NHH

Norwegian School of Economics
Bergen, Fall, 2018

Forecasting Norwegian Inflation with Deep Neural Networks

The application and comparison of different feedforward architectures

Benjamin Aanes & Mathias Gullien

Supervisor: Walter Pohl

Master thesis, Economics and Business Administration,
Economics / Finance

NORWEGIAN SCHOOL OF ECONOMICS

This page intentionally left blank.

Preface

This thesis is written as a concluding part of our Master of Science in Economics and Business Administration at the Norwegian School of Economics (NHH). The thesis is written in conjunction with our majors in Finance and Economics.

Working with this thesis has been challenging and rewarding. We both have a keen interest in macroeconomics, econometrics and big data, and have enjoyed exploring the possible synergies between these fields. The most challenging part has been understanding and applying state-of-the-art deep neural network methodologies. Through this process, we have increased our insight into the field of neural networks for time-series applications.

We would like to thank our supervisor, Walter Pohl, who introduced us to the field of Machine Learning through his course on Big Data, and who nudged us towards the sub-field of Deep Learning. We would also like to thank Benjamin's sister, who has been of great help reading and giving feedback on our work.

Abstract

This thesis investigates the feasibility of applying deep neural networks to macroeconomic forecasting in the Norwegian economy. The thesis is intended for macroforecasters curious about the possibility of utilizing these approaches for macroforecasting. *Deep Neural Networks* is the most recent term coined for directed graphical models which act as universal nonlinear function approximators, optimized through back-propagation. Focusing on monthly Norwegian year on year consumer price inflation, we design three different network architectures, one representing the single hidden layer neural network ubiquitous in the literature and the remaining two representing recent developments in the field. We apply the modern and pragmatic approach to designing network architectures, applying time-series cross-validation to tune network hyperparameters in a problem specific setting, giving ample time to the construction of optimal networks. Each network architecture is trained repeatedly to produce repeat ensemble forecasts of inflation, and the predictive acuity of these ensembles are evaluated against common linear benchmarks. We find that both in the 2000 - 2009 period, used for network design, and in the 2010 - 2017 period, used for final evaluation, at least one of our neural network architectures outperforms the best included benchmark for short term horizons. The forecasting improvements are generally found in times of high volatility. Further, we find that the single hidden layer neural network is dominated by a deep multi-layer perceptron with residual connections. In the evaluation period, a deep convolutional neural network is the best overall forecast method, beating all benchmarks up to the six month horizon. At the three and six month horizons, the improvement over the best benchmark method is 18.2% and 10.6%, respectively. The convolutional neural network performs similar to the best benchmarks at longer horizons. While there exist barriers to the direct implementation of these networks in macroeconomic decision making, we argue, based on our results and recent literature, that including these methods in large statistical model suites, often applied by central banks, could indeed improve forecasting performance.

Contents

List of Abbreviations

Abbreviation	Explanation
AA	`auto.arima` approach
AR(p)	Autoregressive model of order p
ARAIC	ARIMA procedure selecting lowest AIC
ARBIC	ARIMA procedure selecting lowest BIC
AIC	Akaike Information Criterion
ARIMA	Autoregressive Integrated Moving Average
BIC	Bayesian Information Criterion
CNN	Convolutional Neural Network
CPI	Consumer Price Index
CPU	Central Processing Unit
GPU	Graphics Processing Unit
MAE	Mean Absolute Error
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
MA	Moving Average
NN	Neural Network
RMSE	Root Mean Squared Error
RW	Random Walk Model

List of Tables

List of Figures

1 Introduction

Forecasts of macroeconomic indicators, such as inflation, are paramount to decision making at both central banks and in the private sector. Since the influence of monetary policy on the economy works through transmission mechanisms with a substantial lag, policy makers must make decisions on how to handle inflationary or deflationary pressures ahead of time (Szafranek, 2017). Through the transparency revolution brought forth by the new Keynesian formulation of optimal policy, the forecasts themselves, as published in monetary reports, have become an essential part of central banks' modus operandi (Woodford, 2005). The forecasts of inflation also have a direct influence over decision making in the private sector, through the effects of monetary policy, and the expectations of market participants thereof, on the nominal value of long-term commitments (Faust & Wright, 2013).

Forecasting macroeconomic series is a severely difficult task, and because of the importance of accurate predictions of future movements in these series, a great deal of research has been conducted on the subject. Several different approaches have been developed in an attempt to aid decision making. Because of the complexity in the price dynamics of inflation in particular, a large part of the research has been focused on elastic modelling frameworks (Szafranek, 2017).

Deep Neural Networks is the most recent term coined for directed graphical models which act as universal nonlinear function approximators, optimized through back-propagation[1] (Wang, Ma, & Yang, 2014). The term succeeds *artificial neural networks* and *connectionism* as the marquee under which data-driven mathematical models, originally designed after human brain function, *learn* to predict from example. During the late 1990s - mid 2000s, several papers have focused on the feasibility of applying these network models in time series forecasting. Among these, a small fraction focused on forecasting monthly inflation and other macroeconomic variables (Hall & Cook, 2017; McAdam & McNelis, 2005; Moody, Levin, & Rehfuss, 1993; Nakamura, 2005; Stock & Watson, 1998). A majority of these papers has indicated that neural networks had either competitive or superior performance relative to univariate and multivariate benchmark approaches (Crone, Hibon, & Nikolopoulos, 2011). However, due to lacking methodological rigour, the forecasting community was unable to come to a conclusion about the models' predictive acuity (Gonzalez, 2000).

The advent of new and improved techniques has re-ignited the interest for forecasting the macroeconomy using neural networks, especially at central banks (Chakraborty & Joseph, 2017; Hall & Cook, 2017; Szafranek, 2017). Recently, Makridakis, Spiliotis, and Assimakopoulos (2018b) found that a single hidden layer neural network is generally outperformed by statistical methods such as the autoregressive integrated moving average framework. Based on these results, the authors argue that machine learning theorists need to improve their models in order for them

[1] For a thorough explanation of back-propagation, see review by Rumelhart, Hinton, and Williams (1986).

to become a viable forecasting tool.

While most previous research has focused on *single hidden layer neural networks*, Hall and Cook (2017) find that networks utilizing a deep architecture and different network structures outperform the Survey of Professional Forecasters[2] in predicting the movements of monthly US unemployment.

Enticed by this novel focus on deeper architectures and different network structures, this thesis investigates the relevancy of applying these new neural network methods in forecasting the Norwegian macroeconomy. Specifically, we focus on monthly year-on-year inflation in Norwegian consumer prices in the period January 2010 - December 2017. Through the design and evaluation of three different network structures, we investigate the predictive acuity of these methods in macroeconomic forecasting compared to common linear benchmarks, and whether we can improve upon the single hidden layer neural network often found in the literature.

The thesis is intended for macroeconomic forecasters curious about the possibility of utilizing these novel approaches to forecasting economic time series, and dealing with small data time series. Although the basis of our thesis is forecasting inflation through univariate autoregressive specifications, we attempt to design a generalizable and pragmatic model validation-test harness that is directly expandable to other time series, and that can take any number of features. This leads to the following research questions:

1. (a) To what extent is neural networks relevant in forecasting Norwegian macroeconomic indicators, such as inflation?
   (b) Given that data is scarce, are there possible architectural implementations which can improve performance over single hidden layer neural networks?
2. Are neural networks a feasible addition to the macroforecasting toolbox?

Our thesis contributes to the existing literature in several ways. Firstly, we follow Hall and Cook (2017) in focusing on deeper architectures and different network structures. These approaches has, to our knowledge, not been applied to either inflation forecasting or the Norwegian macroeconomy as a whole. We also compare these novel approaches to the single hidden layer neural network, found in the literature, in order to assess whether additional gains in forecasting accuracy can be achieved through extant, but somewhat neglected, architectural implementations. Additionally, we apply the modern and pragmatic approach to designing networks, as described in Chollet and Allaire (2018), applying time-series cross-validation to tune network hyper-parameters in a problem specific setting, giving ample time to the construction of optimal networks.

[2]The oldest quarterly survey of macroeconomic forecasts in the US. See https://www.philadelphiafed.org/research-and-data/real-time-center/survey-of-professional-forecasters/

The thesis is structured as follows. Chapter 2 outlines previous research in the field of time series forecasting with neural networks. Chapter 3 gives a brief introduction to the difference between data modelling and machine learning and an outline of our benchmark models, before delving into the theoretical foundations of neural networks. Chapter 4 details the gathering and preprocessing of our inflation data. Chapter 5 explains our particular evaluation procedure, as well as our approach to finding optimal benchmarks and how we design our neural networks. Chapter 6 reports the results of our forecasting comparison. Chapter 7 discusses the relevancy of neural networks for macroeconomic forecasting in light of our results, in addition thesis strengths and limitations and potential future avenues of research. Chapter 8 concludes the thesis.

2 Background

2.1 Strengths of Neural Networks

Zhang, Patuwo, and Hu (1998) outline four distinguishing features of neural networks for time-series forecasting. Firstly, neural networks are *data-driven* and *self-adaptive*, requiring few, if any, prior assumptions about the functional relationship in question. Secondly, neural networks *generalize*, they are able to learn from presented data and use this knowledge to predict on unseen data. Third, neural networks are *universal functional approximators*, able to approximate any continuous function to any desired accuracy. Fourth, neural networks are capable of modelling *nonlinear* relationships.

The combination of non-linear modelling, and data-driven and self-adaptive nature makes neural networks able to find nonlinear relationships without researchers having to manually implement these relationships based on hypotheses.

2.2 Neural Networks for Macroeconomic Forecasting

Several papers have focused on forecasting inflation and other macroeconomic variables using neural networks. Nakamura (2005) finds that a simple neural network outperforms autoregressive benchmarks in forecasting US inflation at the quarterly frequency. This is expanded by McAdam and McNelis (2005) who find that combining the forecasts of several neural networks with different specifications outperform autoregressive benchmarks for the US, German, Italian and Japanese consumer price inflation at longer time horizons.

Further work by Szafranek (2017) takes modelling ensembles to the extreme. The author creates a model ensemble consisting of 10 000 single hidden neural networks with slightly different

specifications, trained on bootstrap samples of inflation[3]. He finds that the model ensemble significantly outperforms a battery of machine learning and econometric models aimed at forecasting monthly Polish headline inflation. This model ensemble is found to be especially suited for forecasting inflation in small economies after 2011, where inflation has remained persistently lower than the official inflation targets set by central banks.

Szafranek (2017) argues that including such models in the forecasting toolbox would be beneficial to forecasting accuracy, especially in periods of structural change. Furthermore, he argues that the combination of several models, both linear and nonlinear, univariate and multivariate might improve forecast accuracy further. This final point is backed up by the results of Makridakis, Spiliotis, and Assimakopoulos (2018a) who recently concluded that the combination of statistical and machine learning methods was the best overall forecasting method for 100 000 diverse time-series at different frequencies.

Hall and Cook (2017), from the Federal Reserve Bank of Kansas City, apply four different *deep neural networks*[4] in forecasting US monthly unemployment. These advanced architectures have in common a deep network structure, applying more than a single hidden layer, and also applying residual connections between the layers. Using only past values of unemployment, and the series' first and second differences, all four network architectures outperform the Survey of Professional Forecasters (SPF) at one of the tested forecasting horizons. This indicates that, even though data may be scarce, a deeper neural network may be better at forecasting than the ubiquitous single hidden layer neural network.

A particular branch of research focuses on the performance of different methods as standard forecasting tools. In the competition environment, novel forecasting approaches purported to improve performance is compared on the task of forecasting a large and diverse set of different time series. Such competitions are surely important, and have led to strong insight into the general predictive ability of different forecasting methods. For instance, Makridakis and Hibon (2000) report four general results regarding time series forecasting which have become ubiquitous in the literature. Firstly, statistically complex forecast methods do not necessarily produce more accurate forecasts than simple ones. Secondly, the rankings of model performance vary according to the accuracy measure being used. Third, the accuracy of the combination of various methods outperform specific methods on average. Fourth, the performance of various methods depend upon the length of the forecasting horizon.

[3] *Bootstrap sampling* is a common re-sampling technique in machine learning used to artificially increase sample sizes and improve predictive accuracy (see Breiman (1996) for details).

[4] *Deep neural networks* are usually defined as neural network architectures with more than one hidden layer.

2.3 Weaknesses of Neural Networks

Gonzalez (2000) outline the main weaknesses of neural networks. The first is the lack of interpretability, known as the *black-box* problem. Through the inclusion of hidden intermediary functions, which sole purpose is to facilitate network learning, a single input's effect on the final output is hidden. The second is the unlikeliness of finding the true global minimum when training network parameters. Because these parameters are iteratively improved by some, usually fixed, amount, the probability that the final iteration step finds an optimal solution to the predictive problem is very small. The third is the method's large sample requirements. In the literature from mid 1990s - early 2000s the considered sample sizes are very small relative to common modern deep learning applications[5]. Neural networks are inherently data hungry because of the large number of parameters that need to be estimated, and the large number of different patterns which must be presented to the models for them to generalize well. Fourth, neural networks have a tendency to fit training data too closely, a problem known as *overfitting*. Based on this issue, Kuan (2006) argues that the implementation of these models should be handled with care. In Nakamura (2005), approaches to avoid the *overfitting problem* are found to be paramount to performance. The final issue with neural networks is the time consuming trial and error process for designing networks that perform well. One of the main reasons the community has adopted such a laborious design process is the vast number of different *hyperparameters* that must be selected in order for networks to perform well. In addition, changing one parameter changes the accuracy of the networks in a way that is hard to predict. *Ceteris paribus* comparisons of different network parameters are therefore seldom an optimal approach, and thus trial and error experimentation ensues.

3 Theory

We begin with a quick introduction to the difference between *data modelling* and *machine learning*, the umbrella term under which the field of deep learning resides, showing the philosophical and practical difference between the two approaches. As described in Breiman et al. (2001), consider data being generated by some unknown function $f(\mathbf{x})$ which converts inputs $\mathbf{x}$ to outputs y

$$y = f(\mathbf{x}) + \tag{1}$$

where is some stochastic and unknown error term. Both the data modeling and the machine learning approaches are concerned with $f(\mathbf{x})$. For researchers subscribing to the data modeling

[5]Some authors, such as Nakamura (2005), use *quarterly* observations of inflation, training, validating and testing networks on a total of 175 observations. Makridakis et al. (2018b) use several univariate time series, where the longest training period spans 104 observations. In contrast, a modern computer vision dataset, the CIFAR-10, contains 60 000 images.

culture the analysis starts with assuming a stochastic data model for this relationship, giving rise to the popular first words of many articles *"assume that the data are generated by the following model: ..."*. The test of whether this model actually mirrors the nature of $f(\mathbf{x})$ is based on the in-sample *goodness-of-fit* and examination of model residuals against the assumptions underlying the fitted model, in addition to hypothesis driven analysis of model coefficients.

On the other hand, the analysis based on machine learning considers $f(\mathbf{x})$ as a complex and unknown function. The goal is to find another function $\hat{f}(\mathbf{x})$ which utilizes the information contained in the input $\mathbf{x}$ to predict the response y. The test for whether this approximated function closely fits the real world is to measure its predictive accuracy when new data is presented.

Methodologically, the predictive acuity of novel machine learning methods are evaluated by comparing their accuracy against alternative modelling approaches on the same predictive task. This facilitates the need for one or more *benchmark* models, which sole function is to be a yard stick for novel methods claimed to improve accuracy.

In the following sections we outline our choice of benchmarks before delving into the theoretical foundations of deep learning.

3.1 Benchmark Models

3.1.1 The Random Walk Model

The *random walk model* is one of the simplest forecasting methods, and has proven to be remarkably accurate in forecasting financial and economic time series (Hyndman & Athanasopoulos, 2018). This model is

$$y_t = y_{t-1} + \ _t \tag{2}$$

and simply assumes the present value of a series as a forecast for all future values. The model is especially useful for series exhibiting nonstationarity. Obviously, if the underlying series follows a random walk, the highest probability of correctly predicting the direction of the next step is 50%. Thus, the optimal forecast is to guess the same value as the previous timestep. The forecast is the same for all horizons.

3.1.2 The ARIMA Framework

Our other benchmarks are based on the *autoregressive integrated moving average* (ARIMA) framework, which is one of the most widely used forecasting approaches for univariate time series in general (Hyndman & Athanasopoulos, 2018), and also the most common benchmark for neural networks (Gonzalez, 2000). The framework can be applied to different time series

dynamics through its combination of the autogregressive (AR) and moving average (MA) models with a differencing factor. This linear parametric framework assumes that the underlying data generating process is stationary, meaning that the properties of the series do not change over time. Consider the generalized ARIMA model for a given stochastic time series process $\{y_t\}$

$$\Delta^d y_t = \delta + \sum_{i=1}^{p} \rho_i \Delta^d y_{t-i} + \ _t + \sum_{j=1}^{q} \theta_j \ _{t-j} \tag{3}$$

where d is the series´ order of integration[6], δ is an intercept term, p is a given number of *past values* of y_t, ρ are the coefficients related to each of these past values, $_t$ is the contemporaneous error term, assumed to be independently and identically distributed, q is a given number of past *error terms* and θ the related coefficients of these error terms. The optimal combination of p, d and q depends on the structure of correlations in $\{y_t\}$. The data modelling approach to find these parameters is to investigate the correlation structure between present and past values to select which of these values are important in explaining the present value. Once p, q and d are chosen, the model parameters are estimated using maximum likelihood[7].

The ARIMA framework can be decomposed to its constituent models through specific choices of p and q. For a given d, the case when $p > 0$ and $q = 0$ yields an *autoregressive* model (AR), where the series is modelled using only past values of itself. Similarly, $p = 0$ and $q > 0$ yields a *moving average* model (MA), where the series is modelled using only past values of the error term. The combination of $p > 0$ and $q > 0$ gives us an *autoregressive moving average* model, where the series is modelled using both past values of itself and previous errors.

As outlined later in section 5.2, we apply algorithmic approaches in order to find optimal ARMIA parameters. Two of the approaches are based on goodness of fit measures called *information criteria*. These are measures based on the likelihood function used for estimation L. The first is the *Akaike information criterion* (AIC), which directly focuses on prediction (Hyndman & Athanasopoulos, 2018). The AIC measure is defined as

$$AIC = -2log(L) + 2(p + q + k + 1), \tag{4}$$

where L is the likelihood estimate, p and q are the number of past values and past error terms included as model features and k is an indicator taking the value 1 if the intercept coefficient takes a value > 0. The second is the *Bayesian information criterion* (BIC), defined in terms of AIC as

$$BIC = AIC + (log(T) - 2)(p + q + k + 1) \tag{5}$$

[6]The number of times the series needs to be differenced before becoming stationary.

[7]This method finds the values of ρ and θ that maximizes the probability of obtaining the observed data.

where T is the number of time steps in the time series. For both measures, the optimal model is the one which minimizes the functions in either equation (4) or (5). Practically, the AIC gives a larger penalty to more complex frameworks, and is often preferred for forecasting models, while the BIC often is preferred for inference. For a more thorough introduction to the ARIMA framework, see e.g. Bjørnland and Thorsrud (2015).

The benchmarks are further explained in section 5.2.

3.2 Deep Neural Networks

The following outline of the foundations of deep neural networks is based on Chollet and Allaire (2018) and Goodfellow, Bengio, and Courville (2016). We restrict our scope to the suite of feed-forward[8] structures, namely the multi-layer perceptron (MLP) and convolutional neural network (CNN).

3.2.1 Neurons, the Perceptron and the Multi-Layer Perceptron

The most fundamental building block of neural networks is the computational neuron. A neuron is a function $\xi(\mathbf{x})$ which performs a differentiable affine transformation on a vector of inputs, $\mathbf{x}$, following

$$\xi(\mathbf{x}) = \Phi(\omega^T \mathbf{x} + b), \tag{6}$$

where $\omega = [\omega_0, \omega_1, ..., \omega_p]^T$ is a vector of *weights*, $\mathbf{x} = [x_0, x_1, ..., x_p]^T$ is a vector of *features*, b is a bias term, and Φ is an element-wise *activation function*.

The most basic predictive model implementing neurons is the perceptron, introduced in Rosenblatt (1958). Figure 1 shows a graphical representation of a perceptron model.

[8] Meaning that information flows in one direction through the network, from inputs to outputs, as opposed to *recurrent neural networks*, where information about previous activations are looped back into the network. See Chollet and Allaire (2018) for more on recurrent neural networks.

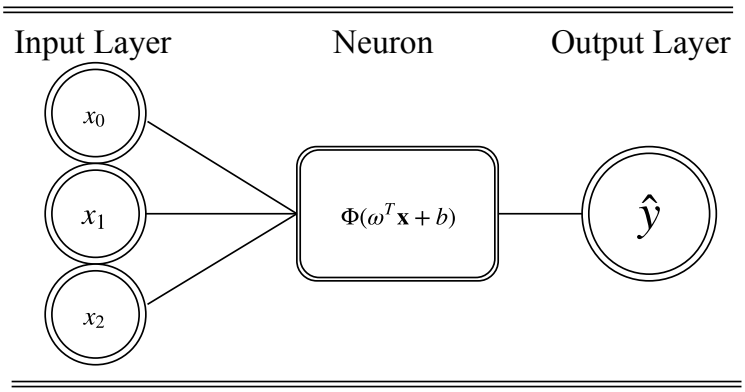

Figure 1: Directed flow graph of a perceptron model. The depicted network consists of an input layer with three nodes, one computational neuron, and one output layer. The inputs are fed to the neuron, producing a scalar prediction of some target variable y.

Notably, for the identity activation function $\Phi(\omega^T \mathbf{x} + b) = \omega^T \mathbf{x} + b$, the perceptron in figure 1 is identical to the linear regression model, where the neurons in the input layer are the input variables, the *features*, the output neuron is the dependent variable, or *target variable*, the weights ω are the estimated coefficients and the bias b is an intercept term (Gonzalez, 2000). Even for nonlinear Φ, a single layer perceptron will never be able to model nonlinear relationships, due to it only having one node of computation (Hall & Cook, 2017).

To enable a network to learn such patterns multiple neurons similar to equation (6) are stacked in a layered structure, creating a *multi-layer perceptron* model. The objective of the network is to create a functional approximation of $f(\mathbf{x})$, f through leveraging the intermediary functional representations produced by these layers of neurons. Defining the layers as $\Upsilon = f(\mathbf{x}; \theta)$[9], f can consist of an infinite number of intermediary functions $f_2, ..., f_n$, so that $f = f_n...(f_3(f_2(f_1(x; \theta_1); \theta_2); \theta_3); ...; \theta_n)$, where $\mathbf{x}$ is the input layer and f_n is the output layer. How the remaining layers is used to help approximate f is dictated by a learning algorithm, and their behaviour is not explicitly defined (Touretzky & Pomerleau, 1989). They are therefore commonly referred to as *hidden layers*.

Figure 2 depicts a multi-layer perceptron with three hidden layers, where each layer consists of several neurons as in equation (6), each performing a transformation on the inputs before passing it on to the next layer.

[9] where θ consists of the weights, ω, and the bias vector b.

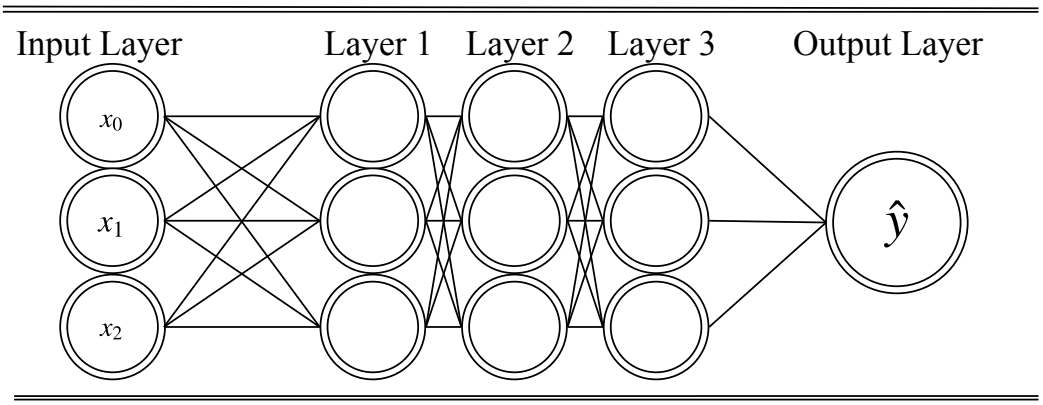

Figure 2: Directed flow graph of a multi-layer perceptron. The network consists of an input layer with three nodes, followed by three hidden layers, each with three computational neurons. The hidden layer is followed by an output layer, producing a scalar prediction of some target variable y.

Notice how the network connects every neuron of one layer to all neurons in the next layer, creating a densely connected network of computational nodes. A network with one hidden layer, a nonlinear activation function and a sufficient number of neurons can approximate any piecewise continuous function to any desired level of precision (Hornik, Stinchcombe, & White, 1989), giving such networks the *universal approximator* property (Kuan, 2006). It is through this property that neural networks are able to recognize complex nonlinear patterns in the data (Gonzalez, 2000). The term *neural network* has come to represent a multi-layer perceptron with a single hidden layer. This network structure is ubiquitous in the literature. Zhang et al. (1998) argue that the focus on single hidden layer neural networks, as opposed to networks with more layers or other structures, is due to a single hidden layer being sufficient for the universal approximator property.

3.2.2 Convolutional Neural Networks

We focus on an additional feedforward network structure, the *convolutional neural network*. These networks have had great success in image recognition tasks, but can be applied to any problem where the inputs have a meaningful spatial structure[10] (LeCun & Bengio, 1995).

These networks have at least one *convolutional layer*, as in equation (7)

$$\xi = \Phi((\mathbf{x} \quad \mathbf{K})(\tau) + \mathbf{b}), \tag{7}$$

Which applies the *convolution operation* instead of matrix multiplication

[10]Recently, similar networks with *dilated kernels* have had great success in audio generation (Van Den Oord et al., 2016), and also for time series forecasting (Borovykh, Bohte, & Oosterlee, 2017). We focus on the general structure of these networks, without the dilated kernels, as described in Chollet and Allaire (2018).

$$(\mathbf{x} \quad \mathbf{K})(\tau) = \sum_{\tau=0}^{t} \mathbf{x}(\tau)\mathbf{K}(t-\tau), \tag{8}$$

where K is a *kernel* which is *convolved* over $\mathbf{x}$. For image processing, the kernel is usually a $2x2$ matrix of weights, called a 2 dimensional kernel. Time series only have one spatial dimension and the kernel is therefore usually a 1 dimensional kernel. The kernel slides across the input data, giving as outputs weighted sums of a given number of the inputs. The outputs from a convolutional layer are called *feature maps*. The width of the kernel, deciding how many of the inputs are extracted at each step, is known as the *kernel size*, and is one of the additional hyperparameters of these networks. Essentially, convolving by a kernel with a kernel size of 2 applies the weight matrix

$$W_C = \begin{bmatrix} W_0 & W_1 & 0 & 0 & 0 & \dots \\ 0 & W_0 & W_1 & 0 & 0 & \dots \\ 0 & 0 & W_0 & W_1 & 0 & \dots \\ 0 & 0 & 0 & W_0 & W_1 & \dots \\ 0 & 0 & 0 & 0 & W_0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \end{bmatrix}$$

As we see from the matrix, a convolutional layer connects the inputs to only a few neurons using the *same* weights. This allows the networks to extract features from local patches of inputs, which allows for modularity in the functional representations which can be leveraged for prediction (Chollet & Allaire, 2018). Weight sharing also leads to data efficiency which may improve the network's accuracy when less data is available (LeCun & Bengio, 1995). A convolutional layer can consist of several *filters*, each with its own set of weights. Each filter learns to recognize specific local patterns important for predicting the output. Recently, Oord et al. (2016) introduced *causal convolutions*. This is a method to stop convolutions from violating local temporal ordering when transforming inputs to outputs. This type of convolution is preferred when working with time series (Chollet et al., 2015).

One interesting feature of convolutional neural networks is their ability to model multivariate inputs through feature *channels*. Given $m > 1$ univariate time series, a convolutional layer may learn to recognize patterns in each series simultaneously. The network passes a seperate kernel over each of the univariate series, producing filters of equal width, which work in unison to predict the output (Zheng, Liu, Chen, Ge, & Zhao, 2014).

A common implementation is *pooling* filters, used to extract the most pertinent information from the outputs of convolutional layers. The most common is the *max pooling filter*. Practically, this filter slides a fixed size 1 dimensional convolutional kernel across its inputs, outputting the maximum value of the inputs within the kernel. The size of this max pooling kernel is known as the *pool size*, and is inversely proportional to the length reduction of the inputs. For instance,

a pool size of 2 halves the length of the inputs.

3.2.3 Activation Functions, Normalization and Output Layers

The rectified linear unit (ReLU) activation function introduced in Glorot, Bordes, and Bengio (2011) has been the de-facto standard since its conception, and is the first choice when selecting the activation function of a hidden layer (Goldberg, 2016). The function returns a positive value or zero, which helps reduce the number of parameters in the network (Chakraborty & Joseph, 2017)

$$ReLU(\omega^T \mathbf{x} + b) = max(0, \omega^T \mathbf{x} + b) = \begin{cases} 0 & \omega^T \mathbf{x} + b < 0 \\ \omega^T \mathbf{x} + b & otherwise \end{cases} \tag{9}$$

When the activation function returns an output of 0, or close to 0, the neuron barely responds to received impulses indicating that these are not important in approximating of $f(x)$.

Because ReLU is unbounded, networks can learn to be too dependent on a given input. It is therefore important to normalize the input data before passing it through the network. A common practise is to normalize the data to the range $[-1, 1]$ following

$$z(x) = \frac{1 - (-1)}{max(x) - min(x)} \quad (x - max(x)) + 1, \tag{10}$$

The choice of activation function (and number of neurons) in the output layer depends on the predictive task. For a scalar regression, the most common is to use a linear activation function and a single neuron, $\Phi(\omega^T \mathbf{x} + b) = \omega^T \mathbf{x} + b$. By applying a single neuron in the output layer, we force a vector to scalar transformation. Thus, the output layer is a linear combination of the incoming signals.

3.2.4 Loss Functions and Optimization

The parameters in each layer, θ, are determined jointly to minimize the deviance between our predictions and the actual target values through a given *loss function*. The choice of loss function, as with the choice of activation function, depends on the predictive task at hand. A common function for scalar regression is the *mean squared error* (MSE)

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{11}$$

where y_i is the realized value of observation i, and $\hat{y}_i$ is the model's prediction of observation i. The loss function is minimized through an iterative optimization technique applying the *back-*

propagation algorithm[11]. The first iteration initializes the parameters using small randomized values, and uses these parameters to make a prediction of either a single observation, or a *batch* of observations. How many observations each iteration predicts is known as the *batch size*. The loss function is calculated based on this initial prediction, before the parameters are adjusted in the direction of the negative gradient of the function with respect to the weights. How far in this direction the weights are altered is dictated by the *learning rate*. In this way, the network incrementally improves its predictions either until the procedure terminates or some other criterion is fulfilled. This is the reason we say that the models *learn*.

3.2.5 Hyperparameters, Capacity and Regularization

The number of hidden layers defines a network's *depth*, while the number of neurons in each layer defines a network's *width*. The depth and width of a network jointly defines its *architecture* and also the *hypothesis space* of possible functional representations which can be leveraged. The flexibility of functional representations a network can leverage decides its *capacity*. If a model has too much capacity, it can learn to fit the training data too closely. This is known as *overfitting* the data, and can hamper the model's accuracy when predicting on new data. If a model has too little capacity, it can not learn sufficiently complex relationships between inputs and outputs, leading to *underfitting*. Monitoring and negating overfitting is a vital part of designing good networks. The data being modelled is often partitioned into training, validation and test datasets, where such monitoring is the main purpose of the validation dataset. Balancing model capacity can be done efficiently by comparing optimization convergence between data used to train the model and the unseen validation data. The act of finding optimal network capacity is known as *hyperparameter tuning*. Any modifications we make to a learning algorithm to reduce its out-of-sample error, but not its training error, is called *regularization* (Goodfellow et al., 2016). We focus on *dropout layers* as a means of regularizing our networks.

Dropout regularization introduces layers which randomly *drops* a certain percentage of the incoming connections in an attempt to prevent the learning algorithm from becoming too dependent on specific inputs (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). The researcher chooses the position of dropout-layers, and the rate at which connections are dropped at each layer.

3.2.6 Residual Connections

Networks with a deeper structure have the same properties as more shallow structures, but with an exponentially lower number of training parameters (Mhaskar, Liao, & Poggio, 2016). One of the main historical issues for building deeper networks has been the *vanishing gradient problem*.

[11]For a more detailed explanation see Rumelhart et al. (1986)

When more layers are added, the gradient based weight updates may become vanishingly small. Another issue with building deeper networks is the *degradation problem*, where accuracy is saturated with increasing network depth. He, Zhang, Ren, and Sun (2016) solves these problems through the implementation of *residual connections*[12], which are skip connections between the inputs and outputs of the layers in a network. The idea is that, given an ideal underlying mapping $H(\mathbf{x})$, we let the layers in the network fit a residual mapping $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$. Solving for the original mapping we get $H(\mathbf{x}) = F(\mathbf{x}) + \mathbf{x}$. This formulation is realized through performing identity mapping between the inputs and outputs to each layer, called residual connections.

Recently, Hall and Cook (2017) applied residual connections to build several deep architectures which performed well in forecasting monthly US unemployment.

3.2.7 Repeat Ensembles

In the forecasting community, combinations of forecasts have been found to increase forecasting accuracy (Makridakis & Hibon, 2000). Timmermann (2006) argues that this accuracy gain is due to a diversification effect. This effect is larger if forecasts from the included models are less correlated (Timmermann, 2006). All models are inherently wrong to some degree, but can incorporate different input information, estimation techniques or assumptions. Therefore a combination of the forecasts produced by several different models will be more robust. Forecast combinations for macroeconomic indicators such as inflation are popular among central banks. For instance Bjørnland, Gerdrup, Jore, Smith, and Thorsrud (2010) outline the Norwegian central bank's SAM-system, which consists of 140 different individual models, and is actively used in the bank's decision making. In the machine learning community, forecast combinations are known as *model ensembles*.

Because neural networks are initialized with randomized weights the learning algorithm may, in repeated runs of the same network, terminate in different local optima. This inherent stochasticity can have a large impact on the predictive accuracy of a single network architecture in repeated runs. One approach to mitigate this stochasticity is to train each network architecture several times, and either choose the error measure of the best performing network, or report an average error across all networks in order to evaluate the architecture's predictive prowess (Hall & Cook, 2017). Another approach is to harness the variation in each training run to create model ensembles. In order to create these *repeat ensembles*, each neural network architecture is trained several times and the forecasts from each individual network is combined using some linear combination. Because the network initialization is random there is no way, a priori or ex post, to know which of the individual models will be better at forecasting. Therefore, we combine the forecasts using the simple mean. Given k repeated runs, the ensemble forecast for observation i is given by

[12]The ReLU activation function also alleviates the vanishing gradient problem, see Glorot et al. (2011).

$$\hat{y}_i^{ensemble} = \frac{1}{k} \sum_{j=1}^{k} \hat{y}_{ij} \tag{12}$$

where $\hat{y}_{ij}$ is the j-th training run forecast of observation i. This approach is obviously computationally intensive. Therefore, repeat ensembles are especially applicable when data is scarce. Additionally, these ensembles allow us to be somewhat agnostic as to the combination of learning rate and the number of training iterations.

4 Data

To test the performance of different neural network architectures for macroeconomic forecasting, we apply these methods to Norwegian inflation. We begin with the unadjusted Norwegian consumer price index (CPI), in the period March 1920 to December 2017. The series was gathered from Macrobond on March, 21. 2018. We start with unadjusted data because the available series has significantly more observations[13]. To adjust the data for potential seasonal variations, we apply the *X13ARIMA-SEATS* filter through the `seasonal` R-package. See Hyndman et al. (2017) for a thorough explanation of this approach to seasonal adjustment. We transform the adjusted series to monthly year-on-year consumer price inflation following

$$\pi_t = \Delta_{12} log(CPI_t) = log(CPI_t) - log(CPI_{t-12}) \tag{13}$$

This choice of year-on-year inflation is due to the fact that the Norwegian central bank uses year-on-year inflation as the target variable. The Norwegian central bank focuses on the energy and tax adjusted measure of core inflation (CPI-ATE), but the bank also publishes forecasts based on the headline CPI measure. As can be seen from Figure 38 in Appendix D, the resulting inflation series exhibits erratic behaviour prior to the *cut-o* line in 1975, probably due to different recording standards. For our models to have optimal accuracy, we need high quality data. We therefore decide to start our *CPI* series in January 1974, before applying the year-on-year inflation transformation on the shorter series. The resulting series is depicted in figure 3.

[13]The first available observation of the series adjusted by the Norwegian statistical agency (SSB) is from January 1985.

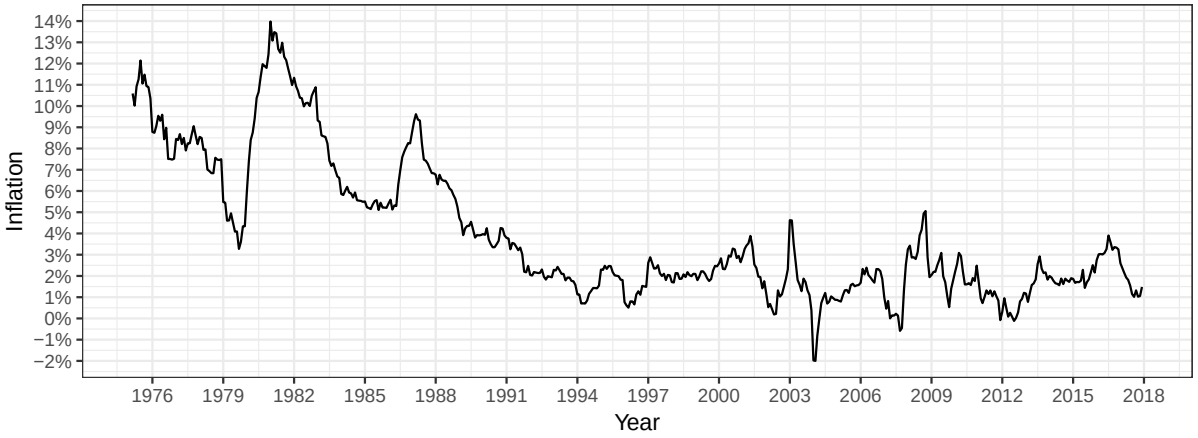

Figure 3: Monthly Norwegian year-on-year consumer price inflation 1975 - 2017

The series of inflation spans the period January, 1975 - December, 2017, where we lose one year of observations due to our year-on-year transformation. This gives us a total of 516 monthly observations.

As evident, the Norwegian inflationary process has changed over time. Before 1993, the series seems to have a declining trend, and higher relative peaks and troughs. The sudden change can be explained by the implementation of inflation targeting in 2001, unofficially a part of the Norwegian central bank's policy since 1998. From this point the Norwegian central bank makes policy decisions to keep inflation stable at a target value. This change in regime is not compatible with a stationary process for Norwegian inflation in the 1975 - 2017 period.

Stationarity is necessary for our ARIMA benchmarks. While neural networks do not require such an assumption, the presence of these breaks may have implications for the method´s predictive accuracy (Stock & Watson, 1998). A common approach to deal with structural breaks when the break-time is known is simply to estimate one model before the break and one model after (Bjørnland & Thorsrud, 2015). For our neural networks this yields fewer total training observations, which may lead to worse performance. Through preliminary comparisons of network performance using the whole series and the post-1993 period, we find that the networks using the whole sample perform better than the ones where the training set starts in 1993. At a similar juncture, Moshiri and Cameron (2000), after testing the Canadian inflation rate for a structural break due to the shift to inflation targeting, difference their series until it is stationary before training their neural networks. Through preliminary analysis, we find that networks trained to forecast the first difference of inflation perform worse in terms of the level of inflation, and have a more homogenuous structure[14].

Hall and Cook (2017) argues that because of the self-regularizing nature of neural networks, researchers can be less selective with how we pre-process the data in question. Because our networks seem to prefer the non-stationary series, we decide to use the full sample to train our

[14]Essentially, all networks were optimal with a 2 lag specification.

neural networks. Since our benchmarks require stationarity, we estimate these on the post-break sample 1993 - 2018 (Series for 1993 - 2018 depicted in figure 39 in Appendix D).

5 Methodology

In this section we introduce our evaluation methodology, our benchmark methods and our network architectures. We begin by outlining time-series cross-validation, the method we use to design network architectures, and our particular evaluation metrics. We then explain our benchmark methods, before delving into neural network designs.

We follow the terminology of Stock and Watson (1998). A forecasting *model* is a singular model which is either estimated once, or re-estimated each period to produce forecasts. A forecasting *method*, on the other hand, can apply any information available at the time to produce forecasts, for instance averaging over several models. With this terminology, repeat ensembles are forecast methods that consists of multiple individual models.

5.1 Model Evaluation

There are several ways to get good estimates of out of sample forecast accuracy. A common approach in the machine learning community is to split the data into *training* and *test* sets, where the model parameters are estimated on the training set and out of sample performance is calculated on the test set. For small data problems, re-sampling techniques are preferred. One of these is *k-fold cross-validation* (James, Witten, Hastie, & Tibshirani, 2013). In this procedure, the data is randomly shuffled and split into k equally sized *folds*. We then iterate over the k folds, using each fold as a separate test set, and the remaining $k - 1$ folds as a training set. The cross-validation error is the average error measure across all folds. When we are faced with a temporal dimension, random sampling causes issues. Particularly, a model trained on future data may lead to *look-ahead-bias* (Choudhary & Haider, 2012). For this reason, *time-series cross-validation* (Hyndman & Athanasopoulos, 2018) partitions the data into training and test sets which are adjacent in temporal ordering. Figure 4[15] outlines the procedure.

[15]This figure is based on https://gist.github.com/robjhyndman/9fa152c585442bb076eb42a30a020091

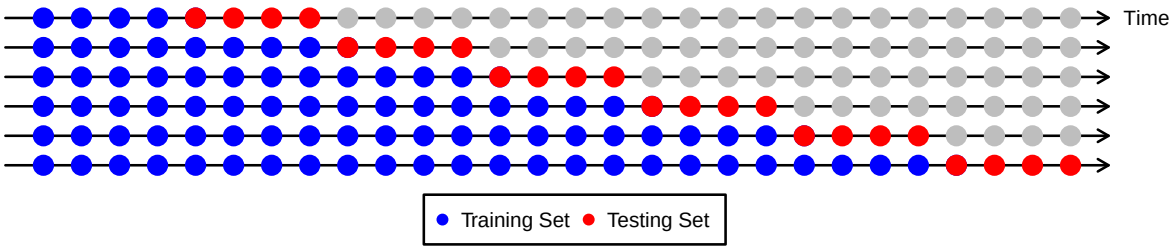

Figure 4: Time-series cross-validation. Each row represents a different temporal fold of the data. The training set is shown in blue, while the test set is shown in red. Moving along the time dimension the training set is expanded with the test data from the previous fold.

Each row in the figure represents a temporal fold. For each of these folds, a model is trained on the training partition and some error measure is calculated on the test partition. The average error measure over all the folds becomes our *cross-validation* error. While the figure illustrates the case where each test set consists of four observations, any number of observations can be sequestered into each test set. The first training set, which in the figure is represented by four blue observations, generally contains enough observations for the forecasts in the first test set to be reliable. The most robust estimates of out of sample accuracy are gained through creating a fold for each observation. This particular approach is at the forefront when evaluating machine learning and deep learning models for time-series forecasting (Hyndman & Athanasopoulos, 2018). The illustrated figure above produces 24 out of sample forecasts by training a model on each of the 6 folds. With one test observation per fold, 24 models need to be trained on separate folds to produce the same number of forecast. Because of the computational expense of the latter procedure, the trial and error process of designing deep neural networks quickly becomes infeasible.

Because of this potential trade-off between estimate robustness and computational expense, we apply a two-step evaluation procedure. To monitor overfitting and design network architectures, we apply time-series cross-validation as illustrated in Figure 4, where each test set consists of twelve months of observations, in the *validation period*. To test the final network architectures, we apply time-series cross-validation with monthly temporal folds in the *test period*. In addition to giving more robust estimates of the test period forecast errors, this procedure is also equivalent to *pseudo out of sample forecasting*, which is one of the most common approaches in the statistical forecasting literature, see e.g. Stock and Watson (1998).

Our test period contains observations of inflation between January 2010 and December 2017, yielding 96 test observations. The choice of validation period is dictated by both the minimum number of observations for a reliable first forecast, and the fact that our benchmark models are estimated on the post-break sample 1993 - 2018. Tkacz and Hu (1999) guesstimate that the point where neural networks improve noticeably over linear models at approximately 300 observations. This gives us the period January 2000 to December 2009 for validation, yielding 298 initial observations for our neural networks and 84 initial observations for our benchmarks

(See end of Section 4). Figure 42 in Appendix D.5 shows the validation and test periods, while table 15 shows summary statistics for both periods.

Practically, this gives us 9 data partitions for validation, where the first training set ends in December 1999, the second in December 2000 and so on. Which twelve observations that are sequestered into the accompanying test set depend on the forecast horizon. Using a one month horizon as an example, the first test set consists of the observations for January 2008 - December 2008. For a twelve month horizon, the final test set contains observations December 2008 - November 2009[16]. Notice that we keep the number of forecasted values fixed for each horizon. For each of these 9 data partitions, we train our benchmarks and networks on the training set and calculate the forecast error on the accompanying test set. For each temporal fold we calculate error measures, and report the average error measure over all folds as our final evaluation metric.

For the test procedure, the first training set stops in December 2009. Assuming a forecast horizon of one month, the accompanying test set contains the observation for January 2010. We follow the same approach with a fixed number of forecasted values. This means that we get 84 distinct training and test splits for each forecast horizon in the test period. Instead of report the average error measure over all folds, we calculate the error measures based on the out of sample forecasts.

5.1.1 Evaluation Metrics

There are several possible measures for evaluating forecasting performance, see e.g. Shcherbakov et al. (2013). The characteristics and emphasis differ between measures (Bjørnland & Thorsrud, 2015). When evaluating the relative performance of different forecasting methods on the same predictive task, scale dependent metrics based on forecast errors are common. We focus on two error measures, the *Root Mean Squared Error* (RMSE) and the *Mean Absolute Error* (MAE).

The RMSE is defined as

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2} \tag{14}$$

where $\hat{y}_t$ is the time t forecast from a model or method we are evaluating. The RMSE expresses the model prediction error in the same units as the variable of interest, in our case in terms of average percentage point error in inflation. We will report the metric in terms of basis point error in inflation. Because this is a quadratic function of the errors, the metric gives more weight to large errors. When considering forecasting methods for inflation aimed at aiding decision making, this property of the evaluation function is desirable.

[16]The test set ends in November because of the particular way we have coded our evaluation procedure.

The MAE is de ned as

$$MAE = \frac{1}{T}\sum_{t=1}^{T} |y_t - \hat{y}_t| \tag{15}$$

As with the RMSE, MAE expresses the error in the same units as the variable of interest, however, the function is not quadratic and does not give the same weight to large deviations as the RMSE. Crone and Pre mar (2006) stresses the importance of applying a non-quadratic error measure when comparing out of sample forecasting performance.

By comparing di erent error measures indications of speci c forecasting performance can be evaluated (Shcherbakov et al., 2013). For instance, lower relative RMSE may indicate that the approach is better at forecasting extreme cases, while a higher relative MAE may indicate that the approach is more robust.

5.2 Benchmark Selection Methods

Our neural networks are compared to a total of ve benchmarks. Two of the benchmarks are singular models. These are the random walk model (RW), and the best performing autoregressive speci cation in the validation period. The remaining three benchmarks are di erent automatic model selection approaches, which perform model selection for each training and test partition in our evaluation procedures.

5.2.1 Random Walk Model (RW)

The random walk model, as outlined in section 3.1.1, simply guesses that in ation at any horizon is always equal to the contemporaneous value. Thus, the model has no parameters that must be selected. We denote this model RW.

5.2.2 ARIMA Methods Based on Information Criteria (ARBIC & ARAIC)

The rst two methods apply an automated procedure based on the AIC and BIC information criteria. For each training and test partition in our validation and test procedures, we estimate ARIMA models with AR-terms [17] in the range 1 to 12. We select the speci cation with the lowest BIC (ARBIC) and the speci cation with the lowest AIC (ARAIC) in the training set, and use these models to forecast the test set.

[17] Bj rnland et al. (2010) did not nd signi cance for any MA-terms when forecasting quarterly Norwegian year on year in ation, and subsequently did not consider the moving average part of the ARIMA framework when comparing forecasting models. Although our data is at a monthly frequency, we also decide to omit the MA-terms for this procedure.

5.2.3 auto.arima Approach (AA)

The third method is also based on ARIMA and proposed in Hyndman, Khandakar, et al. (2007) and denoted AA. It is implemented through the auto.arima function from the R-package forecasts. This approach has been implemented in forecasting competitions with good results (Makridakis et al., 2018b). For each training and test partition, the procedure finds the order of integration of the training data and searches for the optimal number of AR and MA terms based on information criteria[18]. We limit the choice of AR and MA terms in the range 1 to 12, and let the model consider both differencing and inclusion of trend. This means that the algorithm may also pick the random walk model if a given training partition exhibits non-stationarity.

5.2.4 Autoregressive Model Selection (AR)

In some cases, model selection methods based on in sample criteria, such as the ones above, have potential detrimental effects on out of sample forecasting due to changes in the underlying series (Stock & Watson, 1998). Therefore, we also estimate ARIMA models with AR-terms in the range 1 to 12, and pick the specification with the lowest RMSE in the entire validation period to forecast the test period. These models are denoted AR followed by the number of lags included. For our validation period an AR with two lags has the lowest error at the one month horizon[19]. Further this model is denoted AR2.

5.3 Neural Network Design

Because of the immense number of possible network hyperparameters, we need to restrict our scope prior to modeling. We therefore make a few design choices which stay fixed for all networks. Firstly, we only consider the ADAM optimizer[20] with its default parameters. The default learning rate is set to 0.001. All network architectures are trained using a batch size of 4[21]. The optimizer aims to minimize the MSE loss function, defined in equation (11). Secondly, to produce forecasts for longer horizons, we apply the multi-neural network method described in Makridakis et al. (2018b). This entails training separate networks for each considered forecast horizon. The target information values are shifted forward in time by the forecast horizon. We consider four horizons: 1, 3, 6 and 12 months ahead. We focus on the one month horizon when designing networks, and then use the best network architectures to forecast the other horizons.

[18] The procedure finds the combination of p and q which minimizes an adjusted AIC measure, AICc, defined as, $AICc = AIC + \frac{2(p+q+k+1)(p+q+2)}{T-p-q-2}$. For models with $d = 0$ the model includes an intercept, while for $d > 0$, the model sets the constant to zero. Variations of the models are considered repeatedly until the lowest possible AICc is found (Hyndman et al., 2007).

[19] Validation RMSE for all autoregressive models can be found in Table 6 in Appendix B.1.

[20] For more on the ADAM optimizer, see (Kingma & Ba, 2014).

[21] Each weight update in network training is based on 4 samples of information.

Third, all our final network forecasts are repeat ensembles of a given number of repeated initializations of the same network architecture. Lastly, for network design we only consider the RMSE evaluation metric.

5.3.1 Inputs and Normalization

Through training, the network identifies which inputs and neurons are important in predicting the output. If the model is presented with many input features it should be able to self-regularize to some extent. For instance, the networks should be able to perform lag selection by themselves. Following Hall and Cook (2017) the inputs to our neural networks are lagged values of the rate of inflation ( ) and lagged values of the first and second order differences of inflation (; ^{2} , respectively). The inflation features are normalized for each training and test partition following equation (10) using the training partition´s maximum and minimum values. Normalizing the entire series prior to partitioning into training and test sets could induce look-ahead bias in the results, because the network receives implicit information about the future maximum and minimum values. The outputs of our neural networks are re-scaled to their original values before calculating the evaluation metrics.

5.3.2 Number of Models in each Repeat Ensemble

In order to chose the number of individual models to include in each repeat ensemble, we explore how the validation RMSE evolves as more individual models are added. We train 30 single hidden layer neural networks with 6 computational neurons, applying ReLU activation, and a linear output activation. The network uses p = 12 lagged values as inputs. Figure 5 illustrates how repeat ensemble RMSE and average individual model RMSE evolves.

Figure 5: Number of models to include in the repeat ensemble. Red line represents ensembling RMSE. Blue line represents average RMSE of individual networks in the ensemble. Light grey band represents minimum and maximum validation RMSE for individual models. Darker grey band represents interquartile range (IQR) of individual model RMSE.

Ensemble RMSE is reduced by adding more models, while the average RMSE of the individual models remains noticeably higher. For this exact set of 30 networks, the best performing single model is found after 10 random initializations, while the worst is found after 16. Notice that the model ensemble is very close to the best individual model without performing any model selection. For this exact network architecture, 15 individual models seems like a good balance between computational expense and ensemble performance. There is some evidence that adding more than 30 models could improve performance further, but this quickly becomes computationally infeasible. In order to make the comparison of several architectures feasible, we decide to use fteen models in the repeat ensembles for each network structure.

5.3.3 Lag Selection

When choosing the number of models in each repeat ensemble we used 12 lags as inputs. This number was chosen somewhat arbitrarily. Further analysis indicates that the number of lagged values used in the models are paramount to performance. In order to investigate whether another lag speci cation improves accuracy, we train 15 simple architectures with 26; 12; 24; 30 and 36 lags as inputs. Figure 6 shows a boxplot of the RMSE from the individual models, as well as the ensemble RMSE, for each of the six di erent lag speci cations.

Figure 6: Lag selection. Validation RMSE in basis points for ensembles with di erent lag speci cations. The boxplots show the distribution of RMSE for individual networks. Red line represents RMSE of model ensemble.

From the gure, we see that the number of included lags has an impact on the spread of di erent model forecasts, as depicted by the size of the black boxes. Additionally, we see that the repeat ensembles have noticeably lower RMSE than the individual models. For this network architecture it appears that the initial 12 lag speci cation is a decent trade-o between RMSE and the number of inputs to the model.

When designing neural networks each parameter choice a ects performance in a way that is hard to predict. For instance, increasing the number of neurons or implementing dropout regularization for the networks in Figure 6 could change the optimal number of lags. This gives rise to a chicken and the egg type issue, where the order in which you nd optimal parameters may have large implications for the nal network's architecture. Essentially, this means that the optimal approach would be to run every possible combination of parameters. To simplify this process we initially assume that the same lag speci cation is optimal for all network structures. After designing the three network structures we again consider di erent lag speci cations.

5.3.4 Number of Hidden Nodes and Dropout Regularization

Based on the literature, there is no direct way of choosing the number of hidden units. While research on the single hidden layer neural network suggests that 6 computational neurons is a good initial value (Makridakis et al., 2018b), no initial values have been proposed for our other network structures. We therefore nd the initial values for the latter two networks through trial and error. We then attempt to improve the networks through coarsely increasing the number of hidden neurons. For each number of hidden units, we increment the rate of dropout in the range $[0.0; 0.1; 0.2; 0.3]$.

5.4 Final Network Architectures

For the sake of brevity, the performance and diagnostics for alternative speci cations of each of the three neural networks is moved to appendix A. In the following section, we brie y introduce our nal network architectures.

We compare three di erent network structures. The rst is a single hidden layer neural network (NN). The second is a deep multi-layer perceptron with residual connections (MLP). The third is a deep convolutional neural network with residual connections (CNN). Through employing these di erent network structures we can explore how robust performance is to di erent speci cations, in addition to exploring whether the architectural implementations in the two latter networks improve forecasting performance.

5.4.1 Neural Network (NN)

The rst network architecture is based on the single hidden layer neural network applied in Makridakis et al. (2018b)[22]. Figure 7 shows the nal network architecture.

Figure 7: Architecture of the Neural Network (NN). Dense layers are equivalent to hidden layers.

The network consists of an input layer, one hidden layer (dense layer) and an output layer. The network takes an input vector of 36 values, 12 for each of the three in ation features, and passes it through a hidden layer with six computational neurons. The layer applies the ReLU activation function to its outputs before passing it on to the output layer. The output layer, consisting of a single computational neuron applying linear activation, produces a scalar prediction at the forecast horizon.

5.4.2 Multi-Layer Perceptron (MLP)

Figure 8 shows the nal network architecture for our multi-layer perceptron. The network is based on Hall and Cook (2017) and has the same overall structure.

[22]The code for this network, as well as other machine learning benchmarks for time series, is publicly available on github: https://github.com/M4Competition/M4-methods/blob/master/ML_benchmarks.py

Figure 8: Architecture of the Multi-Layer Perceptron (MLP).

We include three dense stacks, which are clusters of hidden layers with the same number of neurons. Each of these stacks consists of three hidden layers that apply the ReLU activation function de ned in equation (9). Each of the three layers in the rst stack contains eight computational neurons. The number of neurons in the following stacks are reduced to six and four, respectively. The output of each stack is passed through a dropout layer with a dropout rate of 0.1, before entering the next stack. Between these stacks we add residual connections, which perform element-wise addition between the output of the stack and the inputs to the stack.

After deciding on a nal network architecture, we consider di erent lag lengths. For the multi-layer perceptron model, we nd that a 24 lag speci cation is optimal[23]. For this 24 lag speci - cation, the input layer consists of 72 input nodes, 24 for each of our in ation features.

5.4.3 Convolutional Neural Network (CNN)

Figure 9 shows the nal network architecture for our convolutional neural network. Similarly to the MLP, our convolutional neural network is based on Hall and Cook (2017). The authors apply a network utilizing two convolutional stacks, which are pairs of 1-D convolutional layers. Through experimentation we nd that this architecture does not t our particular predictive problem. Our architecture consists of single 1-D convolutional layers as opposed to pairs. In addition the architecture has fewer dense stacks.

Figure 9: Architecture of the Convolutional Neural Network (CNN).

The nal network consists of an input layer, two 1D convolutional layers separated by a max pooling layer, two dense stacks and an output layer. The inputs are passed through a 1D convolutional layer with twelve lters, a kernel size of six and ReLU activation. The outputs of

[23] Reducing the cross-validation error from 47.2 to 46.1 basis points.

the convolutional layer are fed to a max pooling layer, halving the size of the feature map. The reduced feature map is fed into another 1D convolutional layer identical to the first. The first dense stack consists of three hidden layers, each with six computational neurons applying ReLU activation. This stack is followed by a dropout layer with a dropout rate of 0.2. The outputs of the dropout layer are fed to another dense stack, consisting of three hidden layers, each with four computational neurons and ReLU activation. The outputs of the last dense stack is passed through a final dropout layer, before entering the output layer. The output layer has one computational neuron with linear activation, transforming the inputs into a scalar prediction. We add residual connections between the inputs and the outputs of the first convolutional layer, between the outputs of the max pooling layer and the outputs of the second convolutional layer, and between each dense stack.

We find that a 24 lag specification is optimal also for the convolutional neural network (CNN) [24].

5.5 R Implementation

All models are implemented in the R programming language. The code is written in a reproducible Jupyter Notebook environment, which is available from the authors upon request.

All neural network architectures are implemented using `keras-gpu` version 2.1.5 and `tensorflow-gpu` 1.1.0 through the `reticulate` R-package (Allaire, Ushey, & Tang, 2018). We interface with `keras` through the `R-keras` package (Allaire & Chollet, 2018). We leverage `ggplot2` for visualization (Wickham, 2009), `stargazer` for tabulating (Hlavac, 2018), `lubridate` for iteratively manipulating dates (Grolemund & Wickham, 2011), `tstools` for manipulating time series (Bannert & Thoeni, 2018), the `seasonal` package to implement the X13ARIMA-SEATS filter (Sax, 2017). All neural networks are trained on a 6-core Intel i5 system running at 4.8GHz with an Nvidia GTX1080 graphics card utilizing CuDNN 6.0. The implementations are based on Chollet and Allaire (2018).

6 Results

In this section we report the results of our forecasting comparison. The reported results are for the final network architectures as outlined in section 5.4 for the short term (one and three month) and medium to long term (six and twelve month) forecast horizons. We begin by presenting the results for the validation period before looking at the test period. For each period we first report the forecast accuracy, as measured by our evaluation metrics (RMSE[25] and MAE[26]). Both

[24] The cross-validation RMSE is marginally reduced (51.9 to 51.8).
[25] As defined in equation (14).
[26] As defined in equation (15).

evaluation metrics are negatively oriented, such that a lower value indicates higher forecasting accuracy. We then compare the three network architectures, first against the benchmarks, and then against each other, for the different forecast horizons. Lastly, the performance of the forecasting methods are compared at a yearly level.

6.1 Validation Performance

As mentioned in section 5.1 we apply time-series cross-validation in the 2000 - 2009 validation period. Table 1 reports the evaluation metrics for each of the chosen benchmark methods and neural network architectures.

		Root Mean Squared Error (RMSE)					Mean Absolute Error (MAE)				
	Horizon	1	3	6	12	Mean	1	3	6	12	Mean
Benchmarks	RW	54.99	112.35	144.35	198.09	127.44	40.69	88.54	119.33	162.09	102.66
	AA	51.58	102.91	129.01	160.09	110.90	38.93	83.63	111.84	133.10	91.88
	ARBIC	51.27	98.45	119.04	128.16	99.23	38.58	80.90	100.34	107.54	81.84
	ARAIC	50.90	98.47	121.09	129.46	99.98	38.72	79.74	101.94	110.11	82.63
	AR2	50.83	98.21	118.81	126.04	98.47	38.03	80.78	100.04	106.01	81.22
Networks	NN	50.07	103.58	135.61	175.61	116.22	38.51	79.99	112.93	144.81	94.06
	MLP	46.09	94.91	127.33	161.83	107.54	33.68	74.83	105.40	132.79	86.68
	CNN	52.53	106.52	140.40	166.72	116.54	40.98	85.37	118.18	151.50	99.01

Table 1: Root mean squared error (RMSE) and mean absolute error (MAE) for time series cross-validation in the validation period. RMSE and MAE are reported in basis points. A lower value indicates a more accurate model. The lowest error for each horizon is highlighted in bold face. The Mean is the average RMSE and MAE over all forecast horizons.

6.1.1 Short Term Forecasts: One and Three Month Horizons

Table 1 shows that the best benchmark for the one month horizon is the autoregressive model of order 2 (AR2)[27]. This model yields an average error of 50.83 basis points and 38.03 basis points when forecasting in ation in the 2000 2009 period, measured by RMSE and MAE, respectively.

When considering our neural networks, Table 1 shows ambiguous results for the NN architecture. While NN has a 1.5% lower RMSE than AR2, the MAE is 1.3% higher. The MLP network outperforms all benchmarks, reducing the forecast error by 9.8% and 11.4% relative to AR2 for RMSE and MAE, respectively. The CNN architecture performs worse than AR2, in terms of both RMSE and MAE.

At the three month horizon, AR2 has the lowest RMSE, while ARAIC has the lowest MAE among the benchmarks. Both NN and CNN perform worse than these benchmarks. However, MLP outperforms both benchmarks, decreasing RMSE by 3.4% relative to AR2 and MAE by 6.2% relative to ARAIC.

[27] Remember that the AR2 model is chosen as the best AR model ex post, and as such may have a slightly overstated forecast accuracy in the validation period.

6.1.2 Medium to Long Term Forecasts: Six and Twelve Month Horizons

Table 1 shows that AR2 is the best benchmark for both the six and twelve month horizons. The AR2 yields an average error of 118.81 (100.04) and 126.04 (106.01) basis points, measured as RMSE (MAE) for the two horizons, respectively.

None of the neural network architectures outperform AR2 at the six month horizon. Relative to AR2, the networks increase RMSE by between 7.2% and 18.2%, and MAE between 5.4% and 18.1%. Of the network architectures, MLP has the lowest error measures, reducing RMSE (MAE) by 6.1% (6.7%) relative to NN, and 9.3% (10.8%) relative to CNN.

The results are similar for the twelve month horizon. The increase in RMSE for the neural networks is between 28.4% and 39.3% relative to AR2. Similarly, the MAE is increased by 25.3% to 42.9%. Again, the best network architecture is MLP, outperforming NN and CNN by 7.8% (8.3%) and 2.9% (12.3%) for RMSE (MAE).

6.1.3 Overall Performance in the Validation Period

Overall, AR2 is the best benchmark for forecasting in ation in the validation period. Averaged over all forecast horizons, this benchmark yields a Mean RMSE of 98.47 and a Mean MAE 81.22 basis points. The best overall neural network architecture is MLP, yielding Mean RMSE and Mean MAE values of 107.54 and 86.68 basis points, respectively. The MLP network outperforms RW and AA for both evaluation metrics, while the other neural network architectures are only slightly better than the worst benchmark, RW.

Comparing the overall forecast accuracy of the neural networks, MLP yields a reduction in Mean RMSE of 7.5% and 7.7% relative to NN and CNN, respectively. In terms of Mean MAE, MLP has a 7.8% and 12.4% lower error measure than NN and CNN.

While both NN and CNN are outperformed by, or only slightly better than, benchmarks at all horizons, MLP outperforms the best benchmark at the short term horizons. However, the three network architectures perform worse than the benchmarks at the medium to long term horizons.

6.2 Test Performance

As mentioned in section 5.1, we apply a pseudo out of sample procedure in the 2010 - 2017 test period. Thus, the reported error measures are estimates of the real world accuracy of the di erent methods if they were re-estimated each month in the test period. These results are

reported in Table 2 for the chosen benchmark methods[28] and neural network architectures[29].

		Root Mean Squared Error (RMSE)					Mean Absolute Error (MAE)				
	Horizon	1	3	6	12	Mean	1	3	6	12	Mean
Benchmarks	RW	35.83	60.65	79.00	124.99	75.12	28.30	46.89	64.05	100.14	59.85
	AA	36.42	61.17	76.39	93.53	66.88	28.43	46.38	57.65	73.26	51.43
	ARBIC [30]	36.61	63.38	80.24	93.34	68.39	28.73	47.92	60.01	72.96	52.41
	ARAIC	37.37	64.00	79.25	100.80	70.36	29.12	50.74	62.71	79.37	55.49
	AR2	36.61	63.38	80.24	93.34	68.39	28.73	47.92	60.01	72.96	52.41
Networks	NN	33.49	59.60	80.12	115.05	72.07	26.27	47.28	66.06	98.42	59.51
	MLP	32.58	55.28	74.00	108.15	67.50	25.33	45.66	63.25	90.21	56.11
	CNN	32.64	49.64	68.29	95.81	61.60	26.24	40.93	58.58	79.78	51.38

Table 2: Root mean squared error (RMSE) and mean absolute error (MAE) for the pseudo out of sample approach in the test period. RMSE and MAE are reported in basis points. A lower value indicates a more accurate model. The lowest error for each horizon is highlighted in bold face. The Mean is the average RMSE and MAE over all forecast horizons.

6.2.1 Short Term Forecasts: One and Three Month Horizons

Table 2 shows that all benchmarks perform similarly at the one month horizon, with RMSE ranging between 35.83 and 37.37 and MAE ranging between 28.30 and 29.12. All our neural network architectures outperform these benchmarks, with RMSE ranging between 32.58 and 33.49 and MAE ranging between 25.33 and 26.27. The MLP is the best architecture, and reduces both error measures by between 9% and 13% compared to the benchmarks. The other networks are only slightly worse than MLP. The MAE is around 3.6% higher for both networks, while RMSE is 2.8% higher for NN and 0.18% for CNN, relative to MLP.

At the three month horizon the best benchmark methods, RW and AA, perform similarly, with RMSE ranging from 60.65 to 61.17 basis points and MAE ranging from 46.38 to 46.89 basis points. The NN network performs similar to these, while both the other neural networks are better than all benchmarks. With regards to RMSE, MLP outperforms the benchmarks by approximately 8.9% and 9.6%. However, MLP is only slightly better in terms of MAE with a 1.6% to 2.6% reduction. The best network is CNN which reduces RMSE by 18.2% and 18.8% and MAE by 11.8% to 12.7%, relative to RW and AA respectively.

[28] Model coefficients from the AA, ARBIC / AR2 and ARAIC methods is found in figures 35, 36 and 37 in Appendix B.2.

[29] The actual out of sample forecasts produces by each of our three neural network architectures is found in figures 20, 27 and 34 in Appendix A.

[30] The ARBIC method chose an autogregressive model of order 2 throughout the test period, making the ARBIC and AR2 the same.

6.2.2 Medium to Long Term Forecasts: Six and Twelve Month Horizons

Table 2 shows that the best benchmark at the six month horizon is AA with RMSE and MAE of 76.39 and 57.65 basis points, respectively. The AA method outperforms NN, which is more similar to the other benchmarks. The MLP network shows ambiguous results with a 3.1% reduction in RMSE but a 9.7% higher MAE compared to AA. The CNN is the best network architecture both in terms of RMSE and MAE. It outperforms AA by 10.6% in terms of RMSE. However, the MAE is similar between these methods and differ only by 1.6% in favour of AA.

Considering the twelve month horizon, AR2 is the best benchmark closely followed by AA. These benchmarks have a RMSE of 93.34 – 93.53 basis points and MAE of 72.96 – 73.26 basis points. None of the neural network architectures outperform these benchmarks. However, CNN is the best neural network and is just outperformed by 2.6% in terms of RMSE. However, MAE is 9.3% higher than the two best benchmarks. The other neural networks perform better than the worst benchmark, RW. However, their error measures are more than higher than CNN. The NN architecture has a 20% higher RMSE and 23% higher MAE, while MLP has a 13% higher error across both measures, compared to CNN.

6.2.3 Overall Performance in the Test Period

Overall, CNN is the best method for forecasting inflation in the test period. The best benchmark is AA with 66.88 and 51.43 basis points RMSE and MAE, respectively. The CNN architecture outperforms this benchmark with 7.9% RMSE, while the MAE is marginally lower (0.1%).

Compared to the other neural networks, CNN outperforms NN by 14.5% in terms of RMSE and 13.7% in terms of MAE. The CNN architecture outperforms MLP by 8.7% in terms of RMSE and 8.4% in terms of MAE. Nevertheless, NN and MLP perform within the range of benchmark error measures, both beating the worst benchmark, RW. In terms of RMSE, MLP is better than most benchmarks, and only has a 0.9% higher error than AA. However, with regards to MAE, MLP yields a 9.1% higher error than AA, but still outperforms the worst benchmark, RW, by 6.2%. The NN network is 4.1% better than RW in terms of RMSE, but these methods are similar with regards to MAE (0.6% difference in favour of NN).

6.3 Test Performance by Year

To get a better understanding of the neural network architectures' test performance, we calculate the error measures for each year. The results are presented in figures 10 to 13. In addition to showing the error measures of our different network architectures and the best benchmark, we include the maximum and minimum error measure values of all benchmarks for each year. The

numeric values of the error measures for each year is included in Appendix C for both the validation and test periods. In this section, only the test period is considered.

6.3.1 One Month Horizon

Figure 10: Performance by year at the one month horizon. The left gure shows results for RMSE, while the gure to the right shows the results for MAE. The solid black line represents the error measures from the best benchmark for each horizon, as reported in Table 2. The red, green and blue lines represent the error measures for the CNN, MLP and NN network architectures, respectively. The light grey band shows the maximum and minimum benchmark error measures for each year.

The test performance at the one month horizon is presented in Figure 10. Generally, the neural networks have a similar performance and are better than the benchmarks in the years 2010 - 2013 and 2015 - 2016. However, as illustrated by the grey band, all the benchmarks have a lower RMSE in the year 2014. In terms of MAE, the neural networks perform more similar to the benchmarks.

6.3.2 Three Month Horizon

Figure 11: Performance by year at the three month horizon. The left gure shows results for RMSE, while the gure to the right shows the results for MAE. The solid black line represents the error measures from the best benchmark for each horizon, as reported in Table 2. The red, green and blue lines represent the error measures for the CNN, MLP and NN network architectures, respectively. The light grey band shows the maximum and minimum benchmark error measures for each year.

The performance at the three month horizon is presented in Figure 11. Again, the networks generally perform better in the years 2010 - 2013, while being dominated for the year 2014. There is a larger di erence in architecture performance. For instance, CNN has noticeably better performance for 2010 and 2013, while performing worse than the alternatives for 2014 and 2015.

6.3.3 Six Month Horizon

Figure 12: Performance by year at the six month horizon. The left gure shows results for RMSE, while the gure to the right shows the results for MAE. The solid black line represents the error measures from the best benchmark for each horizon, as reported in Table 2. The red, green and blue lines represent the error measures for the CNN, MLP and NN network architectures, respectively. The light grey band shows the maximum and minimum benchmark error measures for each year.

The test performance for each year at the six month horizon is shown in Figure 12. We see that the neural networks in general perform similar to benchmarks for this horizon. We also see that CNN is better than the best single benchmark for all years except 2014, where the benchmark in general outperform all neural networks.

6.3.4 Twelve Month Horizon

Figure 13: Performance by year at the twelve month horizon. The left gure shows results for RMSE, while the gure to the right shows the results for MAE. The solid black line represents the error measures from the best benchmark for each horizon, as reported in Table 2. The red, green and blue lines represent the error measures for the CNN, MLP and NN network architectures, respectively. The light grey band shows the maximum and minimum benchmark error measures for each year.

Figure 13 presents the test performance for the twelve month horizon. We see that the discrepancy in network performance has grown, and that both the NN and MLP perform worse than the benchmarks. The CNN is better than the other neural architectures and even outperforms the single best benchmark in the years 2011 - 2012.

7 Discussion

7.1 Main Findings

In this thesis we have explored the predictive potential of neural networks in forecasting macroeconomic time series in the Norwegian economy. Through the comparison of di erent methods for forecasting in ation, we nd several interesting results. Firstly, in both the validation and test periods, at least one neural network architecture outperforms the best included benchmark for short term horizons. Secondly, the ubiquitous single hidden layer neural network architecture (NN) applied in the literature is dominated by a deeper multi-layer perceptron architecture with residual connections (MLP) for all horizons, both in the validation and test periods. Third, in the test period, a convolutional neural network with residual connections (CNN) outperforms all alternative forecasting approaches at the three and six month horizons, while performing similarly to the best alternatives at the one and twelve month horizons. Overall, the CNN outperforms all alternatives in the test period.

This section sums up our main ndings, and discusses the best forecasting method of in ation. Further, we discuss these ndings against the literature, and try to answer our research questions.

1. (a) To what extent is neural networks relevant in forecasting Norwegian macroeconomic indicators, such as in ation?

   (b) Given that data is scarce, are there possible architectural implementations which can improve performance over single hidden layer neural networks?

2. Are neural networks a feasible addition to the macroforecasting toolbox?

7.2 What is the Best Forecasting Method for In ation?

Overall, our neural networks seem to perform well at the task of forecasting Norwegian in ation. Our results sheds light on some important topics for discussion which could have an impact on the feasibility of applying these methods in practice. Firstly, we nd that while performing strongly in the test period, the CNN architecture performs worse than the other neural networks in the validation period. Secondly, we nd that the rankings of our included forecasting approaches di er depending on the applied error measure, and when considering di erent forecast horizons. We discuss these in turn, before choosing which of the included network architectures is the most suitable for our forecasting task.

Difference in Validation and Test Performance

The choice of validation approach seems to be very important when dealing with neural networks for macroeconomic time series, as not beating the benchmarks in the validation period is potentially problematic. In practice, the methods that do not improve forecasting accuracy in the validation period would be discarded. If we had discarded the CNN architecture because of relatively poor validation performance, we would not have discovered that the method has the best forecasting accuracy in the test period. Thus, in order to decide upon the best of our forecasting methods, we need to investigate this inconsistency.

In the validation period, the only network architectures to outperform the best benchmarks are NN and MLP, and only at the short term horizons. While the improvement of NN is only slight, the MLP yields a noticeable reduction in the error measures at these horizons. In the test period, all three network architectures outperform the best benchmarks at the short term horizons. Interestingly, the CNN network outperforms all other methods at the three and six month horizons, while the performance is similar to the best alternatives at the one and twelve month horizons.

Why is the performance of CNN noticeably better in the test period than in the validation period? There could be several possible explanations for this. The rankings between forecast methods have been found to be different for different sample periods (Choudhary & Haider, 2012). While some forecast methods may excel at forecasting in times of volatility and structural change, others may perform better in more regular times. We see examples of how the neural network performance differs between years in the figures in section 6.3. Thus, the discrepancy in CNN´s test performance could simply be explained by differences in the behaviour of inflation in the validation and test periods. We found that the CNN network generally performed well in all years except for 2014 - 2015. However, as shown in Figure 42 in Appendix D.5 this period is characterised by low volatility, and a realized inflation close to the test period mean of 2%. All our neural network architectures are outperformed by the benchmarks in this period, for all horizons, while the networks are generally better than the benchmarks for more volatile periods. These results are similar to those in Szafranek (2017), indicating that the networks perform well in times of high volatility or structural change. As outlined in Appendix D.5, the validation and test periods are quite similar in terms of mean inflation, but the former is more volatile. Thus, we would in fact expect the CNN network to perform better in this period. The difference in period is, therefore, a less plausible explanaiton for the difference in validation and test performance.

Another possible explanation may be that the difference is due to overfitting because of the smaller sample size in the validation period. The CNN network has more parameters than the two other network architectures (see network architecture in A.3), making it more prone to overfitting. Appendix A.3.2 shows the CNN network's convergence plots for each of the nine

years in the validation period. Relative to the plots of convergence for the NN and MLP networks in appendices A.1 and A.2, the CNN network does seem to be more prone to over t the data.

Also, the di erent approaches used to estimate error measures in the validation and test periods may impact performance. As outlined in section 5.1, we retrained the network ensembles once each year in the validation period, while retraining the ensembles each month in the test period. This yields a nine-fold increase in the number of individually trained models with random initialisation in the test period. An increase in the number of individually trained models reduces the overall stochasticity due to random initialisation. Since the CNN ensemble seems more prone to over tting and we use the same number of individual models in each of the architectures' ensembles, the CNN ensemble may still exhibit some stochasticity. Therefore, the CNN could perform relatively worse in the validation period, where this residual stochasticity may have an e ect, than in the test period where this e ect is, to a larger extent, averaged out because of the increased number of total models trained.[31]

Finally, the di erence in validation and test periods could be due to the validation period having closer proximity to the structural break discussed in section 4. This entails that the network has (1) more memory of a declining trend and a more volatile series, and (2) that the network has less patterns pertaining to the in ation targeting period. Although the issue of stationarity for deep networks needs further research, evidence suggests that training networks on nonstationary series could have a negative impact on performance (Chollet & Allaire, 2018). For the NN and MLP architectures, the methods perform relatively similar in the validation and test period. Thus, if non-stationarity is the underlying cause for the di erence in performance between the periods, the CNN may be less robust to non-stationarity.

We believe that the most plausible explanation for the di erence is the smaller sample size of the validation period leading to over tting and more stochasticity in the CNN ensemble. The NN and MLP architectures do not seem to be subject to whatever is causing the di erence in performance between the training and test periods. Therefore, it seems that the two alternatives are more robust to the potential causes discussed above. For these reasons, our approach for validation and testing may not be optimal. Perhaps network design could be performed on a smaller validation period closer to the test period. If we had validated our models on e.g. 2010 - 2014, and tested them in 2014 - 2018, we might have more similar performance between the validation and test sets. Another alternative could be to evaluate networks using a di erent procedure. Several other approaches have been proposed, particularly some which do not necessarily adhere to the arrow of time, for instance sequestering a number of randomly chosen samples into a validation set, such as in Hall and Cook (2017), or training networks on bootstrap samples (Szafranek, 2017).

[31] This could be alleviated by performing a similar analysis as in section 5.3.2 separately for the CNN.

Difference in Rankings Based on Different Error Measures

Comparing different error measures with different emphasis is important, and can indicate what situations a given method forecasts well (Shcherbakov et al., 2013). Our results indicate that the rankings of different measures depend on the chosen evaluation metric. This is in tune with the findings in Makridakis and Hibon (2000). The forecasting accuracy of our methods are presented in terms of the error measures RMSE and MAE, and our neural network architectures generally perform better compared to our benchmarks when considering RMSE[32]. A lower relative RMSE indicates that the approach is better at forecasting extreme cases, while a higher relative MAE indicates that the approach is more robust.

The ambiguous cases, such as when the MLP network outperforms the benchmarks at the three month horizon in terms of RMSE, while being outperformed by all but RW in terms of MAE, indicate that the method is better at forecasting extreme cases at the cost of robustness. The clearer cases, such as the CNN outperforming all alternatives in terms of both RMSE and MAE at the three month horizon, indicate that the method is both better at extreme cases and in general.

Selecting the best forecast method may, therefore, depend on the error measure considered.

Difference in Rankings at Different Horizons

The forecasting methods' accuracy also differ in terms of forecast horizon. For instance, the MLP method outperforms the benchmarks at the one and three month horizons, while the architecture is outperformed by our best benchmark methods at longer horizons. These results echo the findings in Makridakis and Hibon (2000) regarding differing horizons, and indicate that there may not exist a single forecast method which is superior for all forecast horizons.

The literature is ambiguous regarding neural network performance at different horizons. Nakamura (2005) finds that their neural network outperforms, or performs similar to, benchmarks up to the six month horizon, while performing worse at the twelve month horizon. Other studies (Moshiri & Cameron, 2000; Szafranek, 2017, e.g.) find that the predictive power of their included models is higher at the longer forecast horizons. These differences in the literature may be due to the use of different time series, with different dynamics, different methods for producing forecasts, e.g. the multi-neural vs. iterative approach (Makridakis et al., 2018b), or the implementation of slightly different architectures and network inputs. Our results, along with the literature, shows the importance of comparing forecasting methods at different time horizons. However, for neural networks, this can be time consuming. Thus, in this thesis we have

[32] This could generally be explained by the fact that our learning algorithm minimizes the MSE, which is also a quadratic loss function of the errors, similar to RMSE.

focused on the one month horizon when designing networks. The performance of our networks at the di erent horizons could, to some extent, be explained by this.

Our NN and MLP architectures are consistently good at forecasting the short term up to three months, and inferior at forecasting the longer horizons. These results are in line with literature on similar network architectures (Hall & Cook, 2017; Nakamura, 2005). A di erent speci cation in terms of network capacity or number of included lags could lead to networks which are better at forecasting longer horizons. The CNN network, however, seems to perform well at both short and longer term horizons. Especially relative to the NN and MLP architectures at the longer horizons, where the CNN is considerably more similar to the best benchmarks than the two other structures. Thus, under optimal conditions, the CNN architecture may be better at forecasting not only short term, but also for the medium to long term horizons. Therefore, the network structure is an interesting branch for further research.

Which Network Architecture is the Best at Forecasting Norwegian In ation?

We nd that the ubiquitous neural network architecture (NN), similar to the application in Makridakis et al. (2018b), gives marginal performance improvements over our benchmarks on the one and three month horizons. The MLP architecture, based on Hall and Cook (2017), improves upon the benchmarks at the one, three and six month horizons, and also improves upon the NN model at all horizons. The CNN model, also based on Hall and Cook (2017), improves noticeably over the benchmarks at the one, three and six month horizons, while also outperforming both the NN and MLP models at the three, six and twelve month horizons. Among the networks, overall performance is improved over the NN by introducing deeper networks with residual connections. The overall performance is further improved, especially at longer horizons, by applying a convolutional neural network structure.

However, as discussed, the CNN only performs well in the test period, while the ranking of MLP relative to the benchmarks is similar for both periods. So, which network is best at forecasting Norwegian in ation, MLP or CNN?

The MLP is robust and performs well at the short-term, both for the validation and test periods. Similarly, if we consider the accuracy of the alternative speci cations of the MLP architecture from which we chose the best performing network ensemble in the validation period (see Table 4 in Appendix A.2), all speci cations improve upon the best benchmarks at the one month horizon. This further indicates that the accuracy of MLP is robust to model speci cation.

The CNN is, however, the best network in the test period. Whatever the reasons underlying the di erence in validation and test performance, under optimal conditions the CNN seems to be a suitable forecasting method for Norwegian in ation. Therefore, the convolutional neural network structure could be interesting to apply in future research. Some of the underlying reasons for the instability in performance, as previously discussed, could quite easily be mitigated. The small

sample size is naturally reduced as time moves forward, making over tting less of a problem. Increasing the number of individual CNN networks in each repeat ensemble could further reduce potential random initialization stochasticity.

Thus, both the MLP and CNN networks can be applied to forecast in ation better than our benchmarks at some horizon. While the MLP seems to be more robust at forecasting the short term, the CNN network has the potential of also forecasting well at the medium to longer term horizons. Therefore, of our tested network architectures, the CNN network is the most interesting network structure for further research. But, does this mean that the neural networks are feasible macroforecasting tools?

7.3 Are Neural Networks Feasible Macroforecasting Tools?

Several papers argue that neural networks should be part of the macroforecasters toolbox (Gonzalez, 2000; Kuan, 2006). In our study, we nd that our neural networks generally outperform the random walk model and di erent ARIMA methods at some forecast horizon. These are meaningful benchmarks to beat, and are commonly applied to measure the accuracy of neural networks in the literature (e.g. Hall & Cook, 2017; Nakamura, 2005). Beating these benchmarks indicates that the small sample size of in ation, and similar macroeconomic series, at the monthly frequency is not a complete hindrance to the application of neural networks. Furthermore, we are able to improve upon a single hidden layer neural network through di erent architectural implementations. This indicates that expanding the scope of neural network structures could further improve accuracy. These ndings are generally positive as to the implementation of neural networks as a macroforecasting tool, and indicate that this could lead to potential forecasting gains.

At shorter forecasting horizons, several central banks apply forecast combinations based on large model suites. For instance, the system used by the Norwegian central bank (SAM[33]) consists of 140 di erent econometric approaches (Bj rnland et al., 2010). As such, an important real world test of relevancy would be whether the inclusion of these networks into this model suite improves performance. Recently, Makridakis et al. (2018a) found that the combination of statistical and machine learning methods generally improve forecast accuracy relative to both individual models and combinations of statistical or machine learning methods, separately. Speci cally for in ation, Szafranek (2017) nds similar results when combining such methods. At the same time, evidence suggests that neural networks are one of the best machine learning methods for forecasting time series (Ahmed, Atiya, Gayar, & El-Shishiny, 2010). Therefore, the inclusion of well designed networks with di erent speci cations could have the potential to improve the performance of large model suites, such as SAM. This is an interesting avenue for further research, and an indication that neural networks should, in fact, be part of the

[33] SAM: System of average models

macroforecasters toolbox.

However, the main aim of macroeconomic forecasting is to aid decision making which necessitates a certain level of interpretability in applied forecasting models. One of the main deterrents of the usefulness of these methods is the black box issue (Kuan, 2006). Implementing repeat ensembles, residual connections and deeper network structures further exacerbate the black-box issue. Thus, any gains in forecasting accuracy comes at the cost of inference.

7.4 Strengths and Limitations

Our thesis has a few key strengths. We compare the single hidden layer neural network to common time series methods for in ation used in the literature. In addition, we apply two deeper network structures, derived from networks performing well for US unemployment (Hall & Cook, 2017). We compare our three networks to benchmarks and to each other, providing insight about the predictive acuity of neural networks in general and the possibilities of improving performance through more complex network structures. We also apply time series cross-validation, and give ample time to designing performant networks. This approach to network design is similar to the modern approach in Chollet and Allaire (2018) for larger data problems, and is generally not applied in the literature. We also implement repeat ensembles both to mitigate random stochasticity and to improve forecast accuracy through forecast combination. We compare our results for both short term and long term forecasts, and for di erent error measures. We consider the forecasting performance of the validation and test periods, and discuss potential reasons why they di er. We decompose the error measures in order to investigate which periods the neural networks perform well in.

There are also some limitations in this thesis. We only considered a single macroeconomic series, in ation, and only univariate neural network and benchmark speci cations. This makes us less con dent in the real world accuracy of our networks, and their generalizability to other macroeconomic series. A general approach would be to create a larger battery of benchmarks from both the statistical and machine learning literature. Additionally, we only considered one pre-processing procedure, indicated by preliminary analysis, instead of producing results for di erent procedures. We also had several issues pertaining to computational e ciency. In this respect, our choice of the multi-neural approach to produce $h > 1$ step ahead forecasts may have been suboptimal. We believe that the iterative approach in Makridakis et al. (2018b) is a better trade-o between computational e ciency and multi-step forecasts. Because of our particular evaluation procedure, we unintentionally violated the underlying assumptions of common signi cance tests for equal predictive ability, such as the Giacomini and White test (Giacomini & White, 2006). Thus, we are unable to say anything about the statistical signi cance of the improvement in forecasting accuracy from applying neural networks. Our procedure for designing neural networks is also somewhat limited. To make the comparison

of di erent network structures feasible, for instance, we focus on the simplest network when deciding the number of individual models in each repeat ensemble. Due to time constraints, we only consider the one month horizon while designing networks.

7.5 Avenues for Further Research

Our ndings indicate that neural network forecasting performance can be improved through experimenting with network structure and network depth. Previous research has mainly focused on the single layer hidden neural network, and our results indicate that deeper networks with residual connections can improve upon this simple structure. An interesting branch of research is the implementation of more advanced network structures. A few interesting examples of such structures are the WaveNet architecture in Oord et al. (2016), based on convolutional neural networks with dilated kernels, or the Encoder-Decoder LSTM network in Hall and Cook (2017). Choudhary and Haider (2012) argue that neural networks should be continually compared to extant forecasting approaches in order to gauge its period by period performance. We would like to add to this continuous comparison the implementation of novel approaches from the deep neural network literature.

Focusing on multivariate speci cations of deeper network architectures for macroeconomic forecasting could potentially give gains in forecasting accuracy. For example, Hall and Cook (2017) argue that their networks could be improved through implementing exogenous variables important for the evolution of in ation.

Research into ways of arti cially increasing the sample size of macroeconomic time series could reduce the potential for over tting and may allow a larger set of di erent advanced network architectures to be applied. Bootstrapping, as used in Szafranek (2017) could be one alternative. Implementing di erent architectures into large model ensembles based on bootstrapping could improve performance. In the same vein, it would be interesting to see if techniques such as transfer learning[34] could be a feasible way of improving forecasting performance. Here, the problem of having a small data sample is circumvented by training the rst few layers of a deeper neural network structure on a di erent, high frequency, time series.

Based on recent developments, it would be very interesting to see if the combination of neural networks and extant forecast combination suites, such as those applied at central banks, improve forecasting accuracy.

[34] See e.g. Laptev, Yu, and Rajagopal (2018).

8 Conclusion

In this thesis we have gauged the feasibility of applying di erent neural network architectures to the task of forecasting the Norwegian macroeconomy. Focusing on in ation, we nd that all three of our network architectures outperform common linear benchmarks at one of the tested horizons. The forecasting improvements are generally found in times of high volatility. Furthermore, we nd that the single hidden layer neural network architecture, which has been the focus in large parts of the literature, is dominated by a deeper network architecture with residual connections. A deep convolutional neural network with residual connections is found to outperform the multi-layer perceptron architecture at the medium to long term horizons. Overall, the convolutional neural network is the best tested forecasting method for Norwegian in ation.

We nd that the performance of neural networks in forecasting Norwegian in ation is improved by giving ample time to network design, and through implementing techniques such as residual connections, dropout regularization and convolution. Though there currently exist barriers to the direct implementation of these methods in macroeconomic decision making, the potential gains in forecasting accuracy of including these methods in large statistical model suites, often applied by central banks, could make the methods part of a decision making tool in the future.

Also, the design and application of advanced deep neural network architectures has become signi cantly easier through the advent of simpli ed modelling tools. At the same time the surge in interest has demysti ed the underlying theoretical foundations leading to a signi cant decline in the level of expertise needed to practically implement such advanced forecasting methods.

Although neural networks may not currently be the complete answer to the problem of macroeconomic forecasting, the methods seem to be superior in forecasting certain periods. We therefore argue that neural networks should be continually compared to extant forecasting approaches. We believe these networks are relevant in forecasting macroeconomic time series, and that further research should be focused on the application of this particular technology. If ample time is given to network design, these types of networks could be applied to a large number of small data time series problems in an e ective manner with potential forecasting accuracy gains.

References

Ahmed, N. K., Atiya, A. F., Gayar, N. E., & El-Shishiny, H. (2010). An empirical comparison of machine learning models for time series forecasting. Econometric Reviews, 29(5-6), 594{621.

Allaire, J., & Chollet, F. (2018). keras: R interface to 'keras' [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=keras (R package version 2.1.5)

Allaire, J., Ushey, K., & Tang, Y. (2018). reticulate: Interface to 'python' [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=reticulate (R package version 1.6)

Bannert, M., & Thoeni, S. (2018). tstools: A time series toolbox for o cial statistics [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=tstools (R package version 0.3.6)

Bj rnland, H., Gerdrup, K., Jore, A. S., Smith, C., & Thorsrud, L. A. (2010). Does forecast combination improve norges bank in ation forecasts?

Bj rnland, H., & Thorsrud, L. A. (2015). Applied times series for macroeconomics, 2nd edition (No. 79-104). Gyldendal Akademisk.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional time series forecasting with convolutional neural networks. arXiv preprint arXiv:1703.04691.

Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123{140.

Breiman, L., et al. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). Statistical science, 16(3), 199{231.

Chakraborty, C., & Joseph, A. (2017). Machine learning at central banks.

Chollet, F., & Allaire, J. (2018). Deep learning with r. Manning Publications Co.

Chollet, F., et al. (2015). Keras. https://github.com/keras-team/keras . GitHub.

Choudhary, M. A., & Haider, A. (2012). Neural network models for in ation forecasting: an appraisal. Applied Economics, 44(20), 2631{2635.

Crone, S. F., Hibon, M., & Nikolopoulos, K. (2011). Advances in forecasting with neural networks? empirical evidence from the nn3 competition on time series prediction. International Journal of Forecasting, 27, 635{660.

Crone, S. F., & Pre mar, D. B. (2006). An extended evaluation framework for neural network publications in sales forecasting. In Arti cial intelligence and applications (pp. 179{186).

Faust, J., & Wright, J. H. (2013). Forecasting in ation. In Handbook of economic forecasting (Vol. 2, pp. 2{56). Elsevier.

Giacomini, R., & White, H. (2006). Tests of conditional predictive ability. Econometrica,

74(6), 1545{1578.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse recti er neural networks. In Proceedings of the fourteenth international conference on arti cial intelligence and statistics (pp. 315{323).

Goldberg, Y. (2016). A primer on neural network models for natural language processing. J. Artif. Intell. Res.(JAIR) , 57, 345{420.

Gonzalez, S. (2000). Neural networks for macroeconomic forecasting: a complementary approach to linear regression models. Department of Finance Canada.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning (Vol. 1). MIT press Cambridge.

Grolemund, G., & Wickham, H. (2011). Dates and times made easy with lubridate. Journal of Statistical Software, 40(3), 1{25. Retrieved from http://www.jstatsoft .org/v40/i03/

Hall, A. S., & Cook, T. R. (2017). Macroeconomic indicator forecasting with deep neural networks.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the ieee conference on computer vision and pattern recognition (pp. 770{778).

Hlavac, M. (2018). stargazer: Well-formatted regression and summary statistics tables [Computer software manual]. Bratislava, Slovakia. Retrieved from https://CRAN .R-project.org/package=stargazer (R package version 5.2.1)

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359{366.

Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: principles and practice, 2nd edition. OTexts. https://otexts.org/fpp2/ .

Hyndman, R. J., Khandakar, Y., et al. (2007). Automatic time series for forecasting: the forecast package for r (No. 6/07). Monash University, Department of Econometrics and Business Statistics.

Hyndman, R. J., O'Hara-Wild, M., Bergmeir, C., Razbash, S., Wang, E., & Hyndman, M. R. (2017). Package `forecast'. Online] https://cran. r-project. org/web/packages/forecast/forecast. pdf

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112). Springer.

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kuan, C.-M. (2006). Arti cial neural networks (IEAS Working Paper : academic research No. 06-A010). Institute of Economics, Academia Sinica, Taipei, Taiwan. Retrieved

from https://EconPapers.repec.org/RePEc:sin:wpaper:06-a010

Laptev, N., Yu, J., & Rajagopal, R. (2018). Applied timeseries transfer learning.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 3361(10), 1995.

Makridakis, S., & Hibon, M. (2000). The m3-competition: results, conclusions and implications. International Journal of Forecasting, 16, 451-476.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018a). The m4 competition: Results, findings, conclusion and way forward. International Journal of Forecasting.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018b). Statistical and machine learning forecasting methods: Concerns and ways forward. PloS one, 13(3), e0194889.

McAdam, P., & McNelis, P. (2005). Forecasting in ation with thick models and neural networks. Economic Modelling 22(5), 848{867.

Mhaskar, H., Liao, Q., & Poggio, T. (2016). Learning functions: when is deep better than shallow. arXiv preprint arXiv:1603.00988.

Moody, J., Levin, U., & Rehfuss, S. (1993). Predicting the us index of industrial production.

Moshiri, S., & Cameron, N. (2000). Neural network versus econometric models in forecasting in ation. Journal of forecasting 19(3), 201{217.

Nakamura, E. (2005). In ation forecasting using a neural network.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533.

Sax, C. (2017). seasonal: R interface to x-13-arima-seats [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=seasonal (R package version 1.6.1)

Shcherbakov, M. V., Brebels, A., Shcherbakova, N. L., Tyukov, A. P., Janovsky, T. A., & Kamaev, V. A. (2013). A survey of forecast error measures. World Applied Sciences Journal, 24, 171{176.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from over tting. Journal of machine learning research, 15(1), 1929{1958.

Stock, J. H., & Watson, M. W. (1998). A comparison of linear and nonlinear univariate models for forecasting macroeconomic time series (Tech. Rep.). National Bureau of

Economic Research.

Szafranek, K. (2017). Bagged arti cial neural networks in forecasting in ation.

Timmermann, A. (2006). Forecast combinations. Handbook of economic forecasting, 1, 135{196.

Tkacz, G., & Hu, S. (1999). Forecasting gdp growth using arti cial neural networks. Bank of Canada Ottawa.

Touretzky, D. S., & Pomerleau, D. A. (1989). What's hidden in the hidden layers. Byte, 14(8), 227{233.

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.

Wang, G., Ma, J., & Yang, S. (2014). An improved boosting based on feature selection for corporate bankruptcy prediction. Expert Systems with Applications, 41(5), 2353{2361.

Wickham, H. (2009). ggplot2: Elegant graphics for data analysis. Springer-Verlag New York. Retrieved from http://ggplot2.org

Woodford, M. (2005). Central bank communication and policy e ectiveness (Tech. Rep.). National Bureau of Economic Research.

Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with arti cial neural networks:: The state of the art. International journal of forecasting, 14(1), 35{62.

Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014). Time series classi cation using multi-channels deep convolutional neural networks. In International conference on web-age information management (pp. 298{310).

Appendices

Appendix A Network Tuning

This appendix outlines the process of tuning our networks. For each of our three network structures, we begin by reporting the cross-validation RMSE that results from di erent hyper-parameters. We then look at the convergence plots for the selected model architecture, for each of the years in the validation period and for each forecast horizon. Because all our "models" are actually model ensembles, averaging over several individual models for each network, we consider the*average*model convergence in order to monitor over tting. Speci cally, this is done by averaging the training/test loss over each epoch. To get an idea of the variability in test convergence, we also plot the interquartile range for each epoch, as well as the maximum and minimum values. Finally, we the out of sample forecasts generated by each of the networks.

A.1 Neural Network (NN)

Model#	H.Layers	Neurons	Dropout	RMSE
1	1	6	0	48:6
2	1	6	0.1	57.8
3	1	6	0.2	64.1
4	1	6	0.3	75.4
5	1	12	0	50.7
6	1	12	0.1	50.5
7	1	12	0.2	53.2
8	1	12	0.3	57.9
9	1	18	0	49.9
10	1	18	0.1	49.5
11	1	18	0.2	51.5
12	1	18	0.3	53.6
13	1	24	0	49.3
14	1	24	0.1	52.0
15	1	24	0.2	51.5
16	1	24	0.3	52.8

Table 3: Neural Network (NN) Validation: Di erent Architectures and related RMSE. H.Layers reports the number of hidden layers. For the neural network architecture this is xed to one. Neurons reports the number of computational neurons in the hidden layer. We coarsely increase the number of neurons from 6 to 24. Dropout reports the amount of dropout regularization that is applied. The dropout layer is placed after the hidden layer. RMSE reports the cross-validation error in the validation period, and is reported in basis points. The lowest RMSE is denoted in bold face.

A.1.1 Neural Network (NN): Network Architecture

Figure 14: Neural Network (NN) Architecture. Total trainable parameters: 445. The initial atten layer attens the [samples; timesteps; features] input structure utilized by Convolutional Neural Networks to a 2D tensor of the form [samples; timesteps features] which is compatible with the neural network.

A.1.2 Neural Network (NN): Validation Convergence

Figure 15: Neural Network (NN) Validation Convergence: one month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the one month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

Figure 16: Neural Network (NN) Validation Convergence: three month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the three month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

Figure 17: Neural Network (NN) Validation Convergence: six month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the six month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

Figure 18: Neural Network (NN) Validation Convergence: six month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the twelve month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

A.1.3 Neural Network (NN): Validation Forecasts

Figure 19: Neural Network (NN): Out of sample forecasts in the validation period. Starting from the top left: one month ahead forecasts, three month ahead forecasts, six month ahead forecasts and twelve month ahead forecasts. The blue line represents realized in ation. The red line represents our model ensemble forecast. The dark grey band around the red line represents the 75 and 25 percentile of individual model forecasts, while the light grey band represents the maximum and minimum individual model forecast. The dotted grey lines indicate the rst out of sample forecast made by a given model. For instance, the line coinciding with January 2002 represents the rst of twelve out of sample forecasts produced by the model trained on data until December 2001.

A.1.4 Neural Network (NN): Test Forecasts

Figure 20: Neural Network (NN): Out of sample forecasts in the test period. Starting from the top left: one month ahead forecasts, three month ahead forecasts, six month ahead forecasts and twelve month ahead forecasts. The blue line represents realized in ation. The red line represents our model ensemble forecast. The dark grey band around the red line represents the 75 and 25 percentile of individual model forecasts, while the light grey band represents the maximum and minimum individual model forecast.

A.2 Multi-Layer Perceptron (MLP)

Table (4) reports the cross-validation RMSE obtained by di erent multi-layer perceptron architectures.

Model	Neurons	Dropout	RMSE
1	8, 6, 4	0 X 3	48.1
2	8, 6, 4	0.1 X 3	47:2
3	8, 6, 4	0.2 X 3	47.5
4	8, 6, 4	0.3 X 3	47.8
5	16, 12, 8	0 X 3	48.6
6	16, 12, 8	0.1 X 3	48.5
7	16, 12, 8	0.2 X 3	48.6
8	16, 12, 8	0.3 X 3	47.9
9	24, 18, 12	0.0 X 3	50.6
10	24, 18, 12	0.1 X 3	50.0
11	24, 18, 12	0.2 X 3	49.7
12	24, 18, 12	0.3 X 3	48.3

Table 4: Multi-Layer Perceptron (MLP) Validation: Di erent architectures and related RMSE. Neurons reports the number of computational neurons in each of the dense stacks. Dropout reports the amount of dropout regularization that is applied at each dropout layer. The dropout layers are placed after each dense stack. RMSE reports the cross-validation error in the validation period, and is reported in basis points. The lowest RMSE is denoted in bold face.

A.2.1 Multi-Layer Perceptron (MLP): Architecture

Figure 21: Multi-layer Perceptron (MLP): Architecture. Total number of trainable parameters: 1.055. The initial atten layer attens the [samples; timesteps; features] input structure utilized by convolutional neural networks to a 2D tensor of the form [samples; timestepsxfeatures] which is compatible with the Multi-Layer Perceptron (MLP).

A.2.2 Multi-Layer Perceptron (MLP): Validation Convergence

Figure 22: Multi-layer Perceptron (MLP): Validation Convergence, one month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the one month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

Figure 23: Multi-layer Perceptron (MLP): Validation Convergence, three month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the three month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

Figure 24: Multi-layer Perceptron (MLP): Validation Convergence, six month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the six month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

Figure 25: Multi-layer Perceptron (MLP): Validation Convergence, twelve month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the twelve month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

A.2.3 Multi-Layer Perceptron (MLP): Validation Forecasts

Figure 26: Multi-layer Perceptron (MLP): Out of sample forecasts in the validation period. Starting from the top left: one month ahead forecasts, three month ahead forecasts, six month ahead forecasts and twelve month ahead forecasts. The blue line represents realized in ation. The red line represents our model ensemble forecast. The dark grey band around the red line represents the 75 and 25 percentile of individual model forecasts, while the light grey band represents the maximum and minimum individual model forecast. The dotted grey lines indicate the rst out of sample forecast made by a given model. For instance, the line coinciding with January 2002 represents the rst of twelve out of sample forecasts produced by the model trained on data until December 2001.

A.2.4 Multi-Layer Perceptron (MLP): Test Forecasts

Figure 27: Multi-Layer Perceptron (MLP): Out of sample forecasts in the test period. Starting from the top left: one month ahead forecasts, three month ahead forecasts, six month ahead forecasts and twelve month ahead forecasts. The blue line represents realized in ation. The red line represents our model ensemble forecast. The dark grey band around the red line represents the 75 and 25 percentile of individual model forecasts, while the light grey band represents the maximum and minimum individual model forecast.

A.3 Convolutional Neural Network (CNN)

Model#	Filters	K.Size	H.Nodes	Dropout	RMSE
1	6,6	3, 3	6, 4	0	0.539
2	6,6	3, 3	6, 4	0.1 X 2	0.544
3	6,6	3, 3	6, 4	0.2 X 2	0.529
4	6,6	3, 3	6, 4	0.3 X 2	0.545
5	12,12	6, 6	6, 4	0	0.529
6	12,12	6, 6	6, 4	0.1 X 2	0.527
7	12,12	6, 6	6, 4	0.2 X 2	0:519
8	12,12	6, 6	6, 4	0.3 X 2	0.520
9	16,16	8, 8	6, 4	0	0.540
10	16,16	8, 8	6, 4	0.1 X 2	0.531
11	16,16	8, 8	6, 4	0.2 X 2	0.529
12	16,16	8, 8	6, 4	0.3 X 3	0.526

Table 5: Convolutional Neural Network (CNN) Validation: Di erent architectures and related RMSE. Filters reports the number of lters in each convolutional layer. K.Size reports the kernel size of each convolutional layer. H.Layers reports the number of hidden layers. For the neural network architecture this is xed to one. Neurons reports the number of computational neurons in the hidden layer. Dropout reports the amount of dropout regularization that is applied. The dropout layer is situated after the hidden layer. RMSE reports the cross-validation error in the validation period, and is reported in basis points. The lowest RMSE is denoted in bold face.

A.3.1 Convolutional Neural Network (CNN): Architecture

Figure 28: Convolutional Neural Network (CNN): Architecture. Total trainable parameters: 3.191.

A.3.2 Convolutional Neural Network (CNN): Validation Convergence

Figure 29: Convolutional Neural Network (CNN) Validation: Validation Convergence, one month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the one month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

Figure 30: Convolutional Neural Network (CNN) Validation: Validation Convergence, three month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the three month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

Figure 31: Convolutional Neural Network (CNN) Validation: Validation Convergence, six month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the six month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

Figure 32: Convolutional Neural Network (CNN) Validation: Validation Convergence, twelve month horizon. Final validation convergence for each separate training and test set used for model validation. The network is trained to forecast the twelve month horizon. The blue line represents the average evolution of the training loss for each epoch. The red line represents the average evolution in the test loss for each epoch. The surrounding dark grey ll represent the 75 and 25 percentiles of test errors for given epoch, while the light grey ll represents the maximum and minimum test loss for individual models.

A.3.3 Convolutional Neural Network (CNN): Validation Forecasts

Figure 33: Convolutional Neural Network (CNN) Validation: Out of sample forecasts, validation period. Starting from the top left: one month ahead forecasts, three month ahead forecasts, six month ahead forecasts and twelve month ahead forecasts. The blue line represents realized in ation in the period January 2000 to December 2008. The red line represents our model ensemble forecast. The dark grey band around the red line represents the 75 and 25 percentile of individual model forecasts, while the light grey band represents the maximum and minimum individual model forecast. The dotted grey lines indicate the rst out of sample forecast made by a given model. For instance, the line coinciding with January 2002 represents the rst of twelve out of sample forecasts produced by the model trained on data until December 2001.

A.3.4 Convolutional Neural Network (CNN): Test Forecasts

Figure 34: Convolutional Neural Network (CNN): Out of sample forecasts in the test period. Starting from the top left: one month ahead forecasts, three month ahead forecasts, six month ahead forecasts and twelve month ahead forecasts. The blue line represents realized in ation. The red line represents our model ensemble forecast. The dark grey band around the red line represents the 75 and 25 percentile of individual model forecasts, while the light grey band represents the maximum and minimum individual model forecast.

Appendix B Benchmarks

B.1 Autoregressive Models (AR): Model Selection

	AR1	AR2	AR3	AR4	AR5	AR6	AR7	AR8	AR9	AR10	AR11	AR12
h = 1	0.535	**0.508**	0.509	0.512	0.513	0.519	0.519	0.510	0.510	0.513	0.524	0.525
h = 2	1.004	0.982	0.982	0.999	1.004	1.013	1.001	0.979	0.970	**0.966**	0.999	0.995
h = 3	1.191	1.188	1.191	1.197	1.195	1.205	1.193	1.182	**1.172**	1.186	1.217	1.212
h = 4	1.308	1.260	1.264	1.235	**1.230**	1.243	1.240	1.255	1.257	1.282	1.303	1.293

Table 6: Autoregressive Models with Lags From One to Twelve

B.2 Benchmark Coefficients in the Test Period

B.2.1 `auto.arima` Approach (AA): Coefficients in the Test Period

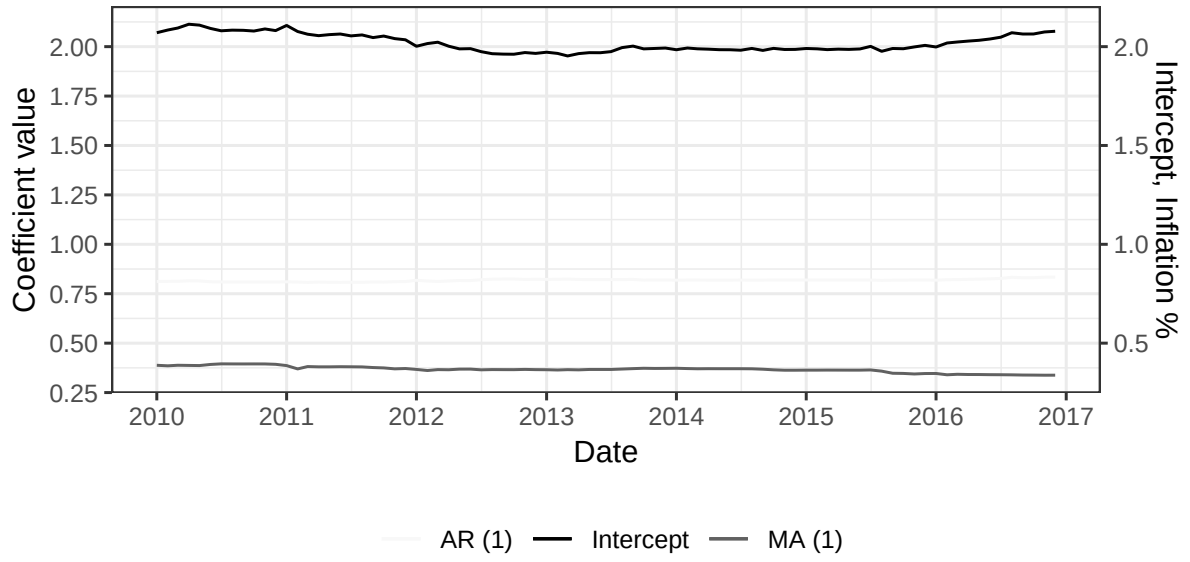

Figure 35: `auto.arima` co□cients in the test period, The AA method consistently picked an ARIMA(1,0,1) model. Left axis shows the value of the coe□cients given to each AR and MA term. Right axis shows intercept in % inflation.

B.2.2 Bayesian Information Criterion Approach and Autoregressive Model of Order 2 (ARBIC/AR2): Coefficients in the Test Period

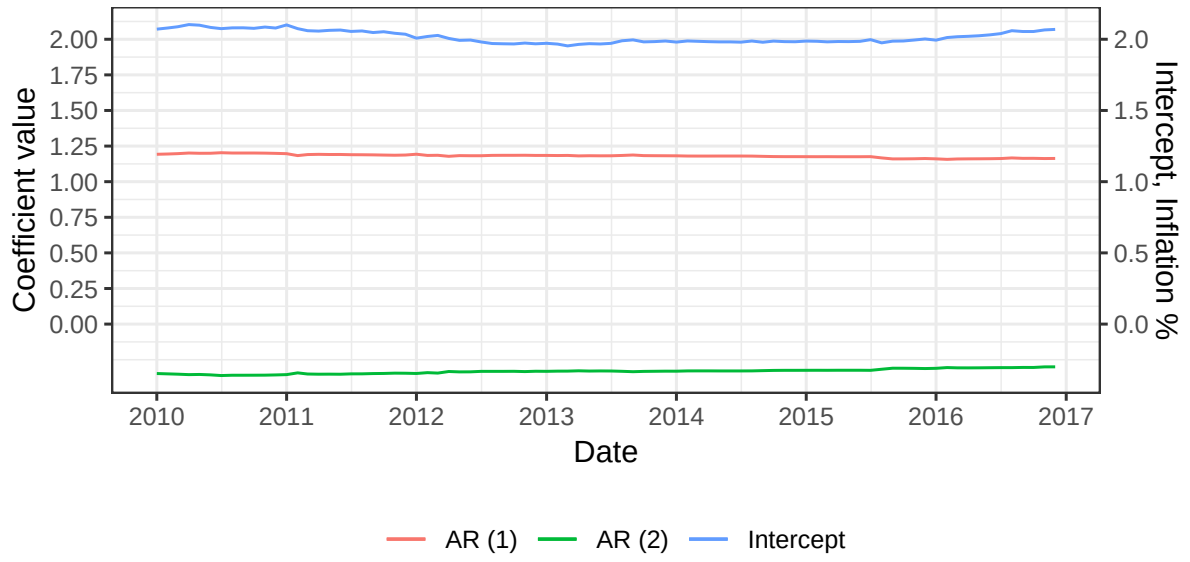

Figure 36: ARBIC and AR2 coe□cients in the test period. The ARBIC method consistently picked an AR(2) model. Thus, the plot also shows the coe□cients for AR2. Left axis shows the value of the coe□cients for each AR term. Right axis shows intercept in % inflation.

B.2.3 Akaike Information Criterion Approach: Coefficients in the Test Period

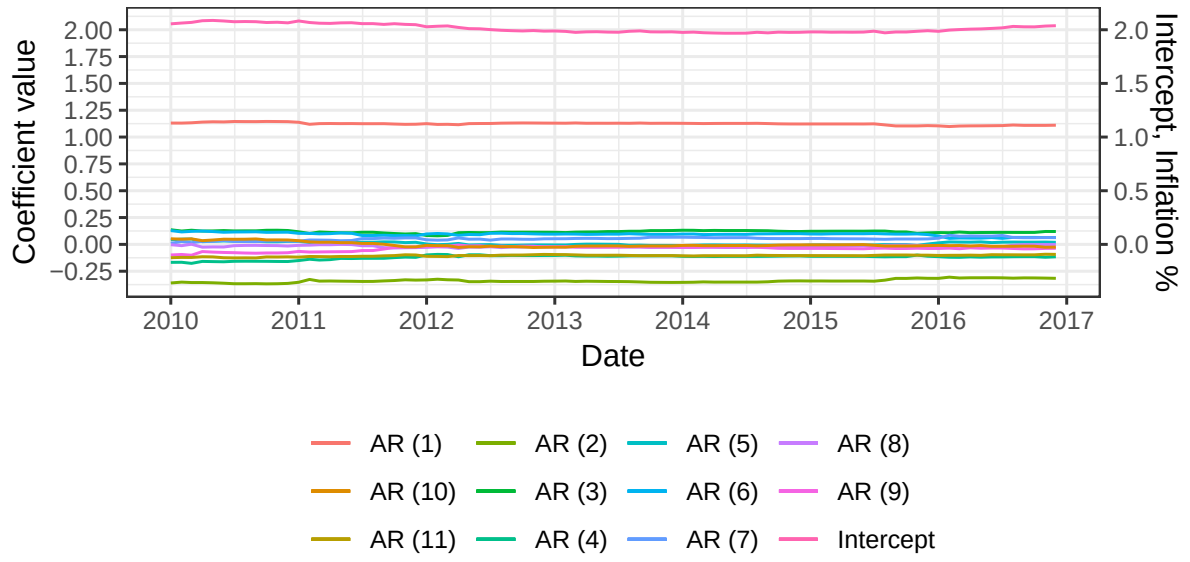

Figure 37: ARAIC coe□cients in the test period. The ARAIC method consistently picked an AR(11) model. Left axis shows the value of the coe□cients for each AR term. Right axis shows intercept in % inflation.

Appendix C Error Measure Decomposition

C.1 Yearly RMSE and MAE in the Validation Period (2000 - 2009)

	2000	2001	2002	2003	2004	2005	2006	2007	2008	Mean	Total
RW	0.284	0.400	0.473	0.876	0.846	0.164	**0.332**	0.764	0.810	0.550	0.608
AA	0.293	0.400	0.473	0.853	0.712	**0.150**	0.354	0.672	0.734	0.516	0.562
ARBIC	0.293	0.390	0.464	0.853	0.749	0.158	0.340	0.672	0.695	0.513	0.560
ARAIC	0.303	0.390	0.464	0.792	0.772	0.175	0.345	0.695	0.646	0.509	0.551
AR2	0.303	**0.382**	0.464	0.814	0.749	0.158	0.340	0.672	0.695	0.508	0.553
NN	0.248	0.404	0.385	0.810	0.628	0.563	0.355	**0.615**	0.498	0.501	0.526
MLP	**0.215**	0.422	**0.284**	**0.781**	**0.600**	0.346	0.413	0.629	0.459	**0.461**	**0.491**
CNN	0.327	0.421	0.356	0.976	0.721	0.382	0.408	0.688	**0.450**	0.525	0.565

	2000	2001	2002	2003	2004	2005	2006	2007	2008	Mean	Total
RW	0.236	0.337	0.382	0.644	0.548	**0.114**	0.262	0.561	0.578	0.407	0.407
AA	0.255	0.337	0.382	0.641	0.457	0.126	0.284	0.537	0.484	0.389	0.389
ARBIC	0.255	0.346	0.383	0.641	0.394	0.135	0.260	0.533	0.525	0.386	0.386
ARAIC	0.264	0.346	0.391	**0.540**	0.534	0.152	0.278	0.516	0.465	0.387	0.387
AR2	0.264	0.336	0.391	0.585	**0.394**	0.135	**0.260**	0.533	0.525	0.380	0.380
NN	0.223	0.354	0.336	0.586	0.516	0.364	0.284	0.414	0.389	0.385	0.385
MLP	**0.185**	0.330	**0.228**	0.546	0.469	0.274	0.284	**0.402**	**0.314**	**0.337**	**0.337**
CNN	0.289	**0.308**	0.305	0.764	0.619	0.303	0.295	0.458	0.346	0.410	0.410

Table 7: Yearly RMSE and MAE in the validation period (2000 - 2010) at the one month horizon. Top: RMSE, bottom: MAE. The table shows the error measures of each of our benchmark methods and neural networks calculated for each year in the validation period. The error measure reported in the results section is found under Mean. Best method for each year in **bold** face.

	2000	2001	2002	2003	2004	2005	2006	2007	2008	Mean	Total
RW	0.423	0.884	1.411	1.863	1.208	0.353	0.773	1.657	1.540	1.123	1.235
AA	0.509	0.884	1.411	1.763	0.799	**0.264**	0.671	1.416	1.544	1.029	1.140
ARBIC	0.509	0.810	1.340	1.763	0.749	0.277	0.668	1.387	1.357	0.984	1.089
ARAIC	0.562	0.810	1.314	1.710	1.135	0.338	**0.584**	1.318	1.091	0.985	1.069
AR2	0.562	0.788	1.314	1.737	**0.749**	0.277	0.668	1.387	1.357	0.982	1.082
NN	0.398	0.749	**1.002**	1.964	1.236	1.073	0.659	1.269	0.972	1.036	1.118
MLP	**0.375**	0.836	1.022	**1.705**	0.945	0.758	0.790	1.298	**0.813**	**0.949**	**1.013**
CNN	0.419	**0.712**	1.121	2.087	1.306	0.968	0.764	**1.255**	0.955	1.065	1.155

	2000	2001	2002	2003	2004	2005	2006	2007	2008	Mean	Total
RW	0.362	0.784	1.111	1.500	0.824	0.277	0.635	1.230	1.246	0.885	0.885
AA	0.398	0.784	1.111	1.383	0.669	**0.198**	0.539	1.253	1.192	0.836	0.836
ARBIC	0.398	0.762	1.015	1.383	0.630	0.207	0.545	1.229	1.111	0.809	0.809
ARAIC	0.445	0.762	1.002	1.285	0.950	0.266	**0.480**	1.133	0.854	0.797	0.797
AR2	0.445	0.735	1.002	1.365	**0.630**	0.207	0.545	1.229	1.111	0.808	0.808
NN	0.325	0.666	0.728	1.419	0.943	0.707	0.584	0.997	0.828	0.800	0.800
MLP	**0.312**	0.704	**0.721**	**1.240**	0.797	0.653	0.674	**0.945**	**0.689**	**0.748**	**0.748**
CNN	0.369	**0.630**	0.730	1.586	1.101	0.847	0.653	0.985	0.781	0.854	0.854

Table 8: Yearly RMSE and MAE in the validation period (2000 - 2010) at the three month horizon. Top: RMSE, bottom: MAE. The table shows the error measures of each of our benchmark methods and neural networks calculated for each year in the validation period. The error measure reported in the results section is found under Mean. Best method for each year in **bold** face.

	2000	2001	2002	2003	2004	2005	2006	2007	2008	Mean	Total
RW	**0.529**	1.463	1.870	2.238	1.345	0.568	1.110	2.204	1.664	1.444	1.561
AA	0.910	1.463	1.870	2.052	0.697	0.318	0.969	1.602	1.731	1.290	1.405
ARBIC	0.910	1.179	1.551	2.052	0.631	0.317	0.954	1.547	1.573	1.190	1.296
ARAIC	1.026	1.179	1.486	**1.879**	1.474	0.332	0.909	1.431	1.182	1.211	**1.280**
AR2	1.026	**1.124**	1.486	2.036	**0.631**	**0.317**	0.954	1.547	1.573	**1.188**	1.288
NN	0.708	1.280	**1.117**	2.495	1.468	1.569	**0.889**	1.503	1.176	1.356	1.440
MLP	0.576	1.341	1.255	2.364	1.289	1.036	1.040	1.654	**0.906**	1.273	1.361
CNN	0.733	1.343	1.143	2.803	1.403	1.240	0.974	1.521	1.475	1.404	1.507

	2000	2001	2002	2003	2004	2005	2006	2007	2008	Mean	Total
RW	**0.442**	1.369	1.496	1.937	0.862	0.525	0.803	1.905	1.400	1.193	1.193
AA	0.856	1.369	1.496	1.760	0.653	**0.248**	0.735	1.493	1.455	1.118	1.118
ARBIC	0.856	1.071	1.153	1.760	0.597	0.250	0.733	1.446	1.163	1.003	1.003
ARAIC	0.974	1.071	1.109	**1.603**	1.277	0.300	**0.669**	1.351	0.819	1.019	1.019
AR2	0.974	**1.013**	1.109	1.718	**0.597**	0.250	0.733	1.446	1.163	**1.000**	**1.000**
NN	0.631	1.150	**0.856**	2.001	1.097	1.310	0.720	1.359	1.041	1.129	1.129
MLP	0.500	1.219	0.988	1.817	1.036	0.889	0.878	1.426	**0.733**	1.054	1.054
CNN	0.683	1.229	0.972	2.233	1.189	0.963	0.839	**1.262**	1.266	1.182	1.182

Table 9: Yearly RMSE and MAE in the validation period (2000 - 2010) at the six month horizon. Top: RMSE, bottom: MAE. The table shows the error measures of each of our benchmark methods and neural networks calculated for each year in the validation period. The error measure reported in the results section is found under Mean. Best method for each year in **bold** face.

	2000	2001	2002	2003	2004	2005	2006	2007	2008	Mean	Total
RW	0.947	2.175	2.000	3.173	1.376	0.891	1.827	3.369	2.070	1.981	2.144
AA	1.035	2.175	2.000	2.419	0.670	0.408	1.549	2.031	2.120	1.601	1.741
ARBIC	1.035	1.455	1.409	2.419	0.639	0.383	1.529	1.952	0.713	1.282	1.424
ARAIC	1.097	1.455	1.352	**1.972**	1.596	**0.308**	1.462	1.596	0.813	1.295	**1.375**
AR2	1.097	**1.353**	1.352	2.326	**0.639**	0.383	1.529	1.952	**0.713**	**1.260**	1.394
NN	**0.922**	2.026	1.295	3.560	0.991	2.117	1.393	2.114	1.387	1.756	1.918
MLP	0.926	2.004	1.418	3.225	1.062	1.521	1.243	2.217	0.949	1.618	1.766
CNN	1.262	1.989	**1.058**	3.207	1.237	1.629	**1.203**	2.167	1.252	1.667	1.790

	2000	2001	2002	2003	2004	2005	2006	2007	2008	Mean	Total
RW	0.837	1.868	1.544	2.206	1.054	0.814	1.659	3.068	1.538	1.621	1.621
AA	0.874	1.868	1.544	1.932	0.614	0.339	1.399	1.836	1.574	1.331	1.331
ARBIC	0.874	1.257	0.988	1.932	0.584	0.323	1.379	1.763	0.579	1.075	1.075
ARAIC	0.914	1.257	0.944	**1.637**	1.492	**0.269**	1.315	**1.446**	0.635	1.101	1.101
AR2	0.914	**1.168**	0.944	1.887	**0.584**	0.323	1.379	1.763	**0.579**	**1.060**	**1.060**
NN	**0.785**	1.696	1.055	2.584	0.867	1.885	1.212	1.883	1.065	1.448	1.448
MLP	0.800	1.722	1.110	2.216	0.847	1.436	**1.032**	2.016	0.772	1.328	1.328
CNN	1.117	1.876	**0.858**	2.746	1.183	1.530	1.121	2.042	1.161	1.515	1.515

Table 10: Yearly RMSE and MAE in the validation period (2000 - 2010) at the twelve month horizon. Top: RMSE, bottom: MAE. The table shows the error measures of each of our benchmark methods and neural networks calculated for each year in the validation period. The error measure reported in the results section is found under Mean. Best method for each year in **bold** face.

C.2 Yearly RMSE and MAE in the Test Period (2010 - 2017)

	2010	2011	2012	2013	2014	2015	2016	Mean	Total
RW	0.424	0.460	0.353	0.372	**0.171**	0.349	0.305	0.348	0.358
AA	0.382	0.487	0.331	0.324	0.213	0.383	0.372	0.356	0.364
ARBIC	0.361	0.479	0.364	0.340	0.203	0.377	0.383	0.358	0.366
ARAIC	0.429	0.473	0.361	0.290	0.230	0.386	0.392	0.366	0.374
AR2	**0.361**	0.479	0.364	0.340	0.203	0.377	0.383	0.358	0.366
NN	0.467	0.384	**0.289**	0.274	0.268	0.322	0.292	0.328	0.335
MLP	0.394	**0.383**	0.300	0.293	0.275	**0.320**	0.295	**0.323**	**0.326**
CNN	0.386	0.388	**0.289**	0.296	0.277	0.346	**0.280**	**0.323**	**0.326**

	2010	2011	2012	2013	2014	2015	2016	Mean	Total
RW	0.343	0.380	0.315	0.310	**0.147**	0.260	0.225	0.283	0.283
AA	0.318	0.389	0.271	0.266	0.174	0.284	0.289	0.284	0.284
ARBIC	0.305	0.380	0.295	0.288	0.164	0.278	0.302	0.287	0.287
ARAIC	0.344	0.371	0.294	0.237	0.186	0.280	0.326	0.291	0.291
AR2	**0.305**	0.380	0.295	0.288	0.164	0.278	0.302	0.287	0.287
NN	0.396	0.316	0.241	0.220	0.186	**0.259**	**0.222**	0.263	0.263
MLP	0.306	0.323	**0.236**	0.238	0.176	0.261	0.234	**0.253**	**0.253**
CNN	0.334	**0.314**	0.253	0.263	0.160	0.284	0.230	0.262	0.262

Table 11: Yearly RMSE and MAE in the test period (2010 - 2017) at the one month horizon. Top: RMSE, bottom: MAE. The table shows the error measures of each of our benchmark methods and neural networks calculated for each year in the testing period. The error measure reported in the results section is found under Total. Best method for each year in **bold** face.

	2010	2011	2012	2013	2014	2015	2016	Mean	Total
RW	0.831	0.743	0.568	0.666	0.188	0.461	0.564	0.574	0.606
AA	0.678	0.802	0.652	0.526	0.165	0.487	0.742	0.579	0.612
ARBIC	0.644	0.838	0.732	0.508	0.162	0.496	0.794	0.596	0.634
ARAIC	0.734	0.751	0.820	0.360	0.318	0.501	0.781	0.609	0.640
AR2	0.644	0.838	0.732	0.508	**0.162**	0.496	0.794	0.596	0.634
NN	0.964	0.664	**0.443**	0.512	0.379	0.462	0.547	0.567	0.596
MLP	0.712	0.701	0.527	0.450	0.428	**0.438**	0.533	0.541	0.553
CNN	**0.473**	**0.618**	0.549	**0.321**	0.462	0.557	**0.438**	**0.488**	**0.496**

	2010	2011	2012	2013	2014	2015	2016	Mean	Total
RW	0.676	0.554	0.488	0.568	0.149	0.395	0.452	0.469	0.469
AA	0.533	0.582	0.533	0.400	0.142	0.406	0.650	0.464	0.464
ARBIC	0.506	0.635	0.578	0.388	0.140	0.416	0.691	0.479	0.479
ARAIC	0.612	0.583	0.688	0.316	0.266	0.415	0.672	0.507	0.507
AR2	0.506	0.635	0.578	0.388	**0.140**	0.416	0.691	0.479	0.479
NN	0.822	**0.524**	**0.368**	0.402	0.312	0.401	0.479	0.473	0.473
MLP	0.586	0.622	0.469	0.350	0.351	**0.377**	0.441	0.457	0.457
CNN	**0.423**	0.542	0.474	**0.282**	**0.322**	0.480	0.342	**0.409**	**0.409**

Table 12: Yearly RMSE and MAE in the test period (2010 - 2017) at the three month horizon. Top: RMSE, bottom: MAE. The table shows the error measures of each of our benchmark methods and neural networks calculated for each year in the testing period. The error measure reported in the results section is found under Total. Best method for each year in **bold** face.

	2010	2011	2012	2013	2014	2015	2016	Mean	Total
RW	0.807	**0.755**	0.791	0.959	0.173	0.766	0.993	0.749	0.790
AA	0.602	1.016	0.776	0.628	0.109	0.819	1.005	0.708	0.764
ARBIC	0.596	1.079	0.897	0.583	0.104	0.828	1.070	0.737	0.802
ARAIC	**0.474**	1.092	1.008	0.449	0.273	0.843	0.988	0.732	0.792
AR2	0.596	1.079	0.897	0.583	**0.104**	0.828	1.070	0.737	0.802
NN	1.009	0.987	**0.577**	0.819	0.551	0.764	0.782	0.784	0.801
MLP	0.872	0.887	0.669	0.724	0.497	**0.720**	**0.741**	0.730	0.740
CNN	0.540	0.886	0.706	0.471	0.482	0.804	0.767	**0.665**	**0.683**

	2010	2011	2012	2013	2014	2015	2016	Mean	Total
RW	0.716	**0.626**	0.724	0.798	0.142	0.628	0.850	0.640	0.640
AA	0.516	0.908	0.563	0.452	0.086	0.678	0.833	**0.576**	**0.576**
ARBIC	0.494	0.976	0.678	0.394	0.085	0.688	0.886	0.600	0.600
ARAIC	**0.411**	0.973	0.881	**0.390**	0.222	0.712	0.802	0.627	0.627
AR2	0.494	0.976	0.678	0.394	**0.085**	0.688	0.886	0.600	0.600
NN	0.749	0.944	**0.475**	0.657	0.490	0.645	0.665	0.661	0.661
MLP	0.764	0.836	0.546	0.592	0.450	**0.625**	**0.612**	0.632	0.632
CNN	0.468	0.870	0.557	0.401	0.444	0.737	0.624	0.586	0.586

Table 13: Yearly RMSE and MAE in the test period (2010 - 2017) at the six month horizon. Top: RMSE, bottom: MAE. The table shows the error measures of each of our benchmark methods and neural networks calculated for each year in the testing period. The error measure reported in the results section is found under Total. Best method for each year in **bold** face.

	2010	2011	2012	2013	2014	2015	2016	Mean	Total
RW	1.148	**1.149**	1.665	0.680	0.365	**1.368**	1.748	1.161	1.250
AA	0.785	1.481	0.684	0.178	0.324	1.450	0.780	0.812	0.935
ARBIC	0.758	1.513	0.656	0.153	0.323	1.459	0.741	0.800	0.933
ARAIC	**0.636**	1.725	0.848	0.379	0.368	1.503	**0.691**	0.878	1.008
AR2	0.758	1.513	0.656	**0.153**	**0.323**	1.459	0.741	**0.800**	**0.933**
NN	1.073	1.498	1.222	0.983	0.505	1.427	1.058	1.109	1.151
MLP	1.363	1.543	0.887	0.612	0.373	1.416	0.803	1.000	1.081
CNN	1.127	1.292	**0.476**	0.490	0.487	1.419	0.877	0.881	0.958

	2010	2011	2012	2013	2014	2015	2016	Mean	Total
RW	0.952	**0.916**	1.486	0.570	0.263	**1.265**	1.559	1.001	1.001
AA	0.734	1.426	0.562	0.140	**0.215**	1.387	0.663	0.733	0.733
ARBIC	0.712	1.464	0.557	0.130	0.217	1.398	0.629	0.730	0.730
ARAIC	**0.593**	1.688	0.708	0.325	0.299	1.439	**0.504**	0.794	0.794
AR2	0.712	1.464	0.557	**0.130**	0.217	1.398	0.629	**0.730**	**0.730**
NN	0.862	1.413	1.024	0.940	0.424	1.332	0.894	0.984	0.984
MLP	1.237	1.452	0.722	0.578	0.316	1.344	0.665	0.902	0.902
CNN	1.064	1.215	**0.387**	0.387	0.391	1.355	0.786	0.798	0.798

Table 14: Yearly RMSE and MAE in the test period (2010 - 2017) at the twelve month horizon. Top: RMSE, bottom: MAE. The table shows the error measures of each of our benchmark methods and neural networks calculated for each year in the testing period. The error measure reported in the results section is found under Total. Best method for each year in **bold** face.

Appendix D Norwegian Inflation Series

D.1 Inflation Series 1921 - 2017

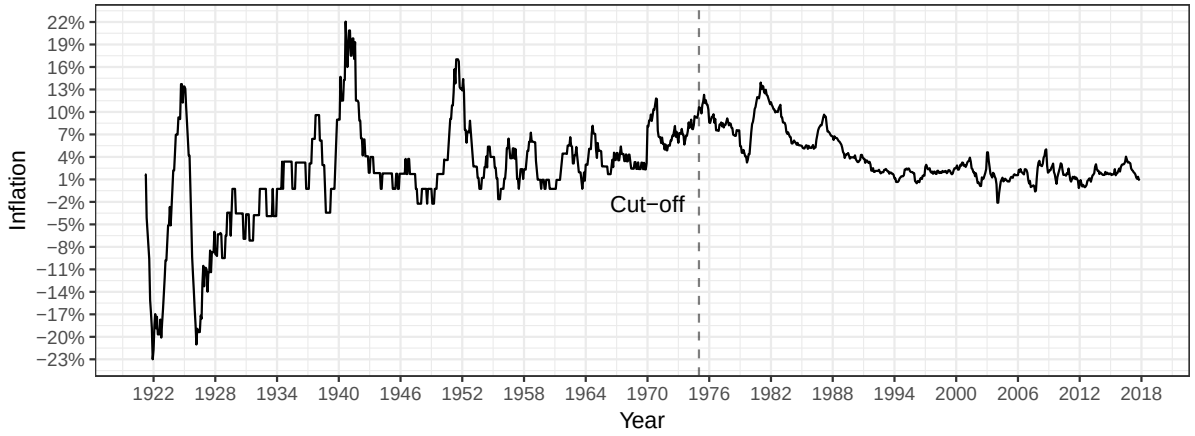

Figure 38: Monthly Y-o-Y growth in Norwegian consumer price index from 1921-2017. The vertical line in 1975 indicates a possible cut-o point where data quality is feasible for our approach. We lose 12 months of observations in the start of the series due to the year on year transformation.

D.2 Inflation Series 1993 - 2017

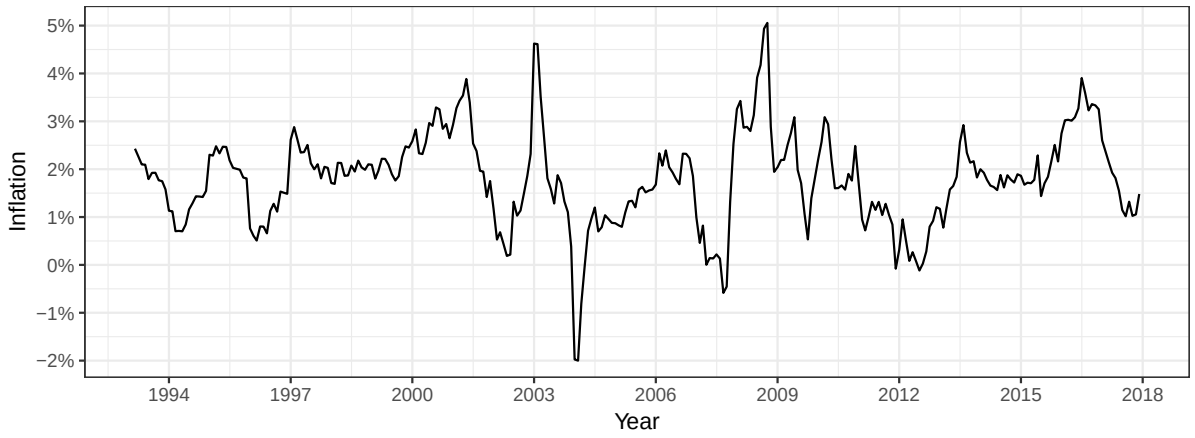

Figure 39: The figure depicts the monthly Y-o-Y growth in Norwegian consumer price index from 1993-2017. This is the series used for training our ARIMA models.

D.3 First Differenced Inflation Series 1975 - 2017

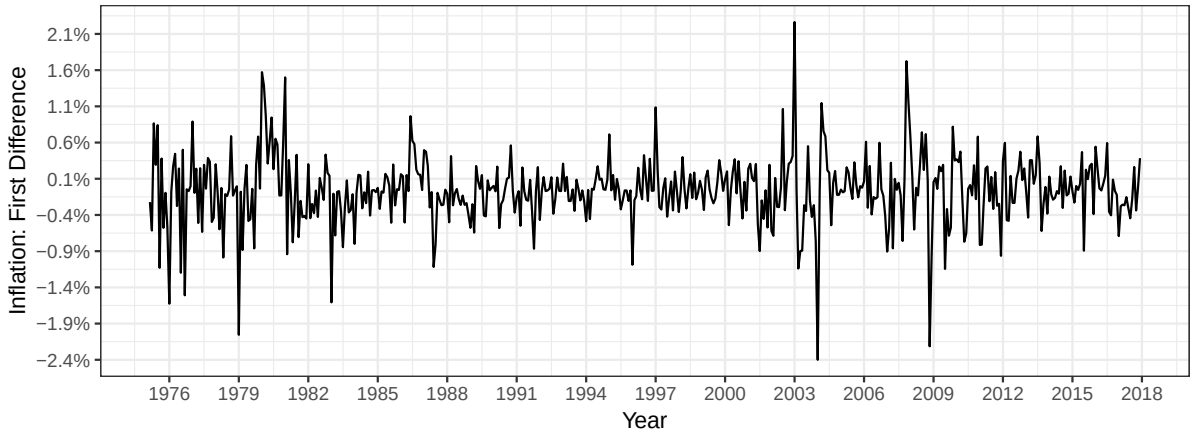

Figure 40: Norwegian Inflation 1975-2017: First Di erence.

D.4 Second Differenced Inflation Series 1975 - 2017

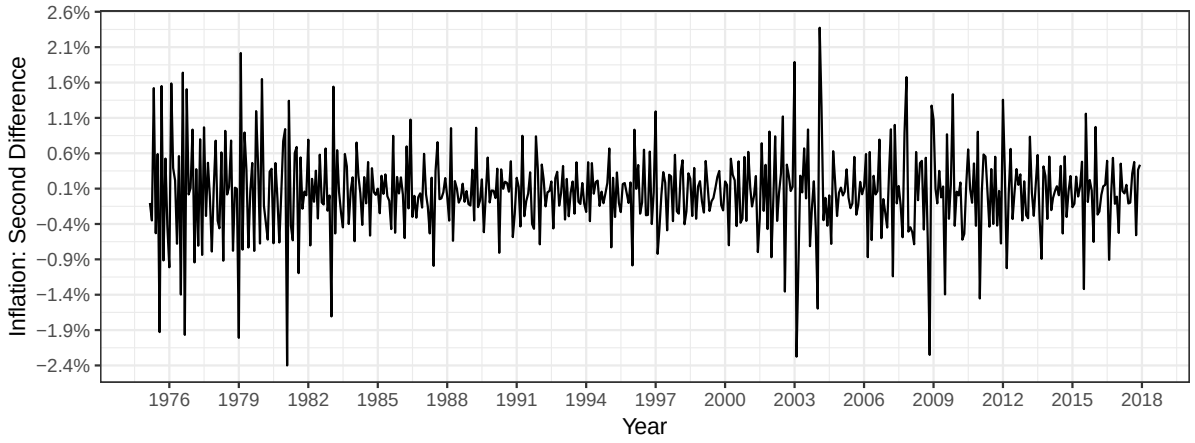

Figure 41: Norwegian Inflation 1975-2017: Second Di erence.

D.5 Summary Statistics for the validation and test periods

Figure 42 shows the validation period (red) and the testing period (blue).

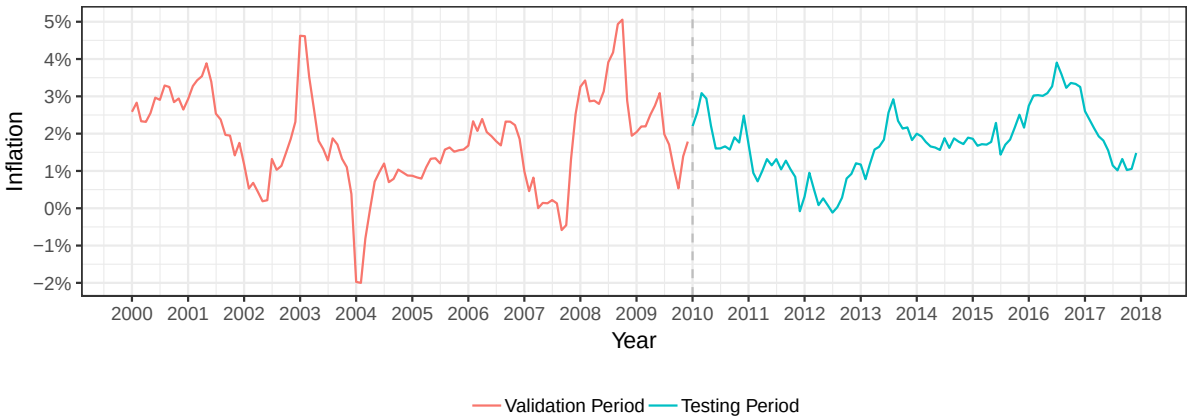

Figure 42: The inflation series for the validation and test periods. The period from 2000 - 2010, colored red, is the period used for network design. The period from 2010 - 2017, colored in blue, is used as a final testing set.

Table 15 reports the mean, standard deviation and min-max values of the Norwegian inflation series for the validation and test periods. While the mean is similar between the two periods, we see that the standard deviation is 44% higher. The validation period also has a deeper negative range, and a slightly higher positive range.

	Validation Period	Testing Period
μ	2.06	2.00
σ	1.27	0.88
Min	-1.75	0.13
Max	5.30	4.15

Table 15: Summary Statistics of the Validation and Test Periods.