the delay module, giving an initial output $y_k(0) = x_m$. This is fed back into the recurrent layer and multiplied by the weight matrix $W$ where weight connections from the delay module is now present. Summed with the bias, $b_k$, it creates net input $v_k(t)$. The activation function receives this input and passes its output, $y_k(t)$, to the delay function that gives the output to the next layer, this time renamed $net_{yk}$. It also feeds the net input

**Figure 2.5:** Delay




*Source: Figure 2.11 - Delay Block (Hagan et al., 2014)*

**Figure 2.6:** Discrete-time Recurrent Network

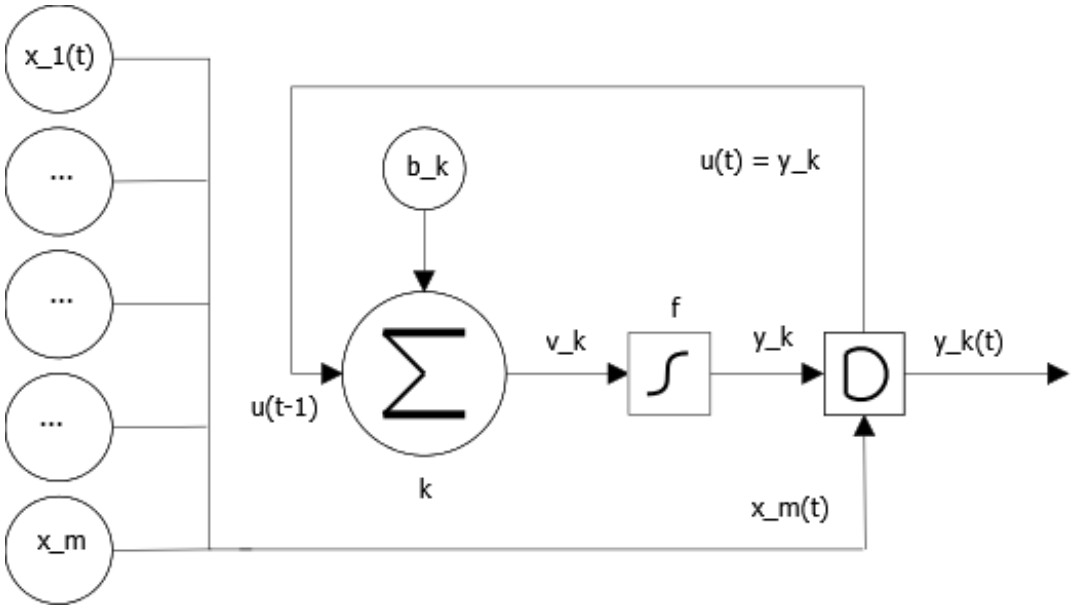



*Figure 2.13 - Recurrent Network (Hagan et al., 2014)*

back into the recurrent layer. The output of the neuron is calculated as:

$$net_{yk}(t) = f_k(Wy_k(t-1) + b_k)$$

The use of layer reference $k$ may not immediately be evident in a recurrent network, but more complex networks can combine different sub-networks, making layer reference highly useful.
Recalling the biological memory framework presented by Atkinson and Shiffrin, the delay module facilitates the "memory" of the computer. While the feed-forward network can capture a pattern and reproduce it with the weight matrix, "memory" is captured in recurrent networks. The idea of earlier synaptic activity is modelled in the recurrent networks by feeding the output back into the model and basing $y_k(t)$ on $y_k(t-1)$ created by earlier output, specifically by the activation function and output function. In this way the recurrent network learns not only patterns in the mapping of inputs to outputs, but also how earlier patterns affect current patterns. The weight matrix is then essentially a loading matrix much like the one used in the dynamic factor model in section 3.1.

### 2.4.2 Activation Functions

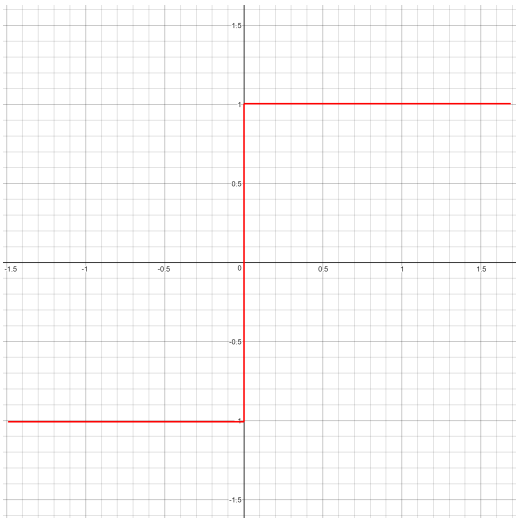
The "on/off" switching of neurons in the biological analogy is imitated by activation functions in the artificial network. The functions are central to the functionality of the networks because they act as gates determining the activity of a neuron. It maps the input to a limited range, making the computation bounded, stable and serparable in the induced local field (Hecht-Nielsen, 1992) (Mitchell, 1997) (Haykin, 1999). Different activation functions exists, and different functions are useful for different applications. In general three basic categories exists; Threshold-, Piecewise-Linear- and Sigmoid functions (Haykin, 1999).

**Threshold Functions:** The threshold function is boolean by nature. It outputs a value depending on the input relative to a threshold. In the example of a simple perceptron shown in Figure 2.3 the function output can be formulated as:

$$f(a) = \begin{cases} 1 & \text{if } a > 0 \\ -1 & \text{otherwise} \end{cases}$$

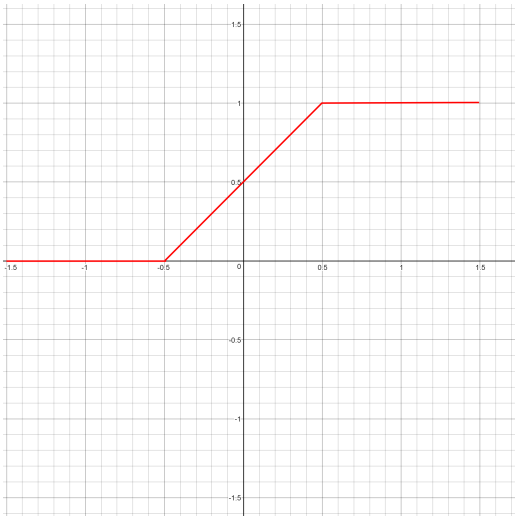
In this example an antisymmetrical hard-limit activation function is used to limit the output to $[-1 \text{ or } 1]$. The linear combination of inputs in weight space, $a = \sum_{m=0}^{k} w_{m,k}x_m + b_k$, has to exceed the threshold of 0 to output 1, else it outputs $-1$ (Mitchell, 1997). Different threshold functions exists, all mapping the net input to output values by a boolean OR operator. Simply meaning that the function maps output based on a *"one-or-nothing"* approach (Haykin, 1999). Figure 2.7 shows

**Figure 2.7:** Anti-symmetrical Hard Limit Function $[-1 \text{ or } 1]$



*Source: (Mitchell, 1997)*

**Figure 2.8:** Piece-Wise Linear $[0:1]$



*Source: (Haykin, 1999)*

the anti-symmetrical hard-limit activation function limit allowing negative output values.

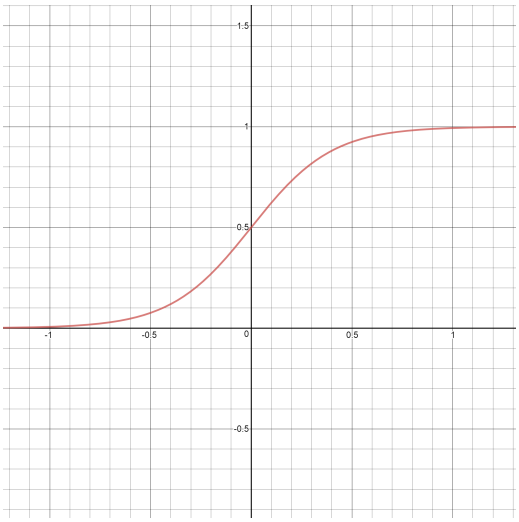
**Piecewise-Linear Functions:** The piecewise-linear function uses a mapping range, allowing the space, $a$, to pass through if it stays within the boundaries of the activation function. An example function can be formulated as:

$$f(a) = \begin{cases} 1 & \text{if } a \geq +\frac{1}{2} \\ v & \text{if } 1 + \frac{1}{2} > a > -\frac{1}{2} \\ 0 & \text{if } a \leq -\frac{1}{2} \end{cases}$$

The piecewise-linear activation function can consists of different linear segments with different slopes, allowing approximation to non-linear functions with simple linear methods. If $a$ is infinitely large the activation function reduces to a threshold function (Haykin, 1999). Figure 2.8 shows the piecewise-linear example.

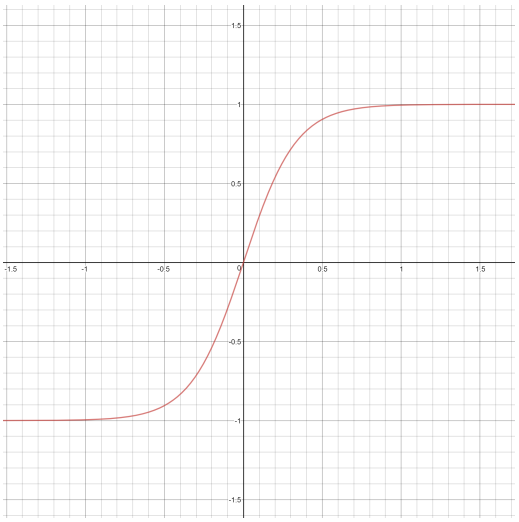
**Sigmoid Functions:** The sigmoid shaped activation functions are the most commonly used functions, because they are differentiable by nature (Haykin, 1999). Both Grossberg (Grossberg, 1980) and Rumelhart & McClelland (Rumelhart et al., 1986) advocates the use of sigmoid shape in non-linear networks because of the derivatives ability to capture non-linear functions in multiple layers. The output of the function lies within a range of $[0:1]$ or $[-1:1]$, depending on the function used, giving a continuous output. Following the biological analogy the use of a

**Figure 2.9:** Logistic Function $[0:1]$




*Source: (Haykin, 1999)*

**Figure 2.10:** Hyperbolic Tangent $[-1:1]$




*Source: (Haykin, 1999)*

range facilitates varying intensity of "firing" between neurons. This makes the class of functions useful for continuous input and regression problems, as some previous outputs may have more impact on future outputs, than others. Two popular sigmoid activation functions are the logistic function and the hyperbolic tangent function.
The logistic function is shown in Figure 2.9 defined as:

$$f(a) = \frac{1}{1 + \exp(-\alpha a)}$$

A slope, $\alpha$, increasing to infinity, transforms the sigmoid function into a threshold function. For all other $\alpha$'s the output can take on a value from the range $[0:1]$. To allow negative output values of the activation function, the hyperbolic tangent shown in Figure 2.10 is used, essentially just providing a rescaled and shifted logistic function (Haykin, 1999). It is defined as:

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

The use of antisymmetrical activation functions, like the hyperbolic tangent in range $[-1:1]$, has shown to yield better results than nonsymmetrical functions like the logistic function in range $[0:1]$ (Glorot & Bengio, 2010).

### 2.4.3 Backpropagation

Up until now the flow of information presented in the PCP framework shares conceptual analogies to the flow of activity in the brain. This is however also where the analogy ends. Any deeper mechanisms of the network shifts focus from realism to computational implementation.

In section 2.2 a learning machine was defined as one that increases its performance at a task with experience. This is true for an artificial neural network in the way it update weights, $W$, and bias $b$ until it can predict accurately. Essentially, the machine is solving an optimization problem of choosing the weights that provides the lowest overall loss, compared to the target training output. This is done by the *Back-propagation* algorithm with gradient descent, based on Rumelhart & McClelland's *"Delta Rule"* (Rumelhart et al., 1986). The algorithm computes the weights in two passes, a forward- and backwards pass.

**Forward Pass** In the forward pass the time series are fed through the network and weights randomly initialized with small numbers (Mitchell, 1997). Output is calculated neuron-by-neuron as defined in equation (2.1). A single hidden layer $k$ is assumed for simplicity, resulting in the output layer being $j$ :

$$y_j = f_j(Wx_m + b_j)$$

with

$$v_j(t) = (W_j x_m + b_j)$$

$v_j(t)$ being a weight space induced by the sum of weighted inputs plus the bias. Input $x_m$ is output fed from the earlier layer. Furthermore, observable neurons in the output layer produces an error signal at time $t$ the size of the difference between the target output, $y_{trg}(t)$, and the computed output from the neuron, $y_j(t)$:

$$e_j(t) = y_{trg}(t) - y_j(t) \tag{2.2}$$

An instantaneous error per neuron is defined as $\frac{1}{2}e_j^2(t)$ with the total instantaneous error in the output layer thus being:

$$\epsilon(t) = \frac{1}{2}\sum_{j=C} e_j^2(t) \tag{2.3}$$

$C$ defining the set of neurons $j$ in the output layer, resulting in the average squared error taking the form:

$$\epsilon_{av} = \frac{1}{N}\sum_{t=1}^{N} \epsilon(t) \tag{2.4}$$

$N$ is the number of observations per input series, and $t$ the time step. The average squared error $\epsilon_{av}$ is the cost function of the training dataset, that the network seeks to minimize. The forward pass ends when the error signal is computed at the output layer (Rumelhart et al., 1986).

**Backward Pass** The error is sent backwards through the network starting from the output layer. At each neuron the weight is changed to reduce the error received according to Rumelhart & McClelland's delta rule. This updating rule relies on a learning parameter, a local gradient in the induced weight space, and an input value from an earlier neuron. The change that has to be made to the weight in order to reduce the error signal is defined as:

$$\Delta w_{jk} = \eta \delta_j(t) x_m(t) \tag{2.5}$$

$\Delta w_{jk}$ is the change to be made to the weight proportional to the partial derivative $\frac{\partial \epsilon}{\partial w_{jk}}$ of the error. $x_m(t)$ is the input values. This would be $net_{yi}(t)$ if the neuron was placed in hidden layer $k$ of a multi layer network, with layer $i$ to the left. $\delta_j(t)$ is the error gradient received and $\eta$ the learning rate of the network chosen by the designer at setup.
The specification of the local gradient, $\delta_j(t)$ depends on the type of neuron. If the neuron is part of the final output layer the gradient is constructed from $e_j$ and the partial derivative as:

$$\delta_j(t) = e_j(t) f_j'(v_j(t)) \tag{2.6}$$

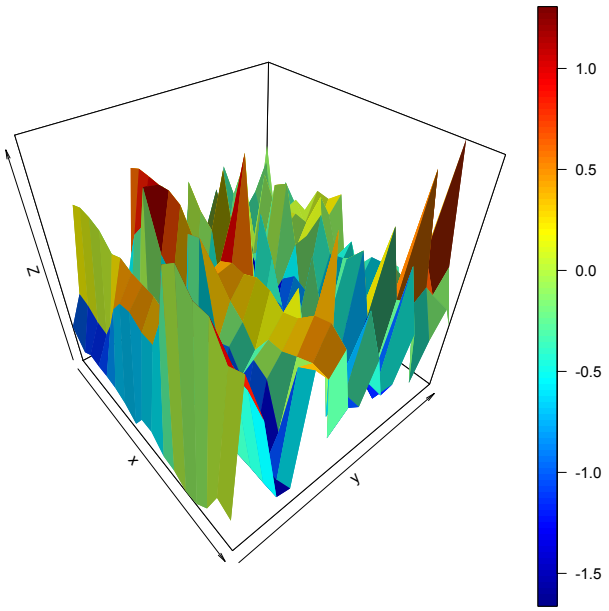
$f_j'$ being the derivative of the activation function and $v_j(t)$ the induced weight space, that the gradient $\delta_j$ is part of. $e_j$ is easy to calculate according to 2.1, as the neuron is shown both $y_j(t)$ and $y_{trg}(t)$.
If the neuron is part of a hidden layer no directly comparable target output is presented, so the error signal received is determined from the error signal already computed. This is done by:

$$\delta_k(t) = f_k'(v_k(t)) \sum_j \delta_j(t) w_{jk}(t) \tag{2.7}$$

With $k$ being left of $j$. The local gradient $\delta_k(t)$ for neuron $k$ in the hidden layer is constructed from the sum of the already computed error gradients in the layer to the right of the hidden neuron. This is multiplied by the corresponding weight $w_{kj}$ between the neurons in the two layers. Thus giving a backwards flow of error signals in the network (Rumelhart et al., 1986).
Intuitively, in an economic context, the value of the different indicators are therefore sent through the network with weights randomly chosen at initialization, and an overall error between the GDP growth prediction of the final output layer is compared to the real GDP growth from the training data. The error between these

is then sent back and weights are adjusted one layer at a time. Each neuron is deciding to vote for which direction to adjust their individually connected weights, in order to bring GDP growth closer to the known target GDP growth. The amount of adjustment is large at first and gets incrementally smaller when the neuron is close to the target GDP. The algorithm "knows" when it is close because of the gradient. When the descent becomes more and more shallow with each step in one direction, it is a sign of the weight reaching a minimum. Figure 2.11 illustrates an example gradient with different minimums.

**Figure 2.11:** Example of a 3D surface gradient




*Source: Constructed from random simulation*

**Back-propagation Through Time (BPTT)** In a recurrent network structure the algorithm is modified to handle time. The result is the back-propagation through-time algorithm. A forward pass of the training set, starting at $t_s$ and ending at $t_N$, is fed through the network. All input- output pairs and weights of the network are saved, and a single backwards pass of the network calculates the gradients as:

$$\delta_j(t) = -\frac{\partial \epsilon_{total}(t_s, t_N)}{\partial v_j(t)} \tag{2.8}$$

Where the total loss is defined:

$$\epsilon_{total}(t_s, t_N) = \frac{1}{2} \sum_{t=t_s}^{t_N} \sum_{j=\Lambda} e_j^2(t) \tag{2.9}$$

$\Lambda$ denoting the index of a neuron placed in the output layer at a specific time step i.e. neuron 1 at time 1 being index 1, 2 at time 1 being 2 and so on. Depending on the placement of the neuron(output or hidden) the local gradient in the recurrent configuration also depends on the iteration. For the last observation, when $t = t_N$, the gradient is simply:

$$\delta_j(t) = f'(v_j(t))e_j(t) \tag{2.10}$$

While the gradient for any observation $t_1 < t \leq t_N$ is:

$$\delta_j(t) = f'(v_j(t))[e_j(t) + \sum_{k=\Lambda} w_{jk}\delta_k(t+1)] \tag{2.11}$$

Assuming that the activation function $f$ chosen by the designer, is the same for all neurons. The adjustment to the weight, when the errors are propagated back to $t_s + 1$, is made as:

$$\Delta w_{jk} = \eta \sum_{t=t_s+1}^{t_N} \delta_j(t)x_m(t-1) \tag{2.12}$$

By sending all sets of observations through the network and calculating the total error $\epsilon_{total}$, the error minimization is finding those minimums that reduces the error the most in all of its history, and not only at the current step.

### 2.4.4 Notes on Back-propagation

Some important features to note on the back-propagation algorithm and gradient descent is:

1. **Activation function:** As shown, the activation function has to be differentiable. Furthermore, activation functions between $[0, 1]$ results in ever increasing adjustments to the weights, causing the gradient to saturate. Antisymmetric activation functions like the hyperbolic tangent $[-1, 1]$, and random initial weights around the mean of the function, solves this to some extent. The hyberbolic tanget has shown good performance in different recurrent structures in the litterature (Glorot & Bengio, 2010).

2. **Saturation of the gradient:** Given a non-linear activation function that squashes a linear combination of input data into a given range, saturation can happen. Inputs having a value close to the infinite boundaries of the activation function results in relatively smooth gradients; the activation function converges to the limit and allowed change of the weight is very limited. With only

a limited number of steps in the gradient search, such a situation poses a computational problem. The weights cannot adjust enough in the limited attempts available. In lucky cases where the output of the network is so close to the target output, that little has to be done to the weight, an unchanged weight is no problem. This is, however, seldom the case, so flexibility in the weight is desirable. The network is charchterized as somewhat too generalized. (Haykin, 1999)

3. **Overfitting:** Opposite to a saturated gradient, is a gradient with too many different protruding points of different heights. This happens with training, as some weights gets changed while others do not. By training too much, too much noise is included in the gradient. This creates a decision gradient that is too specific to the presented training data and thus too complex for generalizing to unseen data. This is also referred to as *overfitting*. Following a human analogy, the network essentially focus too much on learning the training data, only to forget that inputs can take on other values than those presented. The network becomes too confident in it's ability to explain everything, based on the training data. When presented with never before seen input values, it tries to fit the new value to something similar it already knows, often resulting in poor output predictions. (Mitchell, 1997)

4. **Local Minimum:** With non-linear activation functions the error gradient can have multiple minimums. This poses a problem when weights are randomly initialized, because the nearest minimum that the weight converges to, may only be local and not global. This gives a prediction, but not necessarily the most accurate. Identical training runs produces different predictions, with the starting point being the only difference. The use of instantaneous error and average instantaneous error in the back-propagation algorithm makes it stochastic by nature. Averaging all individual gradients from all training samples, somewhat limit the magnitude of local gradient noise (Mitchell, 1997).

5. **Vanishing or exploding gradients:** In non-linear recurrent networks, gradients vanish over time. The combination of non-linear functions result in a temporal evolution of the error, that depends exponentially on the weights (Hochreiter & Schmidhuber, 1997)(Gers et al., 1999). An input from several time steps back produces only a limited, if any, change in the current weight because of this exponential evolution (Haykin, 1999). The network has a hard time learning long term dependencies because of this, and is limited to short term memory. Mainly because the effect of the distant inputs becomes infinitesimal relative to the recent inputs.

### 2.4.5 Long Short-Term Memory (LSTM)

Sepp Hochreiter and Jürgen Schmidhuber presented a solution to this in 1997; The *Long Short-Term Memory* (LSTM) network (Hochreiter & Schmidhuber, 1997). This type of recurrent network employs different *memory cells*: a *"Constant Error Carrousel"*, or CEC, and different gates. The CEC facilitates the networks ability to remember some information and forgetting other. A given composition of different cells is referred to as a *memory block*. To ease the differentiation between the term "cell" and "block", a composite block is simply referred to as a neuron when confusion could arise.

**The Constant Error Carrousel (CEC):** Re-writing equation (2.11) for a single neuron in a hidden layer, gives:

$$\delta_k(t) = f'_k(v_k(t))w_{kk}\delta_k(t+1) \tag{2.13}$$

To keep the recurrent error flow constant, instead of exponential, it is required that:

$$f'_k(v_k(t))w_{kk} = 1 \tag{2.14}$$

The integral of the activation function $f$ thus becomes:

$$f_k(v_k(t)) = \frac{net_y i(t)}{w_{kk}} \tag{2.15}$$

Meaning that the activation has to be linear and constant:

$$y_k(t+1) = f_k(v_k(t+1)) => f_k(w_{kk}y_k(t)) => y_k(t) \tag{2.16}$$

The CEC is placed in the neuron. It can activate the neuron and keep it activated, for as long as it is helpful in reducing the total error of the network. This is determined by the error signal received from other neurons. Adding another neuron is therefore NOT without problems.

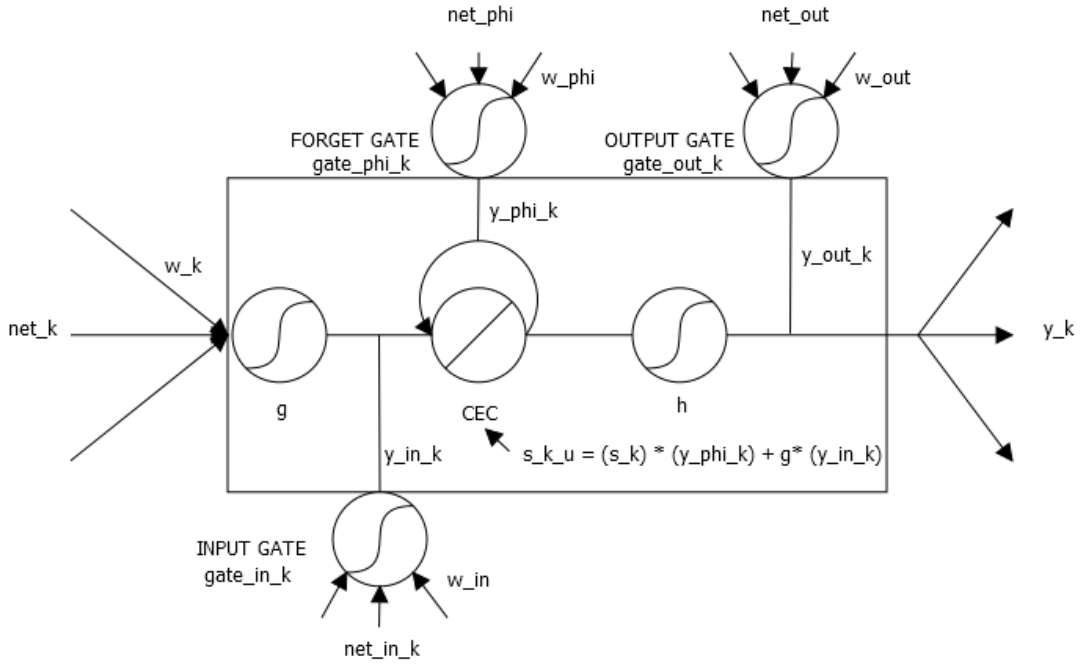
- An *input weight conflict* arises because one neuron with a CEC, connected to multiple neurons by weight $w_{kj}$, will receive different error signals, competing for the state of the neuron. It either has to be active to keep weight updating signals for later, or ignore the signal. In other words, neuron $k$ may end up receiving two signals at the same time, one which tell it to remember its current state and one which tells it to ignore its state. The neuron $k$ can however only do one such thing at a time, so it has to learn when to remember and when to ignore.
  When the CEC rememebers information, it is trapped inside. It can flow unmodified through time inside the CEC and have an effect on new net inputs $net_{c_j}$, but it cannot change (Hochreiter & Schmidhuber, 1997).

In an economic context the CEC allows the network to remember lagged dependencies by storing them, much in the sense of including a lagged value in an econometric model.

- An *output weight conflict* exists because a neuron already activated, acts through a weight to other neurons $w_{ki}$. The output weight can either send a signal to $i$, asking it to remember information, or protect $i$ from confusion by not sending any information at all.

The LSTM network constructs a neuron by using different gates; an *input gate* and an *output gate* to handle the weight conflicts.
In the 1999 paper by Gers, Schmidhuber and Cummins (Gers et al., 1999) a *forget gate*, is implemented to resolve some of the problems with the naive LSTM. With a reccurent weight of the CEC at 1.0, and a truncated weight updating approach, the LSTM is local in space and time (Hochreiter & Schmidhuber, 1997)(Gers et al., 1999). This means that the network is looking only at some part of the gradient at a time, allowing errors outside to vanish or explode. Where the LSTM solves the problem of standard recurrent networks inability to remember long term dependencies, the naive LSTM has a hard time forgetting these dependencies when they are no longer relevant. This is solved by the forget gate, as the 1.0 weight of the self-recurrent connection in the CEC is replaced with this gate. Instead of fixing the start value at 1.0 the forget gate starts close to 1.0 and is allowed to decay over time as the network learns to forget, by descending on its own gradient. It is hereby gradually *"resetting"* the CEC instead of trapping information and keeping it.
In an economic analogy the CEC allows the network to remember lagged dependencies by storing them, much in the sense of including a lagged value in an econometric model. This is done by the input- and output gate when the lag appears significant. Some lags can however loose their significance by including or excluding other lags. The forget gate helps removing these now insignificant lags by resetting the CEC.
All gates are multiplicative in the sense that an input is multiplied by an activation value to determine what is allowed to flow through the gate. The memory cell is shown in figure 2.12.
A self-connected linear CEC is the center piece of the neuron. The self-connecting aspect allows the CEC to feed the current CEC-cell state back into the itself, thus opening for the possibility of remembering the state of $(t-1)$. Following the flow of signals in Figure 2.12, the neuron $k$ receives net input $net_k$ and corresponding weights $w_k$. This is squashed by a function $g$ and multiplied by the signal from the input gate $y_{in_k}$. The effect of the input gate on the net input is determined by squashing earlier gate signals $gate_{in_k}$ and their weights. This $g * y_{in_k}$ is then fed

**Figure 2.12:** The LSTM Memory Block




*Source: Figure 2 (Gers et al., 1999)*

to the CEC. The earlier state of the cell is added to $g * y_{in_k}$ and the forget gate is multiplied on, giving a current cell state of:

$$s_{k_u}(0) = 0$$

and for $t > 0$:

$$s_{k_u}(t) = y_{\phi_k} s_k(t-1) + y_{in_k}(t) g(net_k(t)) \tag{2.17}$$

The output of the neuron is then:

$$y_k(t) = y_{out_k}(t) h(s_k(t)) \tag{2.18}$$

With $h$ being a scaling function that scales the internal state $s_k$. This is multiplied by the signal from the output gate, $y_{out_k}(t)$ creating the output. The gate is evaluating if the output from the cell is relevant and should be send on to the next neuron (Gers et al., 1999).

**The Gates:** The input, output and forget gates produces an output defined as: Input:

$$y_{in_k} = f_{in_k}(gate_{in_k}(t))$$

Output:

$$y_{out_k}(t) = f_{out_k}(gate_{out_k}(t))$$

Forget:

$$y_{\phi_k}(t) = f_{\phi_k}(gate_{\phi_k}(t))$$

With all gate inputs defined by the delayed gate outputs and weights from connected neurons:

$$gate_{out_k}(t) = \sum_u w_{out_k u} y_u(t-1)$$

$$gate_{in_k}(t) = \sum_u w_{in_k u} y_u(t-1)$$

$$gate_{\phi_k}(t) = \sum_u w_{\phi_k u} y_u(t-1)$$

and net input received, simply calculated from earlier neurons and their weights:

$$net_k(t) = \sum_u w_{ku} y_u(t-1)$$

$u$ denoting any connected neuron in the network no matter layer placement.

This flow constitutes the forward pass in the LSTM network. The learning is still happening in the backward pass when errors are adjusted. Learning follows a modified truncated back-propagation through time-algorithm. When the error reaches the neuron, an internal state error, $\epsilon_{s_k}$, is necessary. This requires knowledge of how much each gate and earlier cell state influences the current state, which is obtained by splitting the gradient. For any layer in the network with more than one neuron, the error for the cells are given by:

$$\epsilon_{s_k}(t) = y_{out_k}(t) h'(s_k(t))(\sum_j w_{jk} \delta_j(t)) \tag{2.19}$$

This is calculated for each cell in the layer. The updating of the weights for the state cell itself relies only on the partial derivative of its own state:

$$\Delta w_{km}(t) = \eta \epsilon_{s_k}(t) \frac{\partial s_k(t)}{\partial w_{km}} \tag{2.20}$$

While the weight update for each gate also relies on the sum of errors of all cells in the given layer [1]:

$$\Delta w_{lm}(t) = \eta \sum_{k=1}^{s_k} \epsilon_{s_k}(t) \frac{\partial s_k(t)}{\partial w_{km}} \tag{2.21}$$

$l$ noting that the change is for both input gate $gate_{in_k}$ and forget gate $gate_{\phi_k}$. $m$ is used as all input from another neuron $u$, coupled with the respective gate in $l$.

[1] See (Gers et al., 1999) for the derivative split and partial differencing that gives the weight updating.

### 2.4.6 Summary

Different types of neural networks exists all having their own strengths; The feed forward network is useful in simple classification problems, and recurrent networks handles time series in continual problems. Regardless of network topology, a neural network consists of neurons interacting with each other through weight connections. Each neuron is constructed from an activation function that modifies input to an output. The weights are adjusted according to an error calculated from the difference between the output of the network and the target output from the training data. The error is presented to the network, starting at the end. Each connected neuron then votes on how to change the weight so that the overall error is reduced. This is done in each neuron by finding their own steepest descent towards an error minimum in weight space. In recurrent networks this results in exponential evolution leading to vanishing- or exploding gradients. The recurrent networks is in this sense trained recurrently, but not learning long-term dependencies. LSTM networks solves this by introducing memory cells contained in memory blocks. These enable the network to store information long-term. They do, however, not know when the information becomes irrelevant. Imagine a network that thinks Q1 GDP from 20 years ago is just as relevant as GDP last quarter. It is unlikely. This is solved by adding the forget gate to the LSTM. It allows the LSTM to learn when to forget information. The LSTM with forget gates starts off by receiving input from a training data set. It then sends it through a hidden layer that produces an output. In the hidden layer several memory blocks exists, consisting of different memory cells. These memory cells receive input, modifies it and produces output. The input to these cells are multiplied by a gate value to determine how important the input is. It is then passed through the core of the neuron; the CEC. The CEC stores a copy of the training input and saves it for later. The input is then multiplied by the output gate value, determining if the output is allowed to exit the cell or not. The output allowed to pass through to the final output layer is compared to the target output also supplied with the training data. From this, an error term is calculated. Just like the standard recurrent network, but with different handling of inputs in the memory block. This error is then sent back to the earlier layer, arriving at the output gate. The goal here is to reduce the overall error of the neuron. Since the output of the neuron depends on the input- and forget gate, they both make adjustment to their own weights. Doing so by their own gradient descent. By adjusting the weights recurrently and allowing the weight of the forget gate to change with time, the network is slowly forgetting the information it stored, depending on how much it contributes to the overall error of the network at the current time step. The network is effectively looking at a pair of input- and output values from training data and learning the patterns of the data.

# Chapter 3

# Benchmark Dynamic Factor Model

## 3.1 Dynamic Factor Models

Presented in (Stock & Watson, 2010), factor models provide a solution to the problem of macroeconomic data. The idea behind the models are that $r$ number of latent common factors $F_t$ drive comovement between series in a vector $X_t$ of $N$ variables. Each series may be subject to zero-mean idiosyncratic disturbances $e_t$
The vector $X_t$ is given:

$$X_t = \Lambda(L)F_t + e_t \tag{3.1}$$

and factors:

$$F_t = \Psi(L)F_{t-1} + \eta_t \tag{3.2}$$

$\Lambda(L)$ is the dynamic factor loading matrix $X_t$ of size $(N \times r)$, with $\Lambda_i$ being the loading for series $X_{it}$. $\Lambda_i F_t$ is the common component for series $i$. $\Psi(L)$ is a lag polynomial matrix of size $r \times r$, acting as a loading matrix for lagged values. Factors are asumed to be vector autoregressive system and all processes are assumed to be stationary. $e_t$ and $\eta_t$ are assumed to be uncorrelated with all lags $k$ such that $E[e_t\eta'_{t-k}] = 0$.
With a squared error loss the optimal one-step forecast is given[1]

$$E[X_{it+1}|X_t, F_t, X_{t-1}, F_{t-1}, ...] = \beta(L)F_t + \delta(L)X_{it} + \epsilon_{t+1} \tag{3.3}$$

Essentially opening for the possibility of basing the forecast on not only lagged values of the series, but on more than one predictor with $Lambda_{it}$.
(Stock & Watson, 2010) divides the evolution and estimation of dynamic factor models in in three generations. The first generation is based on maximum likelihood, state space modelling and kalman filtering. The second generation consists of non-parametric averaging methods such as principal components. The

[1] See (Stock & Watson, 2010, p.4) for derivation

third generation is a hybrid of principal components and state space construction. For bencmarking purposes the principal component method outlined in (Stock & Watson, 2002) and (Giovannelli & Proietti, 2014) is followed. This is due to the diminishing computational efficiency of parametric estimation in the state space representation, when the number of coefficients becomes too high (Stock & Watson, 2010). It is further relevant because it is focused on prediction, rather than structural inference and feature selection, thus matching the LSTM presented. The articles referenced in section 4 shows that the hybrid of principal components and kalman filtering is a popular method used in nowcasting different economies because of the filters ability to handle incomplete data. *This specific application of a dynamic factor model is, however, beyond the scope of simple benchmarking*.

### 3.1.1 Principal Components Regression

Following (Stock & Watson, 2010) predicting GDP with principal components in a regression involves two steps:

1. Estimating common factors and principal components from an eigenvalue decomposition of the covariance matrix, ordered according to the eigenvalue.
2. Selecting components based on cross validation, and use these as regressors on the dependent variable.

The estimation of the components is done by transforming the dynamic factor model into a static factor model. Equation (3.1) is kept the same:

$$X_t = \Lambda F_t + e_t$$

While the predicted one step ahead dependent variable is defined:

$$y_{t+h} = \beta_F' F_t + \beta_w' W_t + \epsilon_{t+h} \tag{3.4}$$

$\beta$ being the regression coefficients, $\epsilon_t$ the disturbances, $F_t$ the factors and $w_t$ the weight matrix. $e_t$ is assumed to be uncorrelated resulting in a variance matrix for $X_t$:

$$\hat{\Sigma}_X = T' - 1 \sum_{t=1}^{T} X_t X_t' \tag{3.5}$$

A matrix of eigenvalues $\hat{\Lambda}$ with the largest values for $\hat{\Sigma}_X$ is constructed with $r$ elements. The estimation of the principal components thus becomes a least squares minimization problem:

$$min_{F_1,\dots F_T,\Lambda} V_r(\Lambda, F) \tag{3.6}$$

With:

$$V_r(\Lambda, F) = \frac{1}{NT} \sum_{t=1}^{T} (X_t - \Lambda F_t)'(X_t - \Lambda F_t) \tag{3.7}$$

Normalizing $N^{-1}\hat{\Lambda}'\hat{\Lambda}$ to indentity $I_r$ and setting $\hat{\Lambda}$ to the $r$ largest eigenvector, gives a least squares $F_t$ estimator that is[2]:

$$\hat{F}_t = N^{-1}\hat{\Lambda}' X_t \tag{3.8}$$

The factor estimate $\hat{F}_t$ is thus a vector consisting of the scaled first $r$ components. Multicollinearity is then somewhat avoided, as the factor model construct factors as linear combinations of uncorrelated indicators. An important thing to note is however, that the constructing of the factors in this way, makes them useless for structural inference. It is impossible to say how one indicator affects GDP in a certain way, because the factors are combinations of indicators. This is possible with a Principal Component Analysis method, but is unnecessary for the benchmark model in the thesis.
With factors estimatied, selection of components can be done in different ways. (Giovannelli & Proietti, 2014) presents different methods, but cross-validation is typically used as it works well with small datasets and provides measures against overfitting. It is simply validating parameters in different contexts and minimizing an error term, assuring that the parameters are generalized enough to give good out-of-sample predictions.
The prediction of GDP growth is done with equation 3.4 and 3.8

[2]See (Stock & Watson, 2010) for full derivation of the minimization problem

# Chapter 4

# Data and Models

Implementing an LSTM to economic data is somewhat straight forward. As shown i section 1, the LSTM performs well with univariate time series containing lots of observations. Here the time series is split into sequences defined by a sliding window that predicts the next observation. The window then moves to include the next observation and predicts one observation ahead once again. In a multivariate system, the LSTM is simply fed each series as a separate input and maps each series to the output variable, with weights updated to define how input should be transformed to produce the output GDP. Lagged dependencies are captured by the LSTM network in the recurrent structure of the CEC and significance simulated by the different gates.
The DFM achieves the results by storing transformations in eigenvectors. These are then used with the common factors to transform out-of-sample inputs into a GDP growth output.
This section describes the data used, the specific setup of models and the results of the models.

## 4.1 Data

To compare out-of-sample prediction performance, both the LSTM network and the (Dynamic) Factor Model (DFM) is fed the exact same data, with same training/test split. 19 indicators are used, similar to some nowcasting applications of dynamic factor models in the litterature. The number of indicators in the references are listed in Table 4.1.
Other articles nowcast GDP for France (Barhoumi et al., 2009), Indonesia (Luciani et al., 2015), Ireland (D'Agostino et al., 2011). All with a number of indicators similar to this interval; between 10 and 30. (Alvarez et al., 2016) showed that medium sized datasets with a number of indicators around this range, performs just as well as dynamic factor models fed over 100 indicators.

**Table 4.1:** No. of Indicators in Different Nowcasting Applications

| **Country** | **No. Indicators** | **Article** |
|---|---|---|
| Canada | 23 | (Chernis & Sekkel, 2016) |
| Czech Republic | 28 | (Rusnák, 2013) |
| Norway | 14 | (Luciani & Ricci, 2014) |
| Brazil | 14 | (Bragoli et al., 2014) |
| BRIC + Mexico | | (Dahlhaus et al., 2015) |
| - Brazil | 35 | |
| - Russia | 14 | |
| - India | 29 | |
| - China | 13 | |
| - Mexico | 41 | |

As stated in section 3.1 focus is not on specific applications of dynamic factor models like nowcasting, but general benchmarking. *All data is therefore quarterly and not of mixed frequency, and the underlying estimation method is different from those in the articles, as specified in section 3.1.*
Indicators are often a mix or hard and soft economic variables. Hard indicators coverer real economic data such as production, labor and investment (Rusnák, 2013), while soft indicators includes bond rates, house prices and share prices. Key indicators and their unit of measurements are chosen for the danish economy with reference to *Outlook For The Danish Economy - September 2017* published by the danish central bank (Nationalbank, 2017). The indicators are presented in Table 4.2. Log transformation and differencing is done to relevant variables when necessary.

Indicators like CPI, short- and long-term interest rates, employment, consumption, import- export and balance of trade are common components used when dealing with GDP. These are almost all featured in the nowcast articles, and all present in the economic outlook by the danish national bank. Oil price, total share price and exchange rate are used as financial components. The exchange rate is against american dollars as this currency is used internationally and can explain some of the dynamics driving international trade. Volatile movements in the exchange rate further poses a risk for firms dealing internationally, so the exchange rate reflects a measure for this. The currency pair could have been against the Euro, but due to a fixed exchange rate regime this would not provide any relevant information.
The building permits of residential housing is included as a measure for private investment. A growing number of building permits reflects a growing amount of private investment. Housing prices is also used to reflect this information. The same goes for total credit to Non-financial Corporations. It reflects loans taken by non-financial firms, showing the amount of corporate investment given the as-

**Table 4.2:** Indicators for the Danish Economy

| **Indicator** | **Unit** | **Source** |
|---|---|---|
| Gross Domestic Product | Mio. of DKK | FRED |
| Consumer Price Index<br>- Total All Items | Pct. Growth - YoY | FRED |
| Employed Persons<br>Age 15 and Over | 1.000 Persons | FRED |
| Unemployed Persons<br>- Total Harmonized | 1.000 Persons | FRED |
| 3-Month Interbank Rate | Pct. Per Annum | OECD |
| 10-Year Long-Term<br>Government Bond Yields | Pct. Per Annum | OECD |
| Total Share Prices | Index 2010 = 100 | OECD |
| Brent Crude Oil<br>- Global Price | USD pr. Barrel | FRED |
| Exchange Rate<br>- DKK/USD | DKK to USD | FRED |
| Building Permits<br>- Residential | Pct. Growth - Prev. | FRED |
| Total Credit<br>- Non-Financial Corporations | Billions of DKK | FRED |
| Housing Prices | Index 2010 = 100 | FRED |
| Hourly Earnings DK<br>- Manufacturing | Pct. Growth - YoY | FRED |
| Hourly Earnings EU<br>- Manufacturing | Pct. Growth - YoY | FRED |
| Government Consumption | Pct. Growth - Prev. | FRED |
| Private Consumption | Pct. Growth - Prev. | FRED |
| Import of Goods & Services | Mio. of DKK | DST |
| Export of Goods & Services | Mio. of DKK | DST |
| Balance of Trade | Mio of DKK | DST |

*YoY = Year-on-Year - The same period the previous year*
*Prev. = The previos period*
*Sources: FRED = fred.stlouisfed.org ; OECD = stats.oecd.org ; DST = statbank.dk*

sumption that loans are used for investment purposes.
The growth in hourly wage for the manufacturing sector is included for both Denmark and EU, reflecting the overall national level. The growth level for EU is also included, as this reflects a measure of international competitiveness.

Differencing is done to achieve $I(0)$ stationary series for better prediction.
It may be the case that some series are cointegrated. Error correction terms can be used to handle these conintegrated series (Stock & Watson, 2010). (Banerjee et al., 2017) suggests that including such error correction term may aid the short run forecast. But with a dataset of many predictors, a mix of $I(0)$ and $I(1)$ series is likely to be present. It further creates a problem for the error correction term if structural breaks are present, which is the case for this data. So while the dynamic factor model may suffer a loss of strength by not including long term relationships and error correction terms, the difference stationarity suffice for short run prediction.

## 4.2 LSTM for Danish GDP

The LSTM is programmed in R with the Keras interface for high-level interpretation. Google's Tensorflow is used as backend handling the actual computation. This makes the development of the network relatively easy. The R code for the LSTM and DFM is shown in appendix A.

### 4.2.1 Training The Network

The practical training of the LSTM network can be done in different ways. The one outlined in this thesis is referred to as *batch* training.
One complete pass of a training dataset is referred to as an *epoch*. Weights are updated on an epoch-by-epoch basis. Referring to the use of the interval $[t_s : t_N]$ in equation (2.8), the training data does not necessarily have to be the entire dataset. In this sense the network in trained from batches of data. The recurrent version of Back-propagation presented in section 2.4.3 is based on batch training. The presentation of the ordinary feed-forward back-propagation in 2.4.3 is not. This approach shows a *sequential algorithm*, in which training pairs are presented one at a time and weights updated according to this. Essentially using a batch size of 1. The training is stopped once the out-of-sample prediction accuracy is converging. This provides a measure against overfitting the network, as the accuracy starts to grow when this is happening.
The LSTM network has few, but critical, requirements to the data it is fed:

- **Scaling:** Given the limited range of values an activation function can operate in, it is absolutely necessary to scale the data that is fed to the network.
  To avoid a "looking ahead" bias, the scaling has to be made only on the

training split, and not the entire dataset. The test split can then be scaled and normalized according to the scaling attributes of the training set.

- **Shaping Input:** The input to the network is required to be in three dimensions; The number of features, the number of sequences in each feature and the number of observations in each feature. In the context of the presented data, features would be the different indicators selected.
  The sequence splitting is usually done because the LSTM has a hard time learning relatively long sequences at once (Fischer & Krauss, 2017). Time series with length of say, a 1000 observations, has to be split into smaller sequences of fewer observations. In the case of the data presented with 92 observations in total, and a training size of 72 observations, this is not necessary.

### 4.2.2 Parameters

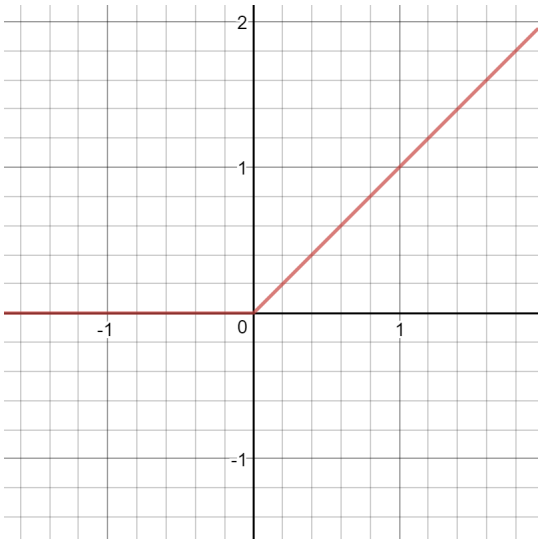
The parameters used in the LSTM network for the danish economy is tuned to produce minimal error between network GDP growth output and target out-of-sample GDP growth. This gives a network with parameters as follows:

- **1 LSTM block with 19 units ; 1 output layer with 1 neuron.** Given the small dataset presented, the network is kept relatively small at 19 units. Using too many units results in faster overfitting, while too few units results in too much generalization. The number of units is not to be mistaken for the number of neurons in the network. Units refer to the size of the cell state; in this case a vector of 19. Not a layer of 19 memory blocks. The one memory block produces 19 outputs, which is the reason for also adding 1 dedicated output neuron. The output of the network is only compared to one value of GDP growth.

- **Training for 150 epochs.** The input- output pairs are presented to the network 150 times, adjusting weights each time. This provides enough weight adjustment to capture patterns, without overfitting.

- **A Rectified Linear Unit (ReLU) activation function.** As shown in equation (2.16) the LSTM needs a linear constant activation function. Referring to Figure 4.1 and the form $f(x) = max(0, x)$, ReLU fills this requirement as a piece-wise linear function. It is shown in (Nair & Hinton, 2010) to speed up computation in one type of recurrent network, as it passes scaled output for any $x > 0$, while also avoiding vanishing gradients by setting negative values to 0. One effect of this is, however, that it renders negative cells useless by locking them at 0, essentially "killing" parts of the network. Contrary to a reasonably assumed negative effect of "dead" cells, (Glorot et al., 2011) finds that zeroes in the gradient can help minimum convergence.

**Figure 4.1:** Rectified Linear Unit Activation function $[0, x]$




*Source: (Nair & Hinton, 2010)*

- **A mean squared error loss function.** The minimization of fitting error at each epoch is calculated as mean squared error. This is the error defined in equation (2.3), with $e_j(t) = y_{trg}(t) - y_j(t)$ defined in equation (2.2):

$$\epsilon(t) = \frac{1}{2} \sum_{j=C} e_j^2(t)$$

- **Stochastic Gradient Descent (SGD) Optimizer.** The optimization outlined in section 2.4.3 is the Stochastic Gradient Descent. Other optimization techniques exists like the ADAM presented in (Kingma & Ba, 2015). The method is different from SGD and provides better optimization for large datasets, because of an adaptive learning rate. Drawing parallels to SGD, the learning rate, $\eta$, is constant. The danish data presented is small compared to time series usually fed to the network, so SGD is chosen as it is slower to overfit.

## 4.3 Benchmark Factor Model

The benchmark factor model is estimated and selected by cross-validation to include 5 common components. This is evident in the plot of the cross-validation error Figure 4.2, and Table 4.3: As it can be seen in Figure 4.2 the error "dips" three times; at 5 components, 11 components and 18 components. The grey line indicates the lowest error using 5 components. The error of the three dips are shown in Table 4.3 to confirm that 5 principal components provides the lowest error of the

**Figure 4.2:** Cross-Validation Error for No. of components

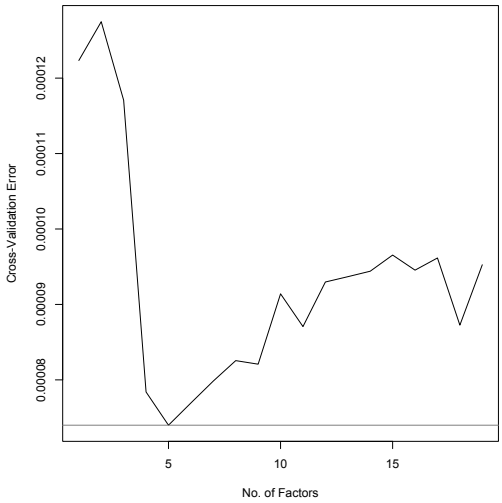



**Figure 4.3:** Cross-Validation Error for No. of Components

| No. of components | Error |
|---|---|
| **5** | **7.292008e-05** |
| 11 | 7.571166e-05 |
| 18 | 7.513744e-05 |

*Bold: The lowest (best) error*

cross-validation. It thereby gives the DFM with the best performance. The distribution of the residuals is show in Figure 4.4. With residuals normally distributed, the estimated model is reliable. The machine learning approach of out-of-sample prediction is done by estimating the coefficients on the training split of the data. It is then fed the out-of-sample inputs which produce an output that is compared to the target output from the out-of-sample data.

## 4.4 Results

The models ability to predict is compared with a squared error, root mean squared error (RMSE) and a mean average error (MAE) of their predictions. RMSE and MAE are common choices when comparing predictions. These methods are, however, prone to scale differences (Hyndman & Koehler, 2005). With the two models fed the same dataset, this problem does not arise.
The prediction of the LSTM is presented in Figure 4.5 and the prediction of the Principal Components Regression in 4.6.
The blue part represents the training fit of the models; the growth in log-transformed GDP predicted by the network. The grey is the actual growth in log-transformed GDP from the data. The vertical grey line is the training- test split of the data set. The red part is thus the log-GDP growth predicted from the out-of-sample test inputs. Both models are able to catch some general movements in both training- and out-of-sample data. Figure 4.7 show both models for comparison. The LSTM

**Figure 4.4:** Distribution of DFM Training Fit Residuals

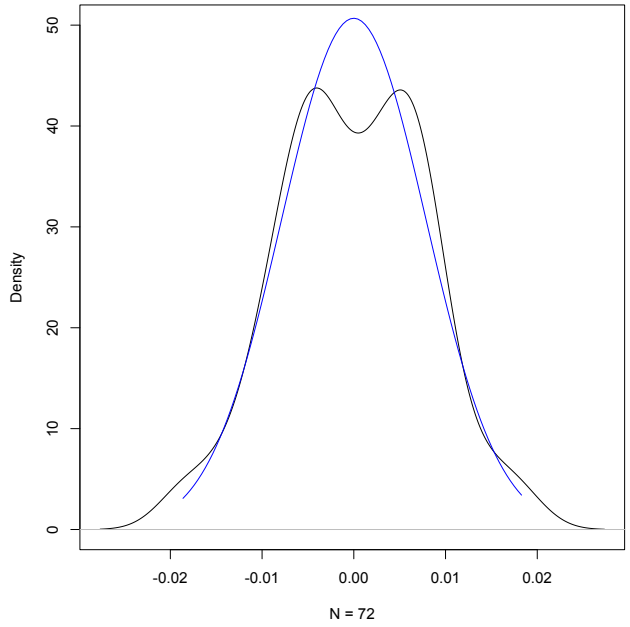
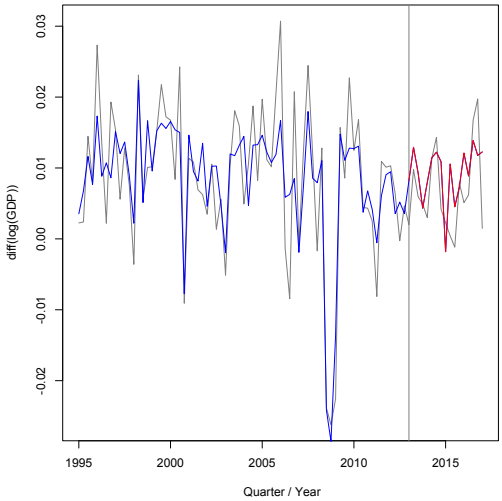
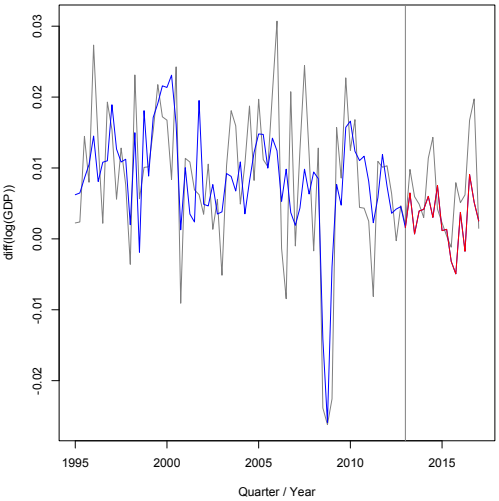



**Figure 4.5:** Fit of LSTM: 19 & 1 units, 150 Epochs, ReLU, MSE, SGD




*Blue: Fit to training inputs*
*Red: Prediction on Out-of-Sample inputs*

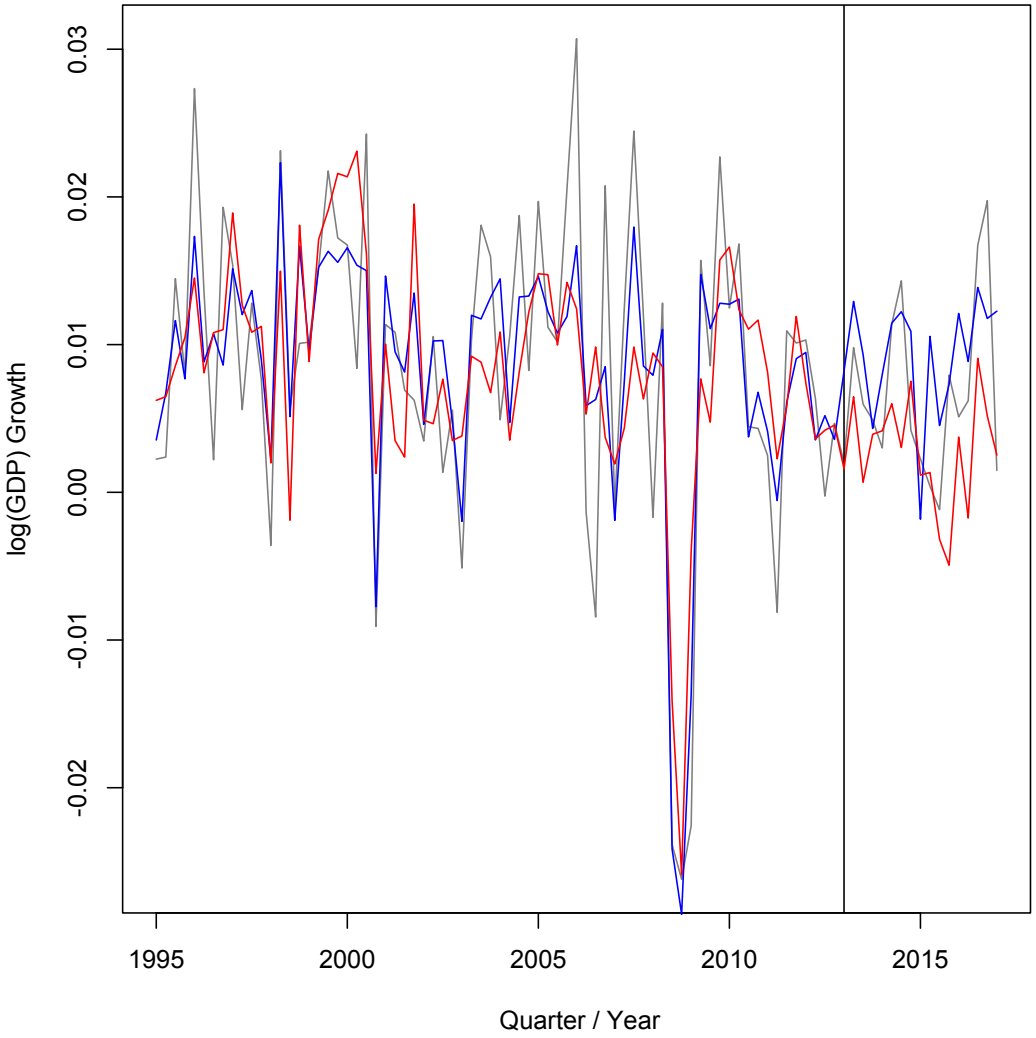
**Figure 4.6:** Fit of Dynamic Factor Model with Principal Components




*Blue: Fit to training inputs*
*Red: Prediction on Out-of-Sample inputs*

is *blue* and the DFM is *red*.

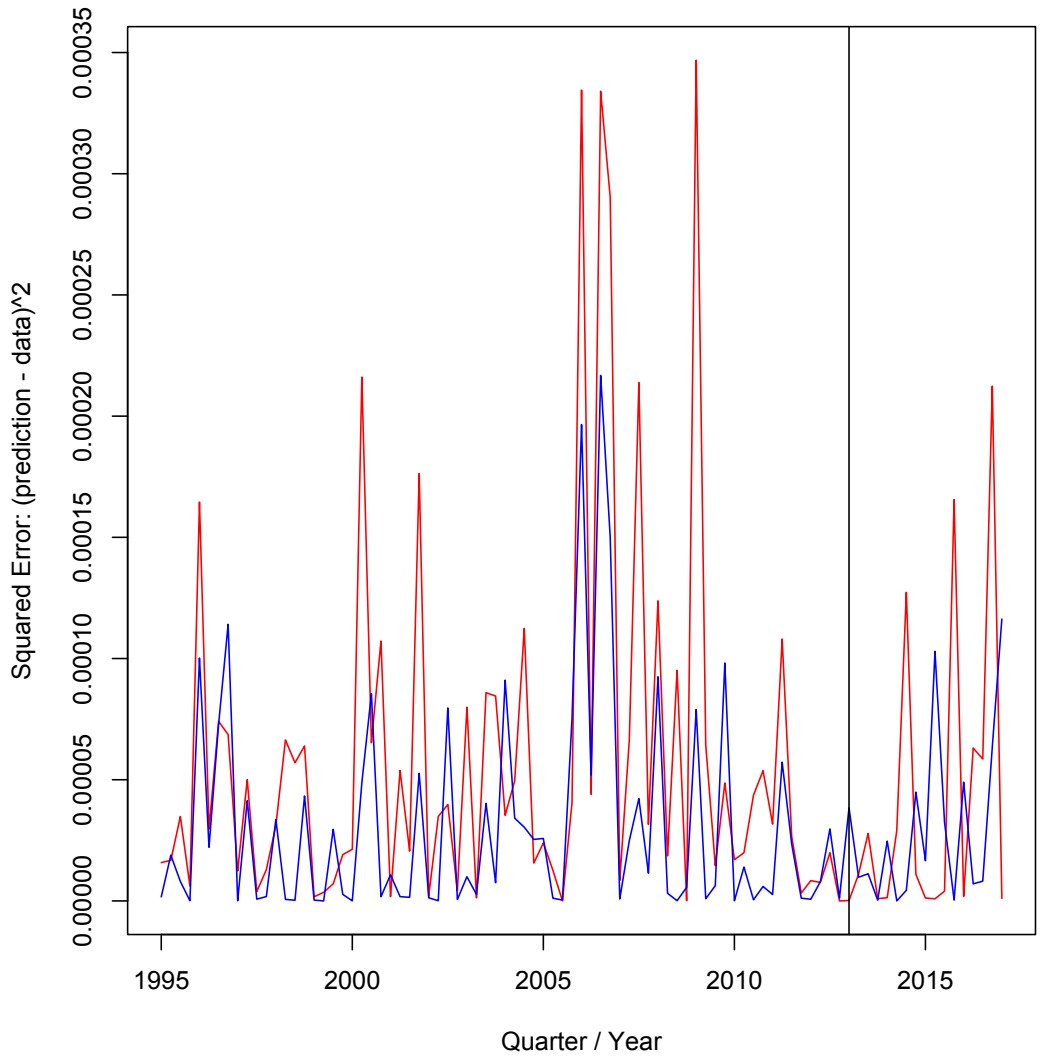
**Figure 4.7:** Fit of LSTM & DFM to GDP Growth




*Green: LSTM from Figure 4.5*
*Red: DFM from Figure 4.6*

The squared error of both models is shown in Figure 4.8. The squared error is a simple measure for the error at each time step, squared to imply distance between the actual and predicted value. This allows tracking of the performance for each model. The scale of the errors are very low. This is partly due to the scale of the underlying data being contained in the interval of $[0.03 : -0.03]$.
It can be seen that the LSTM (blue) generally shows lower error spikes. Also in the out-of-sample prediction. Comparing this with Figure 4.7 the spikes happen at rel-

**Figure 4.8:** Squared Error between model output and data values




*Green: LSTM from Figure 4.5*
*Red: DFM from Figure 4.6*

atively volatile changes in GDP, except at the crisis happening in 2008. This could be an indication of missing indicators with explanatory power, but also a case of reasonable generalization; the models are able to capture underlying patterns in the data, without overfitting.
The RMSE and MAE for both models at training fit, out-of-sample prediction and total fit is presented in Table 4.3. While squared error provides comparability, the spikes happen at different time steps making a general conclusion of performance difficult; Is it more important to get the direction or level correct? Take for example the (negative) spike around 2015 in Figure 4.7. The DFM error in the small dip after 2015 in Figure 4.7 is seen to gradually become smaller in Figure 4.8, leading up to the two predictions being relatively close when the DFM reaches a top, while the data reaches a bottom of a dip. The error is small, but the direction is opposite. Not to conclude that the growth becomes negative, just slower than predicted. RMSE and MAE are averaging techniques giving a general accuracy of the model. Looking at the highlighted out-of-sample measure the LSTM performs better than

**Table 4.3:** Sectional RMSE & MAE for Both Models

| **Model** | **RMSE** | **MAE** |
|---|---|---|
| **LSTM** | 0.005584161 | 0.004289430 |
| - Training | 0.005584284 | 0.004207058 |
| - Out-of-sample | **0.005583643** | **0.004638299** |
| **DFM** | 0.007582204 | 0.006169701 |
| - Training | 0.007816738 | 0.006506790 |
| - Out-of-sample | 0.006495667 | 0.004742028 |

*Bold: The lowest (best) metric*

the DFM in out-of-sample predictions for this specific data. The accuracy of the fit on training data and general fit is somewhat useless in direct comparison, as these can be improved to near perfection by overfitting as much as possible. It does however suggest that more available data will increase the fit of the training data better and provide increased accuracy in out-of-sample prediction. Much like the example of the univariate example in section 1, showing good fitment in both training and out-of-sample prediction.
The MAE and RMSE error difference between the DFM and LSTM is relatively small. The Diebold-Mariano forecasting accuracy test can be used on the residuals of both series, to test for significance in predictive accuracy. A two sided test is used on the null hypothesis that one prediction is better than the other. The result is rejection of the hypothesis with a p-value $> 0.001$, meaning that predictions are equally "good". With the LSTM showing a small improvement in predictive accuracy in this specific setup, this thesis suggest that the LSTM has the ability to perform similar to a principal component factor model.

# Chapter 5

# The Machine Learning Contribution to Economics

This thesis shows that the LSTM can be applied in a more traditional multivariate economic context. It is able to map the values of input series to GDP growth, and successfully recognize some patterns in the underlying series. The stochastic gradient descent applied in the LSTM is, by nature, a greedy search using lots of observations to find the minimum error in the gradient. The results of the analysis also shows that the LSTM in this scenario is able to produce a meaningful and accurate prediction, when the dataset is wide containing relatively few observations. This makes the LSTM useful as a tool also being able to predict in a macroeconomic context, even with relatively few observation. The automatic nature of the LSTM indicates that econometric understanding is less important, and that only economic knowledge is needed for selecting indicators. A prior selection of indicators is done in this thesis to test the predictive power of the LSTM, not focusing on selection of indicators. This is also why the Principal Component Regression method is used, instead of a component analysis factor model with hundreds of series.

## 5.1 Downsides of the Neural Network

As mentioned in section 2.2 the machine learning approach is predictive by nature, and limits the possibilities of more traditional structural inference. This may hurt the networks direct ability to forecast n-steps ahead, even in simple linear extrusions; there are no parameters to extrude. While predictions from available data can be fitted and predicted easily, any output from the model has to be based on input data. While this is a major downside, it still leaves some possibilities for forecasting:

- For univariate prediction the process is simply as presented in chapter 1. Creating a sliding window that produces a step-ahead forecast. This can then be extended by including the forecast as the last observation in the window. It does however require a high number of observations.
- For multivariate applications it would be possible to forecast each series n-steps ahead, which is then used as input for the LSTM to predict output. This can quickly become cumbersome and unreliable when dealing with a high number of input series, as each series has to be forecasted. Furthermore this implies an assumption that all underlying series are unrelated, and does not influence each other. In a macroeconomic perspective, this is highly unlikely.
- To incorporate interdependencies among the series the network could be shaped to produce an output for each series. A sliding window with a given number of observations and the total number of series, can then be fed to the network, producing the next time step for each series as output. Feeding all series at the same time allows the network to capture influence among variables.
- A hybrid of LSTM and VAR models could be created. An underlying system of vector autoregressive models, each consisting of relating macroeconomic indicators, provides forecasts of the time series. The forecast of these series are then used as input to the LSTM. This also makes error correction and impulse-response analysis possible. This is, however, essentially a factor model with an LSTM on top. Such a configuration does not necessarily provide any real benefit over the pure principal component model.

## 5.2 Posibilities of the Neural Network

### 5.2.1 Nowcasting

A natural extension of this thesis would be a nowcast of an economy. The result of this thesis does, however, not directly imply that such an application would be successful. The LSTM requires series of equal dimensions, so basic methodology includes aggregation or interpolation of series, to match each other. Constructed monthly GDP growth can then be fed to the LSTM mapping it to monthly inputs. The hybrid third generation models with principal components and a kalman filter handles the mix of frequencies natively and estimates loadings for each series. This is why it is used in the nowcasting literature in section 4. Because of this it is not possible to suggest that the LSTM would perform on par with the DFM and Kalman Filter, as this thesis compares the network to a principal component factor model.

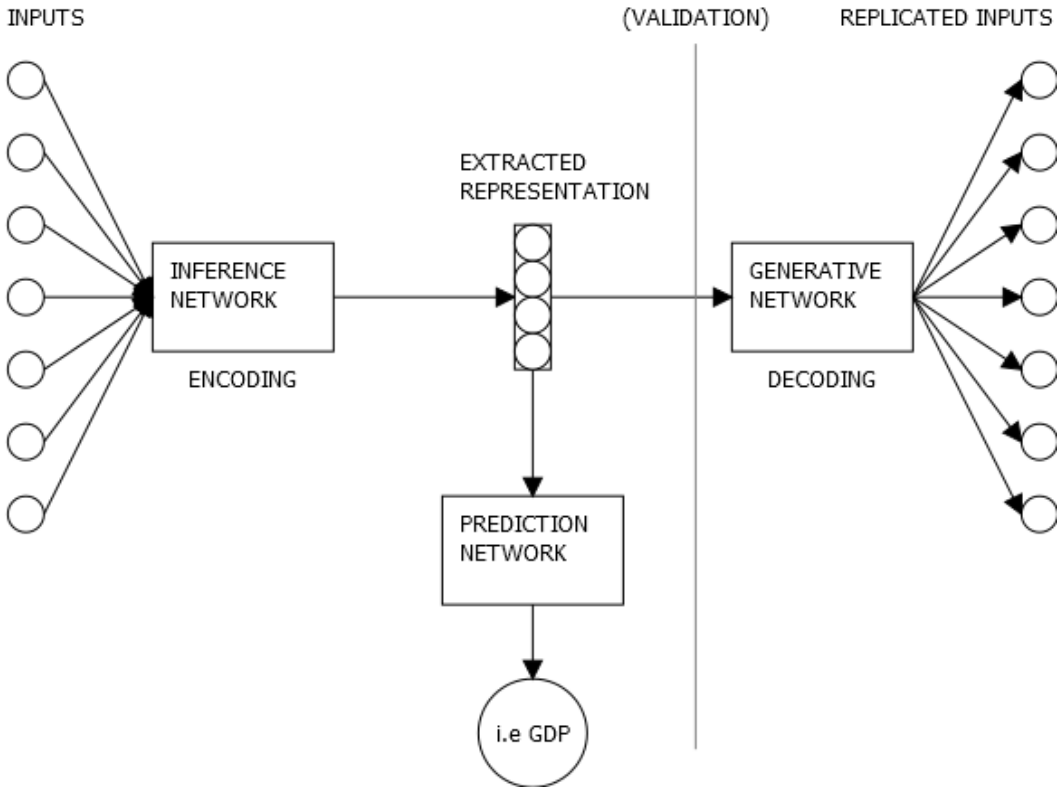
### 5.2.2 Local Interpretable Model-Agnostic Explanations (LIME)

While the algorithm behind outputs in the LSTM is known, real-time states of the network is hidden. The reasoning of why a specific output is produced, is unknown. This requires a certain amount of trust and raises an important question: how much should you trust a prediction? When no evidence or reasoning is given, it is tough to argument as to why the prediction is useful.
The LIME technique presented in (Ribeiro et al., 2016) attempts to open up these hidden states of the network, and explain the choices made when predicting. It is, however, developed and tested on classification models, as these classify inputs on probabilities. The idea behind the method is that parts of the input is removed, and the model then has to predict based on the restricted inputs. The percentage of classification correctness for each component can then be extracted. This provide local prediction interpretation for each component of input, which combined, can help assessing the trustworthiness of the overall model prediction.
The method is model-agnostic and should be applicable to regression models. It is thereby possible to provide necessary information to determine the impact of each regressor. In extend, this could in theory open for the possibility of structural inference in a macroeconomic regression context. The powers of such a framework is unknown and requires further development, but it provides an exciting direction in which the economic relevancy of a regressive neural network could increase.
By modifying the problem in this thesis, it is possible to directly apply the LIME method to a context much like the one presented. By replacing GDP growth with a recession indicator for the danish economy, replacing the ReLU activation function with a softmax classifier function and setting the loss function to cross-entropy, the network should be able to solve classification problems. Such an application requires further research to conclude its successfulness, but if the results are satisfactory it would then be possible to apply the LIME method to open up the network, and in theory give an indication of how the indicators contribute to recession.

### 5.2.3 Variational Auto-Encoders (VAE)

The variational auto-encoder is a newer more complex method of using the LSTM. As the name suggest these methods are essentially encoding information into more dense representations, much like the encoding and compression of files into different formats. An example is the file compression and encoding of .mp3 and .jpg. The basic idea is shown in Figure 5.1. It uses an LSTM in a inference layer to map information in input to a more condensed format, seeking to minimize dimensionality while keeping as much relevant information as possible. In a sense discarding as much as possible without loosing recreational power. This draws parallels to the principal component method of estimating latent factors, with the factors being the condensed information. This information is, like the factors in

**Figure 5.1:** Variational Auto-Encoder




*VAE source: Figure 1 (Kingma & Welling, 2014). Economic application is added*

the principal component regression, a linear combination of inputs.
The framework is validated be decoding the condensed information back into a representation as close as possible to the input data. This is done by another LSTM in the generative layer, based on the latent distribution in the hidden layer. The economic application is thus relevant with a problem much like the one in this thesis; a scenario with many indicators used to predict one variable, like GDP as shown in Figure 5.1. The product is a structural system of combined factors representing the essence of the data, but inference on individual series is still not directly possible.

## 5.3 Data Generation

As mentioned in the introduction in section 1, two philosophies of implementation exists. While this thesis has pitted a machine learning method against an econometric method, the coexistence of both methods aiding each other is still relevant. A discussion of data generation would however imply discussion of ways to extract more information from data, which is not necessarily economic of nature. The VAE and LIME is methods of extracting more information, but only gains economic relevance when used to solve an econometric problem.
The supply of more and new kinds of data is presumably useful, as could be seen in (Henderson et al., 2012) and (Lobell, 2013). Especially in a field like spatial economics where geographic data is used.
But, when doing an asymptotic approximation such as the one performed in the factor model, it does however imply that the effect of new data at some point becomes insignificant with traditional econometric methods. So while this combination of machine data extraction and econometric interpretation in theory is useful, it could at some point loose its effectiveness. The machine learning methods on the other hand only improves with more available data. But until more transparency is introduced in the predictions, their trustworthiness should be evaluated based on the specific context it is used in.

### 5.3.1 Overconfidence in data

While this thesis is data driven, and seeks to explore a new method of "letting the data speak", it is also necessary to step back and reflect on the use of data in the economic field. Computer- and natural sciences have certain laws and a degree of predictability, which makes the LSTM useful in such a framework. It makes sense to squeeze out an extra percent of accuracy. The predictability is somewhat more diffuse in economics. The modelling of rational agents provides a set of pseudo-natural laws of behaviour relying on a utility function, but the function is ultimately individual and not necessarily constant over time. Using data to predict and forecast is thus somewhat equal to the statement that previous behaviour is

equal to future behaviour; *ex-post* is equal to *ex-ante*. A keynesian argument would be that this is not necessarily true, as no situation is ever the exact same. The experience gathered from a given situation would influence the thought process of the same agent. If a future identical situation would arise, experience would make the situation different. The LSTM is no exception to this assumption of equality, as the overfitting scenario draws parallels to this idea. The network can only learn patterns it has seen before. If the network learns previous patterns too well, predictive power starts to deteriorate. Even though predictions are useful in reducing uncertainty in real-time applications like nowcasting, the results of any model should still be considered a guideline, no matter the amount of "intelligence" imitated. Especially if no insight into reasoning behind the prediction is presented.

# Chapter 6

# Conclusion

The LSTM has gained popularity in machine learning because off its ability to incorporate long-term dependencies. It is able to somewhat automatically extract information from data, and recognize underlying patterns. Economic applications of this type of network is relatively limited in the literature.
It has shown good prediction performance on univariate time series, and is therefore often used in financial scenarios.

The LSTM solves some of the problems with the earlier gradient descent method, used in the feed-forward- and recurrent backpropagation algorithms. While feed-forward networks only handles the current time step, recurrent networks remembers only short-term dependencies before gradients vanish. This is due to exponential dependencies on weights, making the effect of old values miniscule with time. By incorporating a forget gate in the LSTM, the network is able to remember information presented, and forget it once it becomes irrelevant. This is achieved by gating information into a memory cell, allowing the neuron to save specific information for later. It is thereby possible for the LSTM to handle multivariate macroeconomic setups, as the LSTM is able to effectively learn patterns in a dataset, all on its own.
The LSTM is fed 19 macroeconomic indicators for the danish economy to predict growth in GDP. Each series is quarterly data consisting of 92 observations. The train- test-split for out-of-sample validation is done so the training is performed on 72 observations. The network is kept relatively small, containing a single layer of 19 units and an output layer of 1 unit. A rectified linear unit activation function is used and a mean-squared-error loss is defined in conjunction with the stochastic gradient descent optimization. Benchmarking is done against a principal component factor model. Five latent factors are constructed from a linear combination of indicators, and then regressed on GDP growth. Multicollinearity is avoided by this approach, but structural inference about individual series is not possible. Weak

serial correlation is allowed as the model follows an approximation approach making results asymptotically consistent.
In out-of-sample prediction the LSTM shows lower RMSE and MAE than the factor model, thus providing a better prediction in this specific scenario. The LSTM produces smaller errors that occurs more often, while the factor model produces fewer, but larger spikes. A Diebold-Mariano test on forecasting accuracy suggest, however, that no model performs significantly better than the other. The conclusion is then, that the LSTM in the scenario presented, performs similar to the principal component regression.
This results places the LSTM as a predictive tool in an economic toolbox. Despite the underlying greedy search algorithm, the LSTM shows that it is able to handle macroeconomic data, which usually contains a lot of variables with few observations in each series.
One downside to this method, in the macroeconomic perspective, is the lack of structural estimation. It limits the possibilities of directly implementing long-run relationships and error correction into the network. It also makes impulse-response policy analysis limited. If such terms are to be used in the prediction, prior econometric modelling is necessary to produce the estimates. Forecasting then becomes possible, as each series can be extended. Either by econometrics with ARIMA models or VAR systems, or by machine learning with a sliding window step-ahead forecast.
Compared to the econometric method of solving the economic problem of this thesis, the LSTM shows no direct advantages over the principal component regression.

The methodology of mapping sequences of input to output does have some immediate strength. The LSTM is able to automatically generate new kinds of data to be used in an economic context. Take for example news headlines. They can be broken down into sequences of words and classified by the network in different categories depending on attitude towards the subject. It can then be used in a sentiment analysis, in combination with traditional quantitative inputs.
Another example is how an image can be broken down into sequences of pixel values, thereby allowing the network to classify what the image is showing. Prediction of for example crop yield or block-development in areas with poor- or missing data can be done with satellite images.
The emergence of new types of data does then, seem promising when machine learning an econometrics is combined, instead of pitted against each other.
The capabilities of new methods in machine learning seems promising in bringing more economically relevant features to the networks.
The Variational Auto-Encoder provides a machine learning translation of the factor framework, and the Local Interpretable Model-agnostic Explanation (LIME) framework is a step in the direction of macroeconomic structural inference being

possible, by providing transparency.

So, while the LSTM can handle different types of data in the computer science literature, this thesis has shown that the LSTM also have some predictive power in a macroeconomic analysis. Some transparency is possible in classification scenarios with the LIME method, *but until more transparency is consistently introduced to the regression problem, the trustworthiness of the predictions made in macroeconomic applications should be evaluated based on their context. This places the LSTM as a powerful predictive tool in the economic toolbox, but most useful when combined with other more transparent methods allowing for interpretation.*

# References


Alvarez, R., Camacho, M., & Perez-Quiros, G. (2016). Aggregate versus disaggregate information in dynamic factor models. *International Journal of Forecasting*.

Antweiler, W., & Frank, M. Z. (2004). Is all that talk just noise? the information content of internet stock message boards. *The Journal Of Finance*.

Atkinson, R. C., & Shiffrin, R. M. (1968). Human memory: A proposed system and its control processes. ..

Ayodele, T. O. (2010a). New advances in machine learning. In Y. Zhang (Ed.), (p. 9-18). InTech.

Ayodele, T. O. (2010b). New advances in machine learning. In Y. Zhang (Ed.), (p. 19-48). InTech.

Baddeley, A. D., & Hitch, G. (1974). Working memory. *Psychology of Learning and Motivation*, *8*, 47-89.

Banerjee, A., Marcellino, M., & Masten, I. (2017). Structural fecm: Cointegration in large-scale structural favar models. *Journal of Applied Econometrics*.

Barhoumi, K., Darné, O., & Ferrara, L. (2009). Are disaggregate data useful for factor analysis in forecasting french gdp. *Document De Travail No. 232*.

Bliss, T. V. P., & Collingridge, G. L. (1993). A synaptic model of memory: long-term potentiation in the hippocampus. ..

Bragoli, D., Metelli, L., & Modugno, M. (2014). The importance of updating: Evidence from a brazilian nowcasting model. *Finance and Economics Discussion Series Federal Reserve Board*.

Chalmers, D. J. (1993). A computational foundation for the study of cognnition. .. Retrieved from `http://consc.net/papers/computation.html`

Chernis, T., & Sekkel, R. (2016). A dynamic factor model for nowcasting canadian gdp growth. *Empir Econ (2017)*.

D'Agostino, A., McQuinn, K., & O'Brien, D. (2011). Nowcasting irish gdp. *MPRA no. 32941*.

Dahlhaus, T., Guénette, J.-D., & Vasishtha, G. (2015). Nowcasting bric+m in real time. *Bacnk of Canada Working Papers 2015-38*.

Desai, R. (2017). Artificial intelligence (ai). .. Retrieved from `http://drrajivdesaimd.com/2017/03/23/artificial-intelligence-ai/`

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179-221.

Enders, W. (2015). *Applied econometric time series*. Wiley.

Eysenck, M. W., & Keane, M. (2005). *Cognitive psychology - a student's handbook*. Psychology Press Ltd.

Fischer, T., & Krauss, C. (2017). Deep learning with long short-term memory networks for financial market predictions. *FAU Discussion Papers in Economics, No. 11/2017*.

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with lstm. *Technical Report IDSIA-01-99*.

Giovannelli, A., & Proietti, T. (2014). On the selection of common factors for macroeconomic forecasting. *CREATES Research Papers 2014-46*.

Glorot, X., & Bengio, Y. (2010). Understanding the diffuclty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, *9*.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*.

Grossberg, S. (1980). How does a brain build a cognitive code? *Psychological Review*.

Hagan, M. T., Demuth, H. B., Beale, M. H., & Jesús, O. D. (2014). *Neural network design* (2nd ed.). Martin Hagan.

Hall, J. S. (2011). Algorithmic probability and friends. bayesian prediction and artificial intelligence. In D. L. Dowe (Ed.), (p. 174 - 183). Springer.

Haykin, S. (1999). *Neural networks - a comprehensive foundation* (2nd ed.). Prentice-Hall.

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. *Neural Networks for Perception*, *2*, 65-93.

Henderson, J. V., Storeygard, A., & Weil, D. N. (2012). Measuring economic growth from outer space. *American Economic Review*.

Herbrich, R., Keilbach, M., Graepel, T., Bollmann-Sdorra, P., & Obermayer, K. (1999). Neural networks in economics.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation 9 (8)*.

Hyndman, R. J., & Koehler, A. B. (2005). Another look at measures of forecast accuracy. ..

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastich optimization. *ICLR 2015*.

Kingma, D. P., & Welling, M. (2014). Auto-encoding variational bayes.

Kuang, Y., Singh, R., Singh, S., & Singh, P. (2017). A novel macroeconomic forecasting model based on revised multimedia assisted bp neural network model and ant colony algorithm. *Multimedia Tools and Applications*.

Lobell, D. B. (2013). The use of satellite data for crop yield gap analysis. *Field Crops Research*.

Luciani, M., Pundit, M., Ramayandi, A., & Veronese, G. (2015). Nowcasting indonesia.

Luciani, M., & Ricci, L. (2014). Nowcasting norway. *International Journal of Central Banking*.

Makridaki, S. (n.d.). *M4-competition.* Retrieved from `https://www.m4.unic.ac.cy/`

McCulloch, W. S., & Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115-133.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Mullainathan, S., & Spiess, J. (2017). Machine learning: An applied econometric approach. *Journal of Economic Perspectives*, *31*(2), 87-106.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restriced boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning*.

Nationalbank, D. (2017). Outlook for the danish economy - no. 16.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "why should i trust you?" explaining the predictions of any classifier.

Rumelhart, D. E., McClelland, J. L., & Group, P. R. (1986). *Parallel distributed processing - explorations in the microstructure of cognition* (J. A. Feldman, P. J. Hayes, & D. E. Rumelhart, Eds.). The MIT Press.

Rusnák, M. (2013). Nowcasting czech gdp in real time. *CNB Working Paper 6/2013*.

Solomonoff, R. J. (2009). Machine learning - past and future. ..

Stock, J. H., & Watson, M. W. (2002). Forecasting using principal components from a large number of predictors. *Journal of the American Statistical Association*.

Stock, J. H., & Watson, M. W. (2010). Dynamic factor models.

Varian, H. R. (2014). Big data: Net tricks for econometrics. *Journal of Economic Perspective 28,2*.

Zhang, D., Yu, L., Wang, S., & Xie, H. (2010). Neural network methods for forecasting turning points in economic time series: an asymmetric verification to business cycles. *Frontiers of computer Science China 4(2)*.

# Appendix A

# R code

The R code related to creation and computation of the models is shown. Data loading, plotting, results etc. is not.
The code is divided in different sections:

1. The day-series LSTM
2. Initialization and sorting of macroeconomic data. Declaration of variables.
3. The macroeconomic LSTM
4. The macroeconomic principal component regression

## A.1 LSTM for forecasting competition analysis

A sliding window approach is used

```
# Choosing 4 series with 1000 observations
ex_series <- (full_day_dat[782:786,2:1000])
day_dat <- as.data.frame(t(ex_series[1:4,]))

###### Preprocessing Data

# Train/Test splitting - 80/20
day_ind <- 800
day_train_dat <- day_dat[1:day_ind,]
day_test_dat <- day_dat[(day_ind+1):1000,]

# Scaling training data. Scaling test data with training scale
dtr_sc <- scale(day_train_dat, center = T, scale = T)
dtr_b <- attr(dtr_sc, "scaled:scale")
dtr_a <- attr(dtr_sc, "scaled:center")
dte_sc <- scale(day_dat, center = dtr_a, scale = dtr_b)
```

```
17
18 # Splitting into sequence. d_series is change manually for each
       series
19 d_series <- 1
20 dw_s <- 1
21 dw_e <- 50
22
23 ## Train data
24 # Creating a matrix with 750 sequences containing 50 observations.
25 dtr_df_in <- matrix(0, ncol = 750, nrow = 50)
26 dtr_df_in
27
28 # Placing the first 50 observations in one row, then moving to
       observation [2:51], [3:52] and so on.
29 for(dtr_c in (1:750)){
30 dtr_df_in[,dtr_c] <- dtr_sc[dtr_c:(dtr_c+49),d_series]
31 }
32 dtr_df_in <- t(dtr_df_in)
33
34 # The output is defined as the next observation after the sliding
       window
35 dtr_df_out <- dtr_sc[51:800,d_series]
36
37 ## Test data
38 # same procedure
39 dte_df_in <- matrix(0,ncol = 200, nrow = 50)
40
41 for(dte_c in (1:200)){
42   dte_df_in[,dte_c] <- dte_sc[(dte_c+750):(dte_c+799),d_series]
43 }
44 dte_df_in <- t(dte_df_in)
45
46 ###### Shaping input for LSTM
47
48 # Create 3-Dimensional tensor for both training and test set.
       Output data is not changed, only renamed to day_tr_y
49 day_tr_x <- k_expand_dims(dtr_df_in, axis = 2)
50 day_te_x <- k_expand_dims(dte_df_in, axis = 2)
51
52 # Shaping input as (no. samples, no. time steps, no. features)
53 d_samples <- dim(day_tr_x)[1]
54 d_steps <- dim(day_tr_x)[2]
55 d_feats <- dim(day_tr_x)[3]
56
57 # Converting tensor to a format useable by keras
58 dtr_x <- k_eval(day_tr_x)
59 dte_x <- k_eval(day_te_x)
```

```
60
61 ####### Building the network with keras
62
63 day_mod <- keras_model_sequential()
64 layer_lstm(day_mod,units = 50, activation = "relu", input_shape =
       c(d_steps,d_feats),recurrent_dropout = 0.4, trainable = T,
       return_sequence = F)
65 layer_dense(day_mod, units = 1)
66 compile(day_mod, loss = "mse", optimizer = "adam")
67
68
69 # Training the network with training data
70 fit <- fit(day_mod,dtr_x, day_tr_y, batch_size = 12, epochs = 40,
       verbose = 1)
71
72 # Predicting the out-of-sample outputs from out-of-sample inputs
73 day_pred_t <-predict_on_batch(day_mod, dte_x)
```

## A.2 Initialization for macroeconomic analysis

```
1 library(keras)
2 library(tensorflow)
3 library(Metrics)
4 library(plsdof)
5
6 # setting seed to ensure consistency between models and
      reproducibility
7
8 set.seed(1000)
9
10 # converting data to a time series object
11 class_dat <- ts(raw_dat[1:90,1:21], start = c(1995,1), frequency =
       4)
12
13 # Log-transforming the data and taking first differences
14 dat <- cbind(
15       diff(log(class_dat[,3])), # GDP
16             class_dat[i:1,4], #cpi_gr
17             log(class_dat[2:90,5:6]), #emp, uemp
18             class_dat[2:90,7:9], #int_3m, int_10y, diff(share_
      yield)
19             log(class_dat[2:90,10:11]), #oil , ex_USD
20             class_dat[2:90,12], #permits
21             diff(log(class_dat[,13])), #bus_cred
22             diff(log(class_dat[,14])), #house
```

```
23              (class_dat[2:90,15:18]), #man_com, eu_com, g_con, hh_
    con
24              diff(log(class_dat[,19])), #xp
25              diff(log(class_dat[,20])), #imp
26              diff(log(class_dat[,21])) #ca
27 )
28
29
30 # Creating global constants for splitting data and setting the
    number of indicators
31 k = 19
32 ind <- 72
33
34 t_len <- length(dat[,1])
35 w_te <- c((ind+1):t_len)
36 w_tr <- 1:ind
```

## A.3 LSTM for macroeconomic analysis

Each input is one indicator. Output is GDP growth

```
1 # Splitting scaled data with windows created earlier
2 train_in <- (dat_in[w_tr,1:(k-1)])
3 train_out <- t(t(dat_out[w_tr]))
4 test_in <- (dat_in[w_te,1:(k-1)])
5 test_out <- t(t(dat_out[w_te]))
6
7 # Creating 3-Dimensional tensors from data
8 x_train <- k_expand_dims(train_in, axis = 2)
9 y_train <- k_expand_dims(train_out, axis = 2)
10 y_test <- k_expand_dims(test_out, axis = 2)
11 x_test <- k_expand_dims(test_in, axis = 2)
12
13 # Shaping input as (no. samples, no. time steps, no. features)
14 nsamples <- dim(x_train)[1]
15 nsteps <- dim(x_train)[2]
16 nfeats <- dim(x_train)[3]
17
18 # Converting tensors to keras data
19 x <- k_eval(x_train)
20 y <- k_eval(y_train)
21 x_t <- k_eval(x_test)
22 y_t <- k_eval(y_test)
23
24 ###### Building the LSTM for macroeconomic data
25 mod <- keras_model_sequential()
```

```
layer_lstm(mod,units = 19, activation = "relu", input_shape = c(
    nsteps,nfeats), trainable = T,return_sequence = F)
layer_dense(mod, units = 1)
compile(mod, loss = "mse", optimizer = "sgd")

# Feeding macroeconomic data to LSTM
fit <- fit(mod, x, train_out, batch_size = 12, epochs = 150,
    verbose = 1)

# Predicting LSTM training fit and out-of-sample GDP growth with
    out-of-sample inputs
pred <-predict_on_batch(mod, x)
pred_test <-predict_on_batch(mod, x_t)

# Descaling prediction back into diff(log()) for interpretation
b <- attr(dat_scale, "scaled:scale")
a <- attr(dat_scale, "scaled:center")
pred_tot <- c(pred,pred_test)
descale <- pred_tot*b[1]+a[1]
```

## A.4 Principal Component Regression

```
# Creating a copy of data
ts_dat <- dat

# Splitting data with windows defined earlier
ts_train_x <- ts_dat[w_tr,2:k]
ts_train_y <- ts_dat[w_tr,1]
ts_test_x <- ts_dat[w_te,2:k]
ts_test_y <- ts_dat[w_te,1]

# Cross-validated principal component regression. Regression
    coefficients are computed by compute.jackknife = T
pcr <- pcr.cv(ts_train_x, ts_train_y, compute.jackknife = T)

# Values from the regression is saved
pcr.int <- pcr$intercept
pcr.coef <- pcr$coefficients
pcr.beta <- c(pcr.int,pcr.coef)

# The model is constructed by multiplying the loading matrix onto
    the training set. Residual can be calculated
pcr.mod <- cbind(rep(1), ts_train_x) %*%pcr.beta
pcr.res <- ts_train_y - pcr.mod

```

```
22 # The fit is saved
23 dfm_fit <- as.vector(pcr.mod)
24
25 # The test inputs are then fed to the model and the fit and out-of
     -sample predictions are merged. dfm_ts is the one plotted later
     .
26 pcr.test <- cbind(rep(1), ts_test_x)%*%pcr.beta
27 pcr_ts <- ts(pcr.test, start = ts.ind, frequency = 4)
28 dfm_ts <- c(dfm_fit, pcr_ts)
29 dfm_ts <- ts(dfm_ts, start = c(1995,1), frequency = 4)
```