Time-series predictions with Recurrent Neural Networks

Studying Recurrent Neural Networks
predictions and comparing to state-of-the-art
Sequential Monte Carlo methods

Lene Finsveen

Master of Science in Physics and Mathematics
Submission date: June 2018
Supervisor: Thiago Martins, IMF

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

Recurrent Neural Networks (RNNs) have shown great success in sequence-to-sequence processing due to its ability to retain memory while incrementally processing sequence elements. It has become a fundamental algorithm for processing text and speech, and is recently becoming more popular on time-series prediction as well. Recent blog posts showing RNN flexibility applied to time-series prediction combined with flexible and easy-to-use APIs such as Keras, are leading to a widespread use of RNN for time-series prediction.

The aim of this thesis is to study the performance of RNN to predict time-series under different scenarios of noise and stationarity. In order to do this, we will simulate time-series from a state-space model (SSM) with known noise and stationarity parameters. An SSM is a subclass of a Bayesian hierarchical model and simulates a hidden state and observation depending only on static parameters and state. This is a common setup found in the signal processing literature. The hidden state will act as an underlying signal while the observation is a mixture of signal and noise. RNN models will try to predict the signal based on the simulated observed values. We are going to experiment with a long short-term memory (LSTM) model, which is the most popular type of RNNs currently used.

We evaluate the quality of the LSTM predictions by computing the cumulative mean square error (CMSE) with respect to the true simulated signal. In addition, we will estimate the signal using state-of-the-art Sequential Monte Carlo methods (SMC) such as the Bootstrap filter and Particle Markov Chain Monte Carlo (PMCMC). We then also compare the performance of the LSTM models with respect to the estimated signals. The objective is to get insights into the LSTM performance level when compared to custom-tailored state-space models.

The general findings are that the LSTM models perform worse than custom-tailored SSM models in non-stationary scenarios. PMCMC outperforms LSTM in all scenarios given the custom-tailored SSM parameters. Bootstrap filtering struggles more with estimating the state in non-stationary scenarios as noise increases, but still perform better than LSTM.

Sammendrag

Rekursive Nevrale Nettverk (RNN) har vist seg å gi gode resultater i prosessering av sekvens-data på grunn av algoritmens evne til å opprettholde internt minne samtidig som elementene i sekvensen prosesseres. RNN har blitt en fundamental algoritme for prosessering av text og tale, og har nylig blitt mer populær innen tidsrekkedata prediksjon. Nylige bloggartikler vitner om at bruken av RNN er blitt mer utbredt ettersom RNN er fleksibel for tidsrekker og APIer som Keras gjør det lett å bruke.

Målet for denne masteroppgaven er å studere RNNs utførelsesnivå for prediksjon av tidsrekker under forskjellige scenarier av støy og stasjonæritet. For å gjøre dette vil vi simulere tidsrekkedata fra en "state-space" modell (SSM) med kjent støy og stasjonæritet. SSM simulerer en ukjent tilstand og observasjon kun avhengig av de statiske parametrene og tilstanden. I litteratur er dette oppsettet mer kjent som signalprosessering. Den ukjente tilstanden vil være et signal og observasjonene er en blanding av signal og støy. RNN modeller skal predikere signalet basert på de kjente observasjonene. Modellen i eksperimentet vil være "long short-term memory" (LSTM) modellen, som er den mest poplære typen RNN.

For å vurdere prediksjonkvaliteten på LSTM prediksjonene beregner vi den kumulative "mean square error"-metrikken (CMSE) for signalet. I tillegg vil to nymoderne "sequential Monte Carlo" metoder estimere signalet, Bootstrap filter og particle Markov Chain Monte Carlo (PMCMC). Utførelsen til alle LSTM modeller vil også sammenlignes mot hverandre basert på signalestimering. Målet er å få innsikt i utførelsesnivået til LSTM når sammenlignet med spesialdesignede SSM metoder.

Resultatene av eksperimentet viser at LSTM modellene estimerer signalet dårligere enn SMC modellene i ustasjonære scenarier. PMCMC utklasser LSTM i alle scenarier, ikke overraskende siden den er spesialtilpasset for simuleringsdataene. Bootstrap filteret estimerer signalet dårligere i de ustasjonære scenariene når støyet øker, men fremdeles bedre enn LSTM.

Preface

Recurrent Neural Networks (RNNs) have become a fundamental algorithm for processing text, speech, time-series - in general, sequence-to-sequence processing. The aim of this thesis is to study the performance of RNN to predict time-series under different scenarios of noise and stationarity. To do this, we will simulate time-series from a state-space model (SSM) with known parameters. SSM simulates a hidden state (a signal) and observations which is a mixture of signal and noise. This setup is found in signal processing literature and the RNN model will try to predict the signal based on observed values. To evaluate the model performance state-of-the-art sequential Monte Carlo (SMC) methods will also estimate the signal and the objective is to get insights into the RNN performance level.

This thesis concludes my master of Technology in Applied Physics and Mathematics with specialization in Industrial Mathematics at Norwegian University of Science and Technology (NTNU). The work was carried out during the spring semester of 2018 at the Department of Mathematical Science under the supervision of Thiago Guerrera Martins.

My sincere appreciation goes to Thiago Guerrera Martins for guidance and contributions during the process of writing the thesis. He has contributed with motivation and experience on the topic and showed great patience during the work that has been carried out. Lastly, he has ensured steady progress during the semester and provided valuable feedback towards the end.

Lene Finsveen
June 18, 2018
Trondheim, Norway

Table of Contents

Chapter 1

Introduction

Motivation

Natural Language Processing (NLP) has benefitted greatly from deep neural networks (Chowdhury, 2005). NLP is an area for exploring how text and speech can be processed by computers in order to perform useful tasks. This involves text processing for a variety of applications, e.g. language modeling, handwritten recognition, text classification and image recognition (Karpathy, 2015; Graves, 2013; Chollet and Allaire, 2018; Ba et al., 2014).

Recurrent Neural Networks (RNN) have in particular shown its success in text and sequence processing and become a fundamental algorithm in this area (Chollet and Allaire, 2018). Text, words or time-series can be represented in sequences, i.e. it is a sequence of single elements (letters, words, data-points). The ability to process elements incrementally while maintaining a memory is what makes RNN great for sequence-to-sequence processing. Chollet and Allaire (2018) elegantly demonstrates that a basic RNN can be easily setup using Keras (Chollet, 2015). This simplicity and flexibility are some of the reasons for its widespread use. Recent blog posts, e.g. Brownlee (2017) also suggest RNNs are growing popular for time-series and real-time data predictions.

This thesis aims to study the performance of RNN to do signal processing. A long short-term memory (LSTM) model is employed to make time-series predictions. The task of LSTM is to capture the signal under different scenarios of stationarity and noise. The signal and observations are simulated from a state-space model (SSM) with known stationarity and noise parameters. To evaluate the RNN state-of-the-art sequential Monte Carlo (SMC) methods will also estimate the signal. Comparison between the LSTM predictions and SMC signal estimation will shed light on the RNN model performance level under different scenarios.

Outline

In Chapter 2 we give background on RNN and previous work on this area. We present a common use of RNN and examples of known applications of LSTM. In Chapter 3, we define SSM and add a section presenting the model parameters and the resulting simulated datasets. Chapter 4 presents RNNs. It contains model definition, experimental setup and results obtained with the LSTM models. Chapter 5 introduces SMC; its definition and practical use. For this method sampling and filtering is given, as well as proposed methods and experimental setup. For the final chapter, Chapter 6, we present our overall findings. In particular a comparison of the methods and an evaluation of LSTM. Lastly, thoughts on the experiment and future work are given.

Chapter 2

Literature Review

RNNs have recently demonstrated great success in sequence forecasting, e.g. language modeling (Karpathy, 2015), financial markets (Cavalcante et al., 2016), speech and handwriting recognition (Graves, 2013), image captioning (Ba et al., 2014) and other real world time-series problems.

A long short-term memory (LSTM) network was first introduced by Hochreiter and Schmidhuber (1997) and is RNN with an appropriate gradient-based learning and more flexible memory. LSTM was introduced to address the constant error backdrop, more known as the vanishing gradient problem (Hochreiter, 1991). Hochreiter and Schmidhuber (1997) showed LSTM could solve complex, artificial long time lags never before solved by RNN and Gers et al. (2001) analyzed the use of LSTM on conceptually simpler time-series problems. It suggests LSTM is only to use when simpler traditional methods fail.

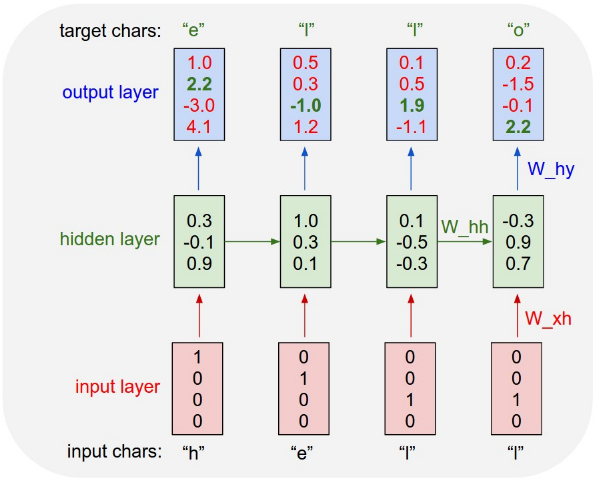

Figure 2.1: An illustration of text being processed by RNN as a sequence of characters (Karpathy, 2015).

Karpathy (2015) shows in a blog post how RNN classifies characters from text by producing the probability of the next character in the sequence. Figure 2.1 shows the input vector ["h", "e", "l", "l"], and the produced target vector ["e", "l", "l", "o"], "o" being the next character in the sequence. The data is sent through a hidden layer where the network is trained, i.e. the weights are adjusted to produce the probability of each character.

Ba et al. (2014) used an LSTM-RNN in image recognition to preserve information from a glimpse feature vector g_n, Figure 2.2. The input g_n combined with the recurrent output state $r^{(1)}$ is sent through LSTM layers to update the internal representation of the input. As Figure 2.2 shows the final outputs $\hat{l}_{n+1}$, the next location in the glimpse and y_s, the next object to process.

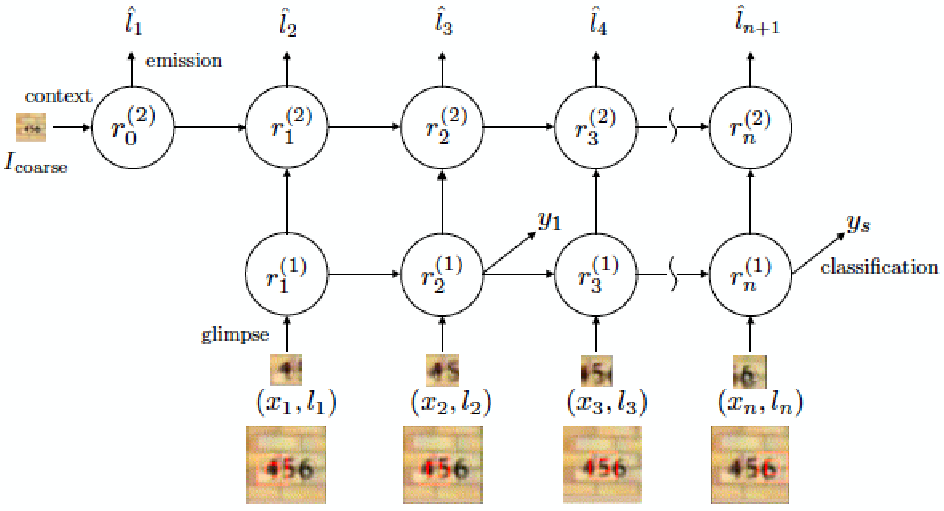

Figure 2.2: Graphical representation of the model Ba et al. (2014) used in image recognition.

Althelaya et al. (2018) evaluated bidirectional LSTM for stock market predictions and found it has better performance on both short- and long-term predictions compared to shallower neural nets.

Graves (2013) generated real-valued sequences from handwriting. LSTM was employed to predict online handwriting by generating sequences with long-range structure. The handwriting was recorded as a sequence of pen-tip locations.

Most papers cited above study the performance of RNNs on applications such as text and speech. Signal processing has not received the same attention from the scientific community, which is a gap that we plan to fill with this thesis by investigating the performance of LSTMs for signal processing and prediction under different scenarios of noise and stationarity.

Chapter 3

State Space Models

The state space model (SSM) is considered to be a subclass of the more general Bayesian hierarchical model (BHM). SSM admits a joint density of the form

$$p(\mathbf{y}_{1:t}, \mathbf{x}_{0:t}, \mathbf{\Theta}) = p(\mathbf{x}_{0:t}, \mathbf{\Theta}) p(\mathbf{y}_{1:t} | \mathbf{x}_{0:t}, \mathbf{\Theta}) \tag{3.1}$$

where the distributions on the right-hand side of (3.1) are the joint prior distribution of the state vector $\mathbf{x}_{0:t}$ and of the parameter vector $\mathbf{\Theta}$, $p(\mathbf{x}_{0:t}, \mathbf{\Theta})$ and the likelihood function $p(\mathbf{y}_{1:t}|\mathbf{x}_{0:t}, \mathbf{\Theta})$, where $\mathbf{y}_{1:t}$ is a vector containing t observations.

When the state is a Markov process and the observation depends only on parameters and state, the joint density takes the following form:

$$p(\mathbf{y}_{1:t}, \mathbf{x}_{0:t}, \mathbf{\Theta}) = p(\mathbf{\Theta}) p(\mathbf{x}_0|\mathbf{\Theta}) \left(\prod_{k=1}^{t} p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{\Theta}) \right) \left(\prod_{k=1}^{t} p(\mathbf{y}_k|\mathbf{x}_k, \mathbf{\Theta}) \right) \tag{3.2}$$

which is also illustrated in 3.1. The right-hand side of (3.2) contains the prior distribution of the parameters $p(\mathbf{\Theta})$, the prior distribution of the initial state $p(\mathbf{x}_0|\mathbf{\Theta})$, the conditionally independent prior distribution of the state transition equation $\prod_{k=1}^{t} p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{\Theta})$ and the conditionally independent likelihood function $\prod_{k=1}^{t} p(\mathbf{y}_k|\mathbf{x}_k, \mathbf{\Theta})$.

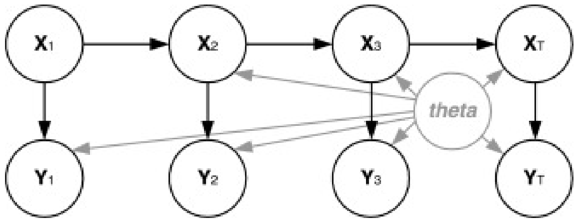

Figure 3.1: Figure illustrates an SSM with parameters $\mathbf{\Theta}$, Markov state process $\mathbf{x}_t$ and observation process $\mathbf{y}_t$ (Mingas et al., 2017).

3.1 Simulate data from SSM

We will simulate data from the following univariate first order SSM to use in our experiments:

$$x_t = \phi x_{t-1} + w_t, \quad w_t \sim \mathrm{N}(0,1) \tag{3.3}$$

$$y_t = x_t + v_t, \quad v_t \sim \mathrm{N}(0,\sigma^2). \tag{3.4}$$

where x_t will be referred to as the signal, and y_t the observed value.

Table 3.1 shows how the model in (3.3) and (3.4) are connected to the SSM class by specifying the different components in the context of (3.2).

Table 3.1: Experiment distributions

Initial state	$x_0 \sim \mathrm{N}(0,1)$
State transition	$x_t \mid x_{t-1}, \phi \sim \mathrm{N}(\phi x_{t-1}, 1)$
Observation	$y_t \mid x_t, \sigma^2 \sim \mathrm{N}(x_t, \sigma^2)$

The aim of the experiments is to predict the signal of the simulated datasets. We will simulate a total of six scenarios, each containing $T = 10000$ time-steps. The simulated date will be both stationary and non-stationary ($\phi \in \{0.1, 1\}$) time-series, each having three signal-to-noise ratios (STNRs) ($1/\sigma^2 \in \{0.2, 0.5, 1\}$). Figure 3.2 shows the resulting time-series and their characteristics. The data will be named by its parameters signal to noise (sn) and transition factor (phi):

$$\text{"sn"} + \text{STNR} + \_ + \text{"phi"} + \phi, \tag{3.5}$$

so that a simulated time-series with STNR equal to 0.2 and ϕ equal to 0.1 will be denoted sn0p2_phi0p1 as the plots in Figure 3.2 show.

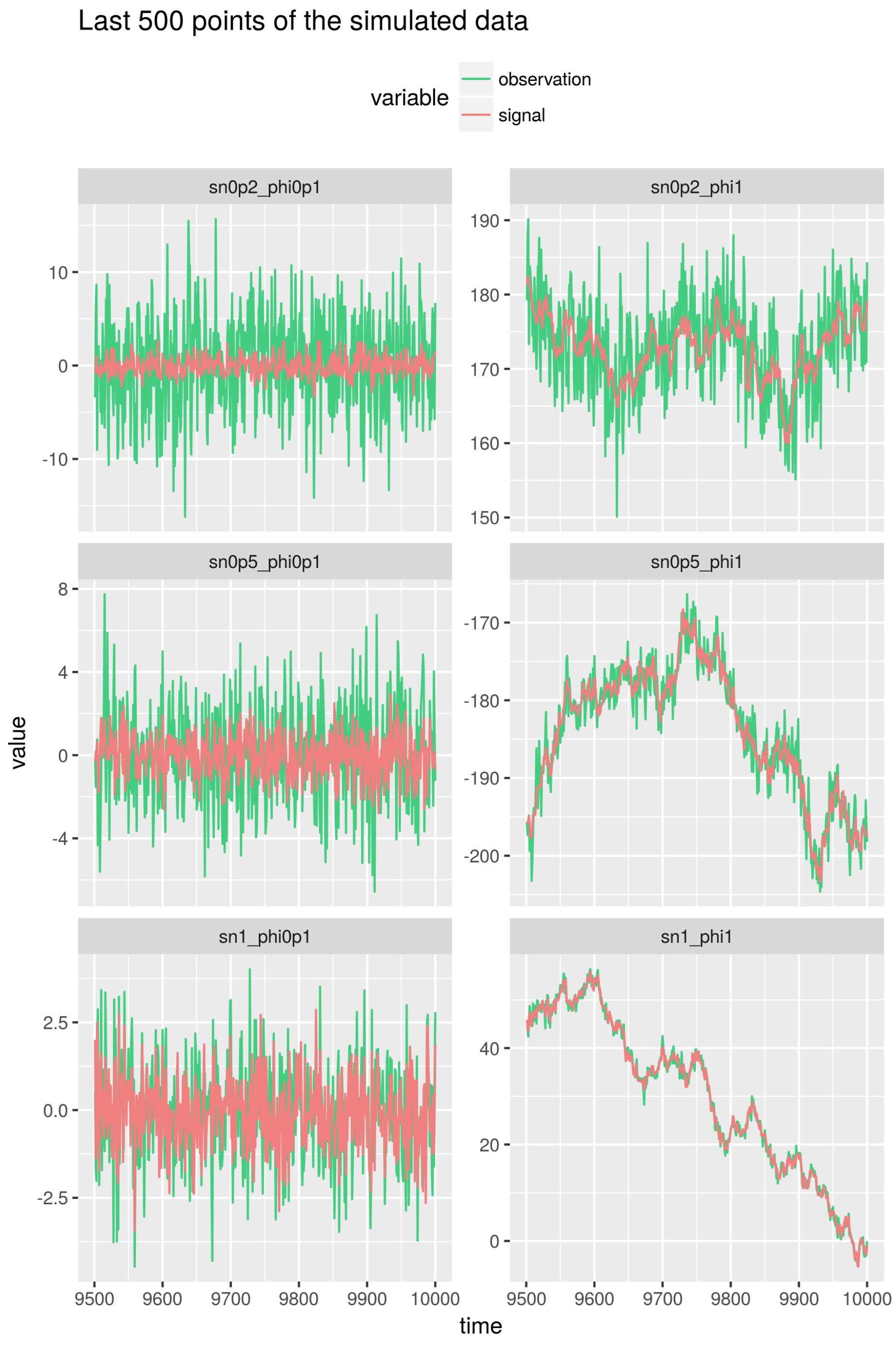

Figure 3.2: The tail (last 500 points) of the simulated datasets. The columns show $\phi = 0.1$ (left) and $\phi = 1$ (right) and the rows are $\sigma^2 = 0.2$ (top), 0.5 (middle) and 1 (bottom).

Chapter 4

Recurrent Neural Networks

Recurrent neural networks (Chollet and Allaire, 2018) are fundamental deep-learning algorithms for sequence processing. Sequence data can be thought of as a sequence of letters, words or values. Applications of RNN are mostly text processing (identifying topic of a book, classifying sentiment of reviews), sequence-to-sequence learning (decoding sentences from one language to another) and time-series forecasting (stock market predictions, temperature forecasting) given recent data. In this thesis an RNN will be employed to time-series forecasting. The aim is to study the behavior of RNN on signal processing in different scenarios of noise and stationarity.

Unlike feedforward nets (e.g. DenseNets and convnets) RNN has memory. The network processes the input incrementally while maintaining a state with information of what is being processed. Instead of processing input in a single step, it loops over all elements in the input-sequence and updates the state constantly. Figure 4.1 illustrates an unfolded recurrent layer wherein each iteration the state is updated with what was produced in the previous step.

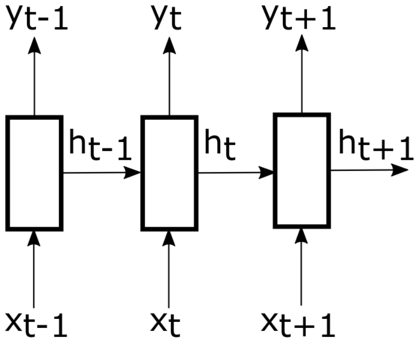

Figure 4.1: Transition of data in a one-layer RNN.

4.1 Defining RNN

A one-layer RNN is a function of a linear transform T between the input at the current time-step, the input x_t and the hidden state at previous time step h_{t-1}:

$$y_t = f(T[x_t, h_{t-1}]), \quad f \in \{\text{sigm, tanh}\}, \tag{4.1}$$

$$T[x_t, h_{t-1}] = U_1 x_t + U_2 h_{t-1} + b, \tag{4.2}$$

where y_t is the observation at time t, U_1 and U_2 are weight matrices to be learned from the data and b is the bias or intercept, also learnable from data. The transition of data is illustrated in Figure 4.1 and the function f is usually taken to be a non-linear function such as sigmoid or a rectified linear function.

More generally, we can define a multilayer RNN (Zaremba et al., 2014) with the following transition function:

$$\text{RNN} : h_t^{l-1}, h_{t-1}^{l} \rightarrow h_t^l \tag{4.3}$$

with layers $l \in [0, L]$, and the function given by

$$h_t^l = f(T[h_t^{l-1}, h_{t-1}^l]), \quad f \in \{\text{sigm, tanh}\} \tag{4.4}$$

also illustrated in Figure 4.2. Here, h_t^0 is an input sequence and h_t^L the activation to predict y_t.

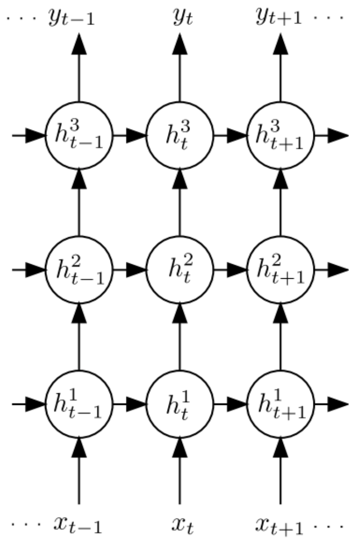

Figure 4.2: CAIS++ (2017) graphically illustrates the transition of data between states in a multilayer RNN.

4.2 Defining LSTM

The long short term memory (LSTM) network is capable of learning long-term dependencies, as well as forgetting unnecessary information based on the data at hand. The LSTM representation, as in Zaremba et al. (2014), is similar to (4.3), but now memory cells c_t^l are introduced:

$$\text{LSTM} : h_t^{l+1}, h_{t-1}^l, c_{t-1}^l \to h_t^l, c_t^l. \tag{4.5}$$

The cell state has minor linear interactions and allows for data to flow along unchanged, as a memory. In RNN the standard module contains one layer, while the LSTM module contains four layers:

$$f = \text{sigm}(T[h_t^{l-1}, h_{t-1}^l]) \tag{4.6}$$
$$i = \text{sigm}(T[h_t^{l-1}, h_{t-1}^l]) \tag{4.7}$$
$$g = \tanh(T[h_t^{l-1}, h_{t-1}^l]) \tag{4.8}$$
$$o = \text{sigm}(T[h_t^{l-1}, h_{t-1}^l]). \tag{4.9}$$

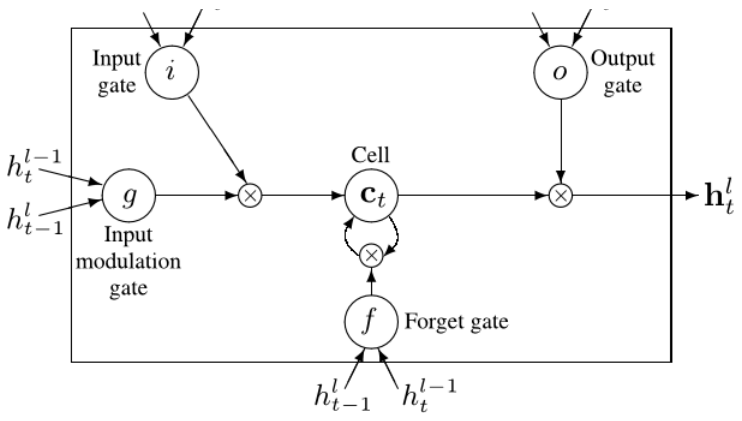

Figure 4.3: A graphical representation of LSTM (Zaremba et al., 2014).

The forget gate f outputs a number between 0 and 1 from the sigmoid activation layer, i.e. $f = 0$ forgets everything in cell c_t, and $f = 1$ keeps all information. The input gate i decides what information to update in the state from the previous time-step, where the sigmoid layer also here returns a number between 0 and 1. In order to update the state, the input layers are multiplied by the input modulation gate g. A candidate vector with elements between -1 and 1 are computed to add or remove information from the state. The cell state is updated in the following way:

$$c_t^l = f \cdot c_{t-1}^l + i \cdot g. \tag{4.10}$$

The output gate o decides what to keep from the memory cell, and output state is

$$h_t^l = o \cdot \tanh c_t^l. \tag{4.11}$$

4.3 Experiment model

The time-series predictions will be performed by an LSTM model using a sliding window approach (Weng, 2017). A window W_t, see Figure 4.4, will act as the feature sequence in order to make one-step-ahead predictions.

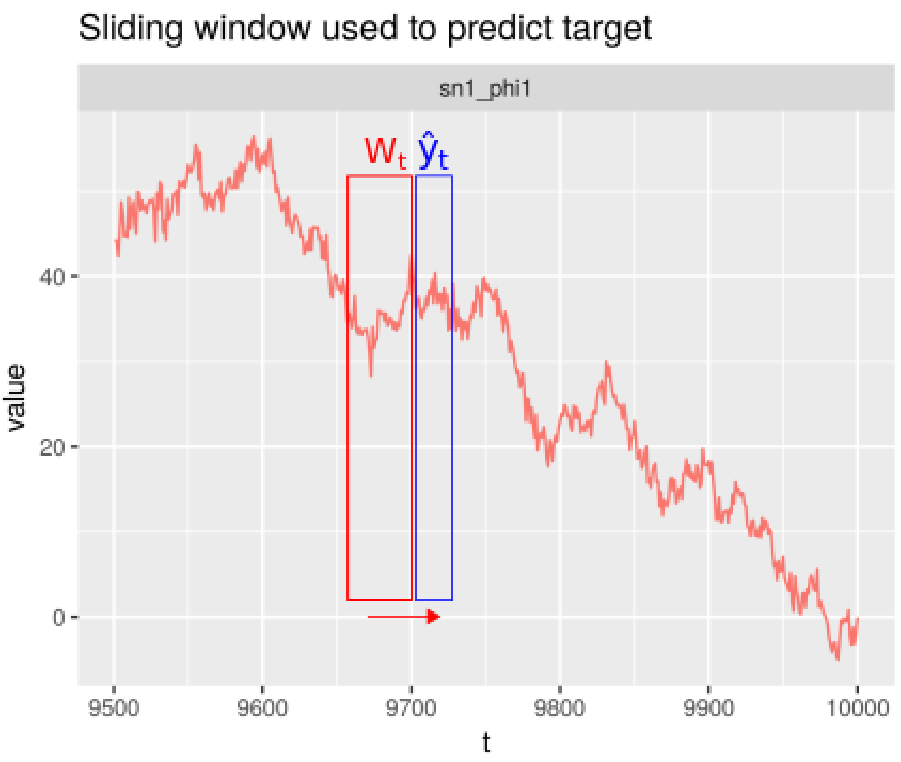

Figure 4.4: The model input is l observed values from the sliding window W_t to predict the target $\hat{y}_t$.

The feature sequence is observed values from the simulations at a given time-step. The length of the feature sequence (or size of the window) is l, which leads to the following input-sequences:

$$W_0 = (y_0, y_1, ..., y_{l-1}) \tag{4.12}$$
$$W_1 = (y_1, y_2, ..., y_l) \tag{4.13}$$
$$\vdots \tag{4.14}$$
$$W_t = (y_t, ..., y_{l-1+t}). \tag{4.15}$$

The value to predict is $y_{(l+t)}$ and is denoted the target of the current time-step, $\hat{y}_t$. The window size will be referred to as **lags** in the future.

4.3.1 Gradient-based optimizing

Gradient-based optmization is the very engine of neural networks (Chollet and Allaire, 2018). The gradual adjustment of trainable parameters i.e. weights and bias in an activation layer (see section 4.1) are the training that machine and deep learning is all about. Gradient descent is the most common way to optimize neural networks (Ruder, 2016).

Gradient descent aims to minimize an objective function $J(\theta)$ parameterized by parameters θ in order to reach a local minimum. By computing the gradient, $\Delta J(\theta)$, a slope is created in a downhill direction to the local minimum. The **learning_rate**, η, determines the step-size in this direction and **optimizer_steps** determines how many times the parameters are updated for each processed data batch.

It is important to have an appropriate learning rate as too big steps can lead to divergence and too small learning rate can lead to slow convergence. Jordan (2018) illustrates this effect accurately in Figure 4.5.

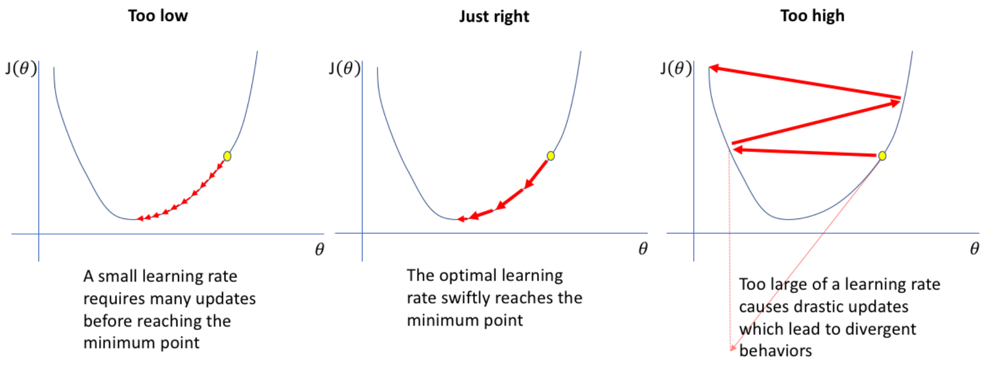
Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Figure 4.5

The basic update rule for stochastic gradient descent (SGD) (Duchi et al., 2011) is

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i} \tag{4.16}$$

but can vary for variations of the method. We wish to study the behaviour of the model with two variations of gradient descent, which are described below.

Adagrad

Adagrad (Duchi et al., 2011) is short for Adaptive Gradient Algorithm and adds a term to SGD in order to adapt the learning rate for each parameter θ_i at every times-step t. Instead of updating all parameters with the same learning rate, Adagrad modifies the general learning rate based on the past gradients $g_{t,i} = \Delta_{\theta_t} J(\theta_{t,i})$ computed for θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - G_t^{-1} \eta g_{t,i} \tag{4.17}$$

where $G := \operatorname{diag}(\sum_{j=1}^{t} g_j g_j^T)^{1/2}$.

Adagrad performs smaller updates to more frequent features and larger updates to the less frequent features. Dean et al. (2012) showed Adagrad can be used for training large-scale neural nets as it improved the robustness of SGD.

Ftrl

Ftrl or Ftrl-proximally (McMahan et al., 2013) is short for Follow the (proximally) Regularized Leader. The algorithm has been used by McMahan et al. (2013) due to its ability to handle larger data sets and larger models. The aim of the algorithm is to get both sparsity and improved accuracy on predictions from the standard SGD, and has shown to do so with minimum computing resources. Ftrl is SGD with regularization and is defined as (McMahan, 2011)

$$\theta_{t+1,i} = \arg\min_{\theta}(g_{1:t,i}\theta_i + t\lambda\|\theta_i\|_1 + \frac{1}{2}\sum_{s=1}^{t}\sigma_s\|\theta_i - \theta_{s,i}\|_2^2). \tag{4.18}$$

The first term of the update is an approximation to the objection function, second term is an non-smooth convex function and the last term is an additional strong convexity. Here, $\sigma_s^t = \frac{1}{\eta_t}$ and are generalized learning rates, while $\lambda > 0$ induces sparsity in the way that zero-features and less important features are removed.

4.3.2 Model setup

Arguments	**Value**
Training optimizer	{"Adagrad", "Ftrl"}
Number of optimizing steps	{10, 100}
Learning rate	{0.1, 0.5}
Number lags	{5, 10, 20}
Number of hidden layers	{1, 2, 3}

Table 4.1: The set of arguments for the LSTM model to investigate.

The model used in the experiments are designed in Tensorflow (Abadi et al., 2015). The model has an RNN architecture containing basic LSTM cells. Training optimizer algorithm, number of optimizing steps and learning rate belongs to the parameter training session (Section 4.3.1). The number of lags or size of feature sequence will be varied. The model will also stack a different number of cells. The arguments variables are listed in Table 4.1 and all combinations will be tested.

The objective function to minimize is the mean squared error metric (MSE):

$$MSE = \frac{1}{n}\big(\sum_{i}^{n} y_i - \hat{y}_i\big)^2. \tag{4.19}$$

Table 4.2: Results of the experiment, showing simulation parameters, CMSE and optimizer hyperparameters for the 5 best models used to predict each of the simulated time-series. Bold rows mark the best result for each series.

Time-series	ϕ	STNR	CMSE	Optimizer	Optimizer steps	Learning rate	Lags	Layers
sn0p2_phi0p1	**0.1**	**0.2**	**2674.3127**	**Adagrad**	**10**	**0.1**	**5**	**1**
sn0p2_phi0p1	0.1	0.2	2674.8955	Adagrad	10	0.1	10	1
sn0p2_phi0p1	0.1	0.2	2675.1684	Adagrad	10	0.1	20	1
sn0p2_phi0p1	0.1	0.2	2675.2325	Ftrl	10	0.1	5	1
sn0p2_phi0p1	0.1	0.2	2675.5648	Ftrl	10	0.1	10	1
sn0p5_phi0p1	**0.1**	**0.5**	**504.9497**	**Ftrl**	**10**	**0.1**	**10**	**1**
sn0p5_phi0p1	0.1	0.5	505.0840	Adagrad	10	0.1	10	1
sn0p5_phi0p1	0.1	0.5	505.2548	Adagrad	10	0.1	20	1
sn0p5_phi0p1	0.1	0.5	505.2751	Ftrl	10	0.1	10	2
sn0p5_phi0p1	0.1	0.5	505.3038	Ftrl	100	0.5	20	1
sn1_phi0p1	**0.1**	**1**	**202.6560**	**Ftrl**	**10**	**0.1**	**5**	**2**
sn1_phi0p1	0.1	1	202.9083	Adagrad	10	0.1	10	1
sn1_phi0p1	0.1	1	203.0482	Adagrad	10	0.1	5	1
sn1_phi0p1	0.1	1	203.0808	Ftrl	10	0.1	10	1
sn1_phi0p1	0.1	1	203.2258	Ftrl	10	0.1	10	2
sn0p2_phi1	**1**	**0.2**	**3909.5377**	**Ftrl**	**100**	**0.5**	**5**	**3**
sn0p2_phi1	1	0.2	3911.7528	Ftrl	100	0.5	10	2
sn0p2_phi1	1	0.2	4324.2036	Ftrl	100	0.5	10	3
sn0p2_phi1	1	0.2	4412.7257	Adagrad	100	0.5	10	3
sn0p2_phi1	1	0.2	4427.3217	Adagrad	100	0.5	5	3
sn0p5_phi1	**1**	**0.5**	**916.6313**	**Adagrad**	**100**	**0.5**	**10**	**3**
sn0p5_phi1	1	0.5	1023.1265	Ftrl	100	0.5	20	3
sn0p5_phi1	1	0.5	1062.7970	Adagrad	100	0.5	20	3
sn0p5_phi1	1	0.5	1287.9312	Adagrad	100	0.5	10	2
sn0p5_phi1	1	0.5	1292.1424	Adagrad	100	0.5	5	3
sn1_phi1	**1**	**1**	**432.1487**	**Ftrl**	**100**	**0.5**	**5**	**3**
sn1_phi1	1	1	523.2287	Adagrad	100	0.5	5	3
sn1_phi1	1	1	545.3177	Ftrl	100	0.5	10	2
sn1_phi1	1	1	569.9921	Ftrl	100	0.5	20	2
sn1_phi1	1	1	624.1589	Ftrl	100	0.5	5	3

4.4 Results

The results, see Table 4.2, indicate stationarity and noise ratio have a significant impact on the LSTM performance level. The CMSE increases with noise and is in general much higher in the non-stationary scenarios. In terms of the hyperparameters, stationarity also impacts which model yields the best results. The LSTM perform better with fewer optimizing steps, lower learning rate and a single-layered LSTM for stationary time-series, whereas best results were obtained with more optimizing steps, higher learning rate and more layers for non-stationary time-series. This is to be expected as models with higher learning rate are more capable to adapt to the changing dynamics of non-stationarity data.

Parameter analysis

In terms of hyperparameters in the optimizer, the results can be further analyzed. Figure 4.6 illustrates the average MSE plotted against each individual parameter type. The figure visually supports what was interpreted in Table 4.2; a low number of steps, low learning rate and low layers work better for stationary time-series while the opposite is true for non-stationary time-series.

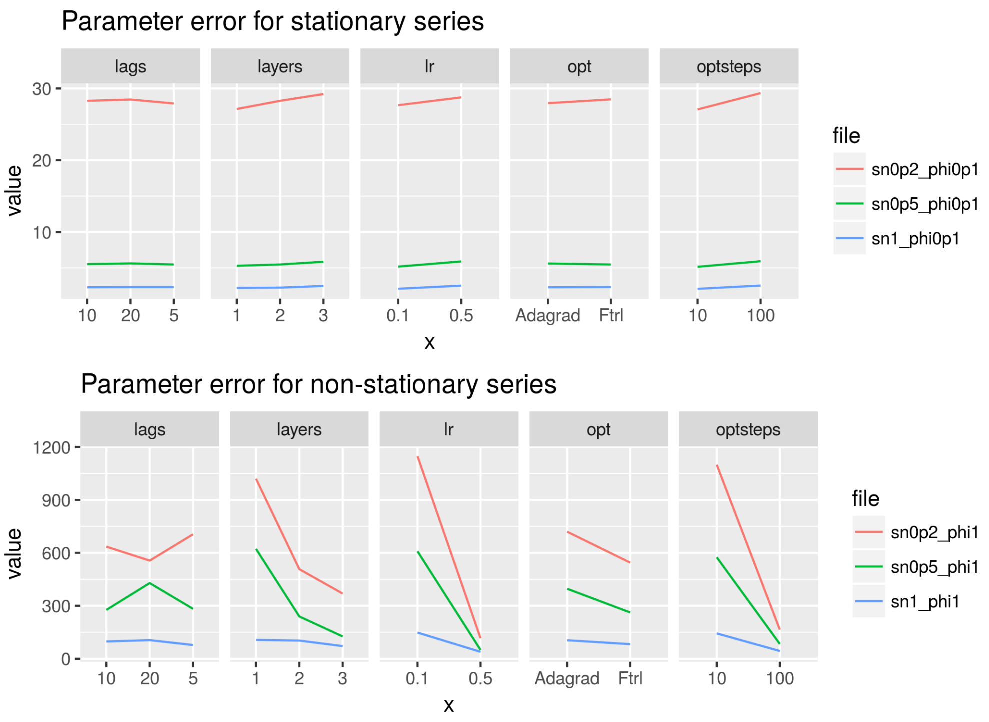

Figure 4.6: The two plots show mean MSE of all predictions it has made in the different scenarios.

Table 4.3: Results for experiment, showing parameters and lowest cumulative MSE (CMSE) for each time-series.

Time-series	ϕ	STNR	CMSE	Optimizer	Optimizer steps	Learning rate	Lags	Layers
sn0p2_phi0p1	0.1	0.2	2674.3127	Adagrad	10	0.1	5	1
sn0p5_phi0p1	0.1	0.5	504.9497	Ftrl	10	0.1	10	1
sn1_phi0p1	0.1	1	202.6560	Ftrl	10	0.1	5	2
sn0p2_phi1	1	0.2	3909.5377	Ftrl	100	0.5	5	3
sn0p5_phi1	1	0.5	916.6313	Adagrad	100	0.5	10	3
sn1_phi1	1	1	432.1487	Ftrl	100	0.5	5	3

Predictions

The LSTM predictions are shown in Figure 4.7, plotted against the signal. The plots show the last 500 time-steps in the time-series. The prediction models used for signal estimations are presented in Table 4.3.

In conclusion, the LSTM-model performs better in stationary, low-noise scenarios. This is supported by both CMSE evaluation and parameter analysis. The LSTM struggles more on non-stationary time-series, where the performance is poor. The results suggest the use of different values of the hyperparameters (learning rate, optimizing-steps, layers) dependent on the stationarity of the time-series. The results obtained were not very sensitive to the choice of the optimizer and the number of lags used, even though figure 4.6 suggest to use Ftrl for non-stationary series.

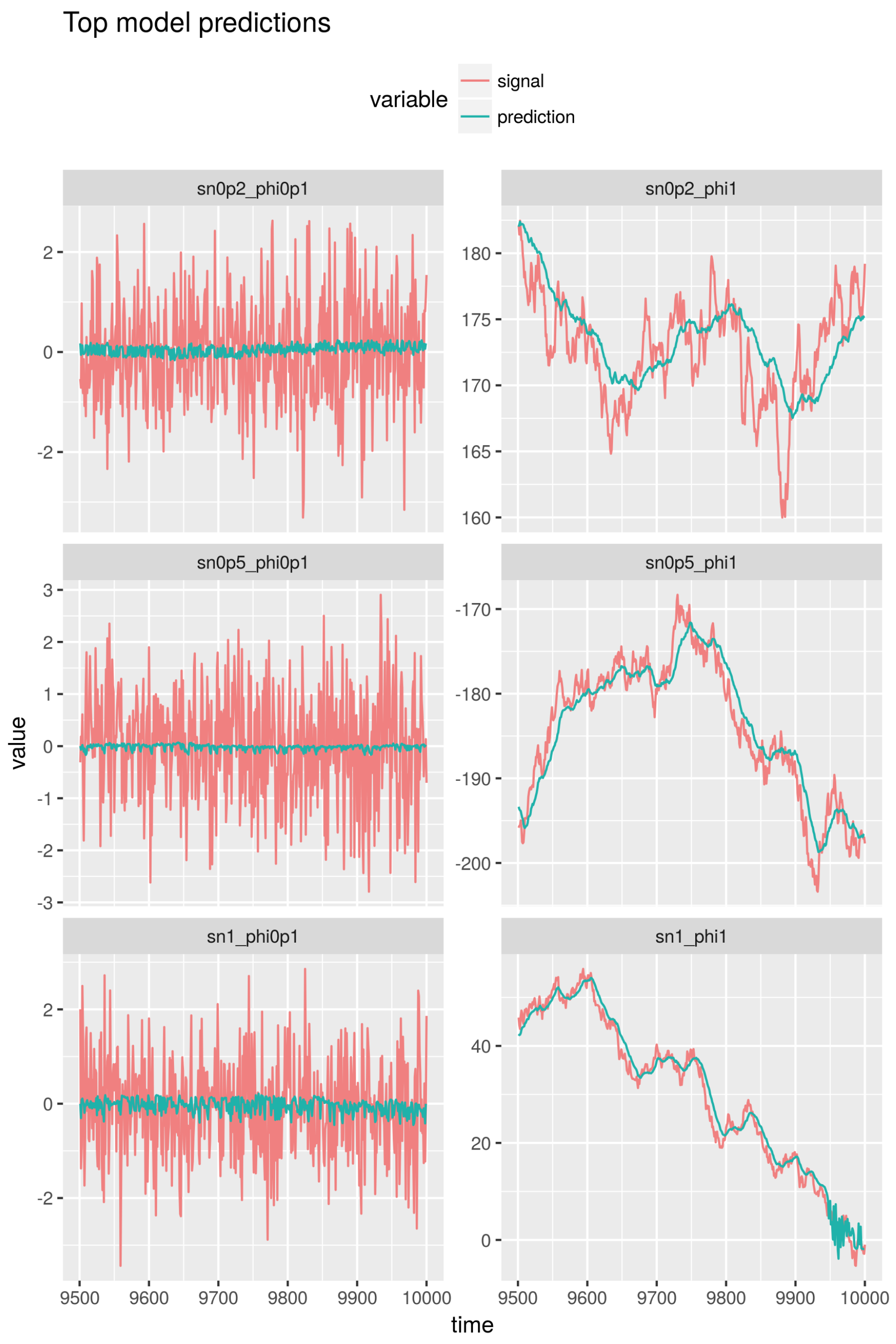

Figure 4.7: Predictions made by the models according to results from Table 4.3.

Chapter 5

Sequential Monte Carlo

For the vast majority of Bayesian models, the posterior distribution of its parameters is not available in closed-form. This requires the use of approximation techniques to compute posterior distributions. If the model of interest is Gaussian, linear and have low dimensionality, methods such as Kalman filtering and hidden Markov model (HMM) filter are convenient ways of computing sequence of posterior distributions through time. However, for more complex models, having high dimensionality, non-linearity and/or non-Gaussianity, simulation-based methods such as Sequential Monte Carlo (SMC) and Markov Chain Monte Carlo (MCMC) are attractive. For SMC, this includes sequential importance sampling (SIS) and sequential importance resampling (SIR), otherwise known as particle and bootstrap filtering respectively (Doucet et al., 2001). Particle Markov Chain Monte Carlo (PMCMC) uses SMC algorithms to design efficient high dimensional proposal distributions for MCMC algorithms (Andrieu et al., 2010). We will define and use both the Bootstrap filtering and the PMCMC methods in the following sections of this chapter.

5.1 Monte Carlo Estimation

For the rest of this chapter, we will use the same notation used for defining SSMs in Chapter 3. We are also assuming that most of the computations are done conditional on the value of the static parameters $\mathbf{\Theta}$, unless stated otherwise. The aim is to estimate the posterior distribution

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \frac{p(\mathbf{y}_{1:t}|\mathbf{x}_{0:t})p(\mathbf{x}_{0:t})}{\int p(\mathbf{y}_{1:t}|\mathbf{x}_{0:t})} \tag{5.1}$$

If we are able to sample N particles from the posterior distribution above, we can compute an empirical particle sampling estimate of (5.1) by

$$P_N(d\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \frac{1}{N}\sum_{i=1}^{N} \delta_{\mathbf{x}_{0:t}^{(i)}}(d\mathbf{x}_{0:t}) \tag{5.2}$$

where $\delta_{\mathbf{x}_{0:t}^{(i)}}(d\mathbf{x}_{0:t})$ denotes the delta-Dirac mass. This representation yield the following estimate for the expected value of $f_t(\mathbf{x}_{0:t})$ with respect to $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$:

$$I_N(f_t) = \int f_t(\mathbf{x}_{0:t})P_N(d\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \frac{1}{N}\sum_{i=1}^{N} f_t(\mathbf{x}_{0:t}^{(i)}) \tag{5.3}$$

5.2 Sequential Monte Carlo

As previously mentioned, most complex models do not admit closed-form solutions to the posterior $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$, making it hard to sample particles directly from it. Importance sampling methods work around this by sampling from an importance sampling distribution, $\pi(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$. The importance sampling distribution is ideally selected to be a distribution that is as close as possible to the target distribution while being easy to sample from.

Then, the importance weights

$$w(\mathbf{x}_{0:t}) = \frac{p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})}{\pi(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})} \tag{5.4}$$

are used to correct the fact that the samples came from $\pi(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ instead of $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$, leading to the following estimates of $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ and $I(f_t)$

$$\hat{P}_N(d\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \sum_{i=1}^{N} \tilde{w}_t^{(i)} \delta_{\mathbf{x}_{0:t}^{(i)}}(d\mathbf{x}_{0:t}). \tag{5.5}$$

and

$$\hat{I}_N(f_t) = \sum_{i=1}^{N} f_t(\mathbf{x}_{0:t}^{(i)})\tilde{w}_t^{(i)} \tag{5.6}$$

where $\tilde{w}_t^{(i)}$ are normalized weights $\tilde{w}_t^{(i)} = \frac{w(\mathbf{x}_{0:t}^{(i)})}{\sum_{j=1}^{N} \tilde{w}(\mathbf{x}_{0:t}^{(j)})}$.

Sequential Importance Sampling

Computing an estimate without modifying the past simulated trajectories gives the SIS method. The importance distribution admits a marginal distribution at the previous time step and when iterating over time one obtains

$$\pi(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \pi(\mathbf{x}_0)\prod_{k=1}^{t} \pi(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{y}_{1:k}) \tag{5.7}$$

This means that the importance weights can be evaluated recursively:

$$\tilde{w}_t^{(i)} \propto \tilde{w}_{t-1}^{(i)} \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(i)})p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})}{\pi(\mathbf{x}_t^{(i)}|\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t})}. \tag{5.8}$$

When the prior distribution is adopted as importance distribution the recursion becomes

$$p(\mathbf{x}_{0:t}) = p(\mathbf{x}_0) \prod_{k=1}^{t} p(\mathbf{x}_k|\mathbf{x}_{k-1}) \tag{5.9}$$

and importance weights are given by $\tilde{w}_t^{(i)} \propto \tilde{w}_{t-1}^{(i)} p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$.

5.2.1 Bootstrap filter

The SIS method becomes ineffective when t increases due to particle degenaration. Similarly to SIS, the Bootstrap rely on importance sampling, but eliminate particles of low importance by adding an additional selection step to the SIS algorithm.

Algorithm 1 describes the pseudocode for the Bootstrap particle filter algorithm.

Algorithm 1: Bootstrap particle filter

```
$t = 0$
Initalization
for $i = 1 : N$ do
    sample $\mathbf{x}_0^{(i)} \sim p(\mathbf{x}_0)$
end
Set $t = 1$
Importance sampling
for $i = 1 : N$ do
    Sample $\mathbf{x}_t^{(i)} \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)})$
    Set $\mathbf{x}_{0:t}^{(i)} = (\mathbf{x}_{0:t-1}^{(i)}, \tilde{\mathbf{x}}_t^{(i)})$
    Evaluate $\tilde{w}_t^{(i)} = p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$
end
Normalize weights
Particle selection
Resample $N$ particles $(\mathbf{x}_t^{(i)})$ from $(\tilde{\mathbf{x}}_t^{(i)})$ with $i = 1, ..., N$
Set $t \leftarrow t + 1$
Go to importance sampling step.
```

5.2.2 PMCMC

Particle Markov Chain Monte Carlo (PMCMC) uses SMC algorithms to design efficient high dimensional proposal distributions for MCMC algorithms (Andrieu et al., 2010) to sample from the joint posterior distribution of the states and the static parameters, $p(\mathbf{x}_{0:t}, \mathbf{\Theta}|\mathbf{y}_{1:t})$.

Initially, a set of parameters $\boldsymbol{\theta}$ are generated and proposed from a proposal distribution, q. Then, a corresponding state vector, $x_{0:t}$, is generated by a bootstrap filter using the newly proposed parameters. The marginal likelihood $l = p(\mathbf{y}_{1:t}|\boldsymbol{\theta})$

is then used to either accept or reject the state and parameter, according to the following ratio:

$$\alpha = \min\left(1, \frac{l'p(\boldsymbol{\theta'})q(\boldsymbol{\theta}|\boldsymbol{\theta'})}{lp(\boldsymbol{\theta})q(\boldsymbol{\theta'}|\boldsymbol{\theta})}\right) \tag{5.10}$$

Algorithm 2 describes the pseudocode for the PMCMC algorithm used in this thesis.

Algorithm 2: Particle-filter marginal Metropolis-Hastings

```
Porpose parameters
$\boldsymbol{\theta}' \sim q(\boldsymbol{\theta'}|\boldsymbol{\theta})$
Filter
$(l', \mathbf{x}'_{0:t}) \rightarrow \text{filter}(\boldsymbol{\theta'})$
$\alpha \sim \text{U}(0,1)$
Accept or reject
if $\alpha = \min\left(1, \frac{l'p(\boldsymbol{\theta'})q(\boldsymbol{\theta}|\boldsymbol{\theta'})}{lp(\boldsymbol{\theta})q(\boldsymbol{\theta'}|\boldsymbol{\theta})}\right)$ then
    Accept move
    return $(\boldsymbol{\theta}', \mathbf{x}'_{0:t}, l')$
else
    Rejected move
    return $(\boldsymbol{\theta}, \mathbf{x}_{0:t}, l)$
end
```

5.2.3 Experiment models

We will estimate the signal based on the observed data simulated in Section 3.1 using both the Bootstrap particle filter defined in Section 5.2.1 and the PMMC method defined in Section 5.2.2. For our experiment, we will use the same model outlined in Section 3.1, under two scenarios.

The **optimal** scenario will assume that we know all the static parameters in the model, i.e. ϕ, the observational noise variance σ^2 and the signal noise variance σ_x^2. In this scenario, we only need to estimate the signal $\mathbf{x}_{0:t}$.

The second scenario, referred here as **semi_optimal**, will assume both variances σ^2 and σ_x^2 to be unknown and define a inverse gamma prior to each of them having shape = 1 and scale = 10. The prediction results can se seen in Figures 5.1, 5.2 and 5.3.

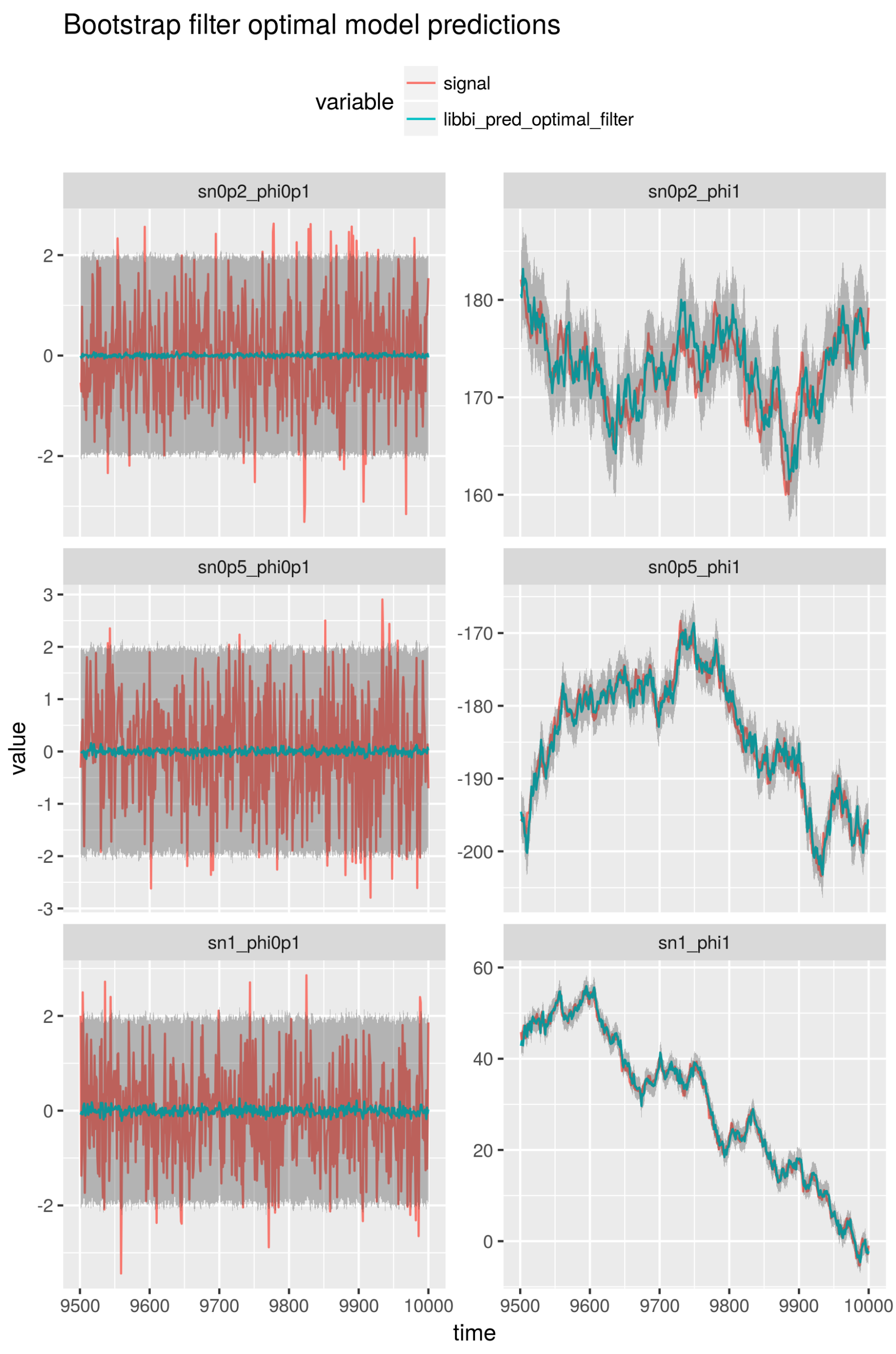

Figure 5.1: The optimal filter model predictions plotted against the true signal with a 95% credibility interval.

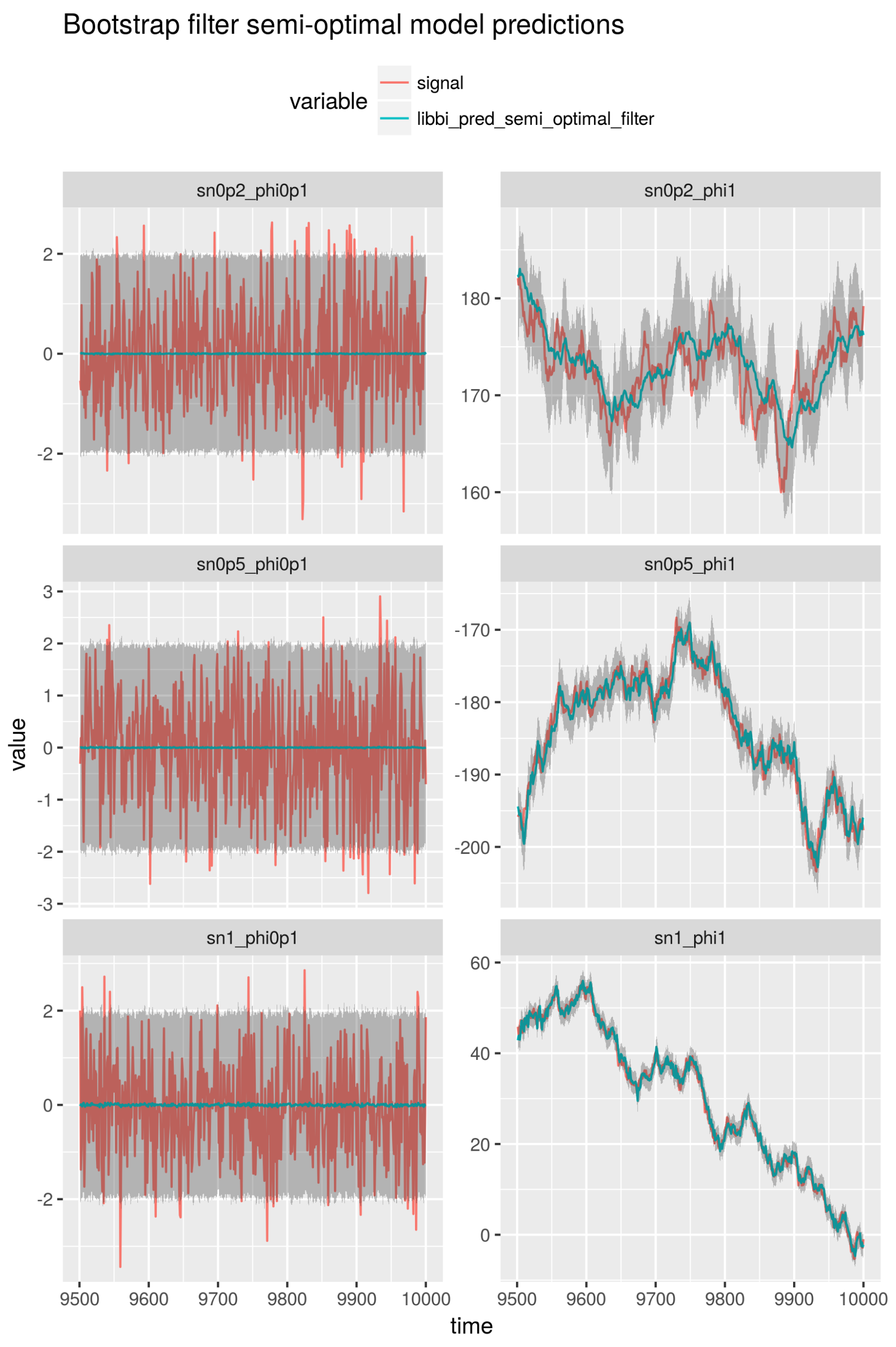

Figure 5.2: The semi-optimal filter model predictions plotted against the signal 95% credibility interval.

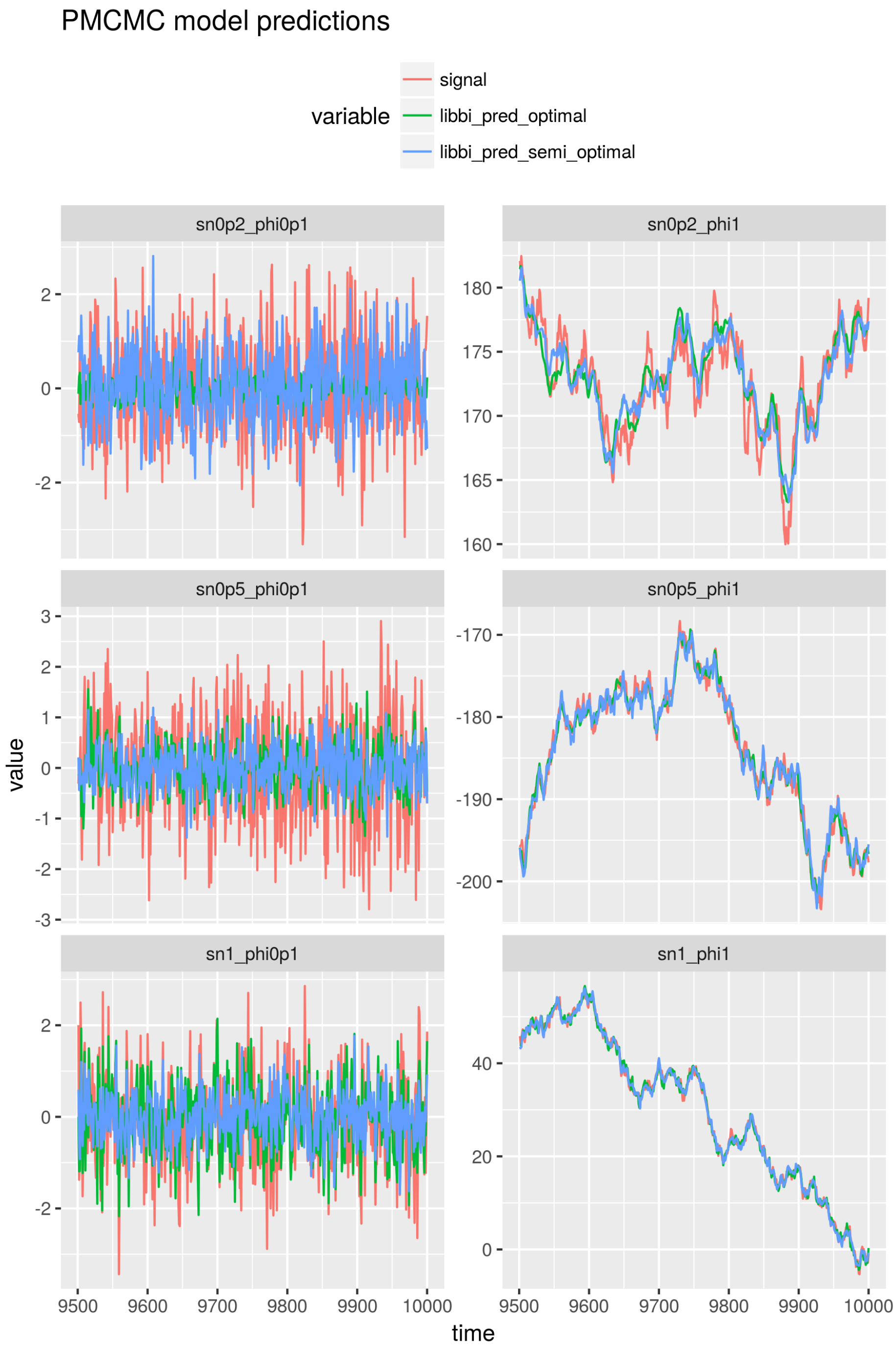

Figure 5.3: The PMCMC model predictions using the optimal model and the semi-optimal model plotted against the signal.

Chapter 6

Discussion and conclusion

This chapter sums up the findings of our experiments. The discussion section will consist of method comparison, both in terms of error evaluation (Figure 6.1) and prediction results (Figures 6.2 and 6.3). The experimental methods are labeled in the figures as Table 6.1 describes:

Table 6.1: The labels of the models in figures.

Model	**Model**
LSTM	tf_pred
PMCMC optimal model	libbi_pred_optimal
PMCMC semi-optial model	libbi_pred_semi_optimal
Bootstrap filter optimal model	libbi_pred_optimal_filter
Bootstrap semi-optimal model	libbi_pred_semi_optimal_filter

After the discussion, we will draw a conclusion to the problem we have investigated; how does the LSTM performance level compare to the custom-tailored SMC methods in the different scenarios?

6.1 Discussion

The error evaluation plot shows the PMCMC methods, the Bootstrap methods plotted against the best LSTM model. Errors from the LSTM model predictions are comparable with the Bootstrap filter in stationary scenarios, while not on the same level as PMCMC. In non-stationary scenarios the LSTM predictions suffer from bigger and increasing errors. It is worse than both Bootstrap filter and PMCMC. Bootstrap filter shows the same amount of errors as LSTM in stationary scenarios, but estimate better in non-stationary scenarios.

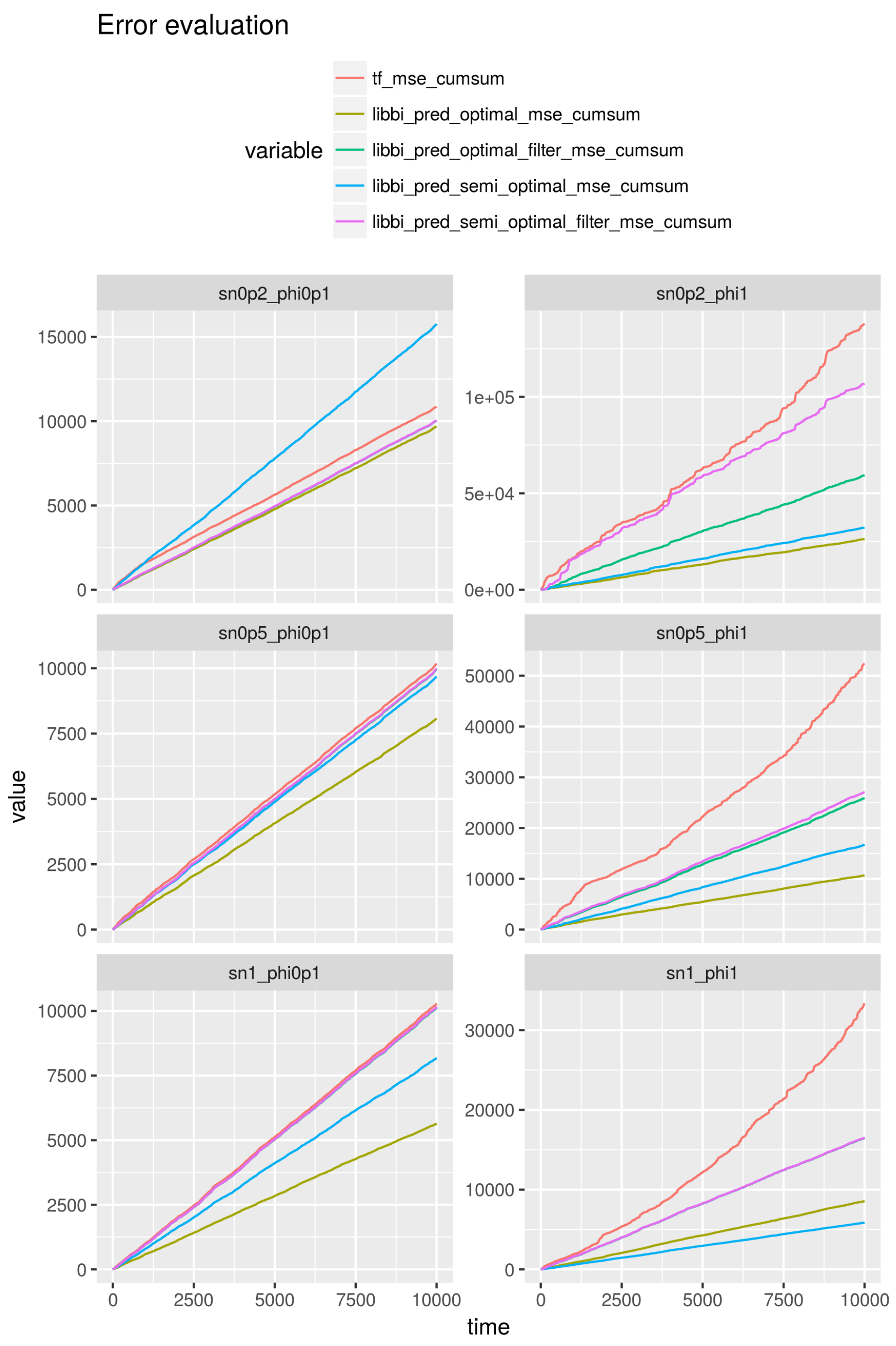

Figure 6.1: CMSE comparison for all experiment models.

Figure 6.2 shows how the LSTM model is not doing a very good prediction job in non-stationary scenarios. It is consistently of and getting worse as noise increases.

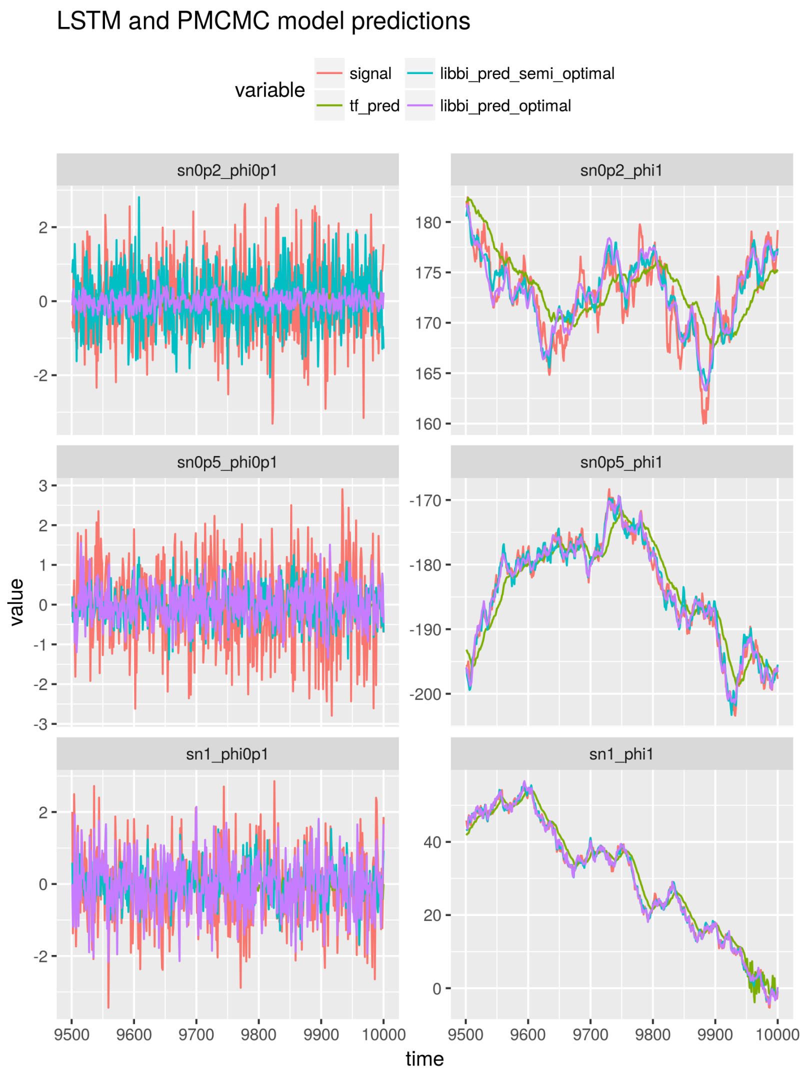

Figure 6.2: LSTM predictions plotted against PMPMC model estimatios.

Figure 6.3 shows Bootstrap filtering methods are performing better than LSTM in non-stationary scenarios. With stationarity, on the other hand, LSTM performs quite well compared to Bootstrap.

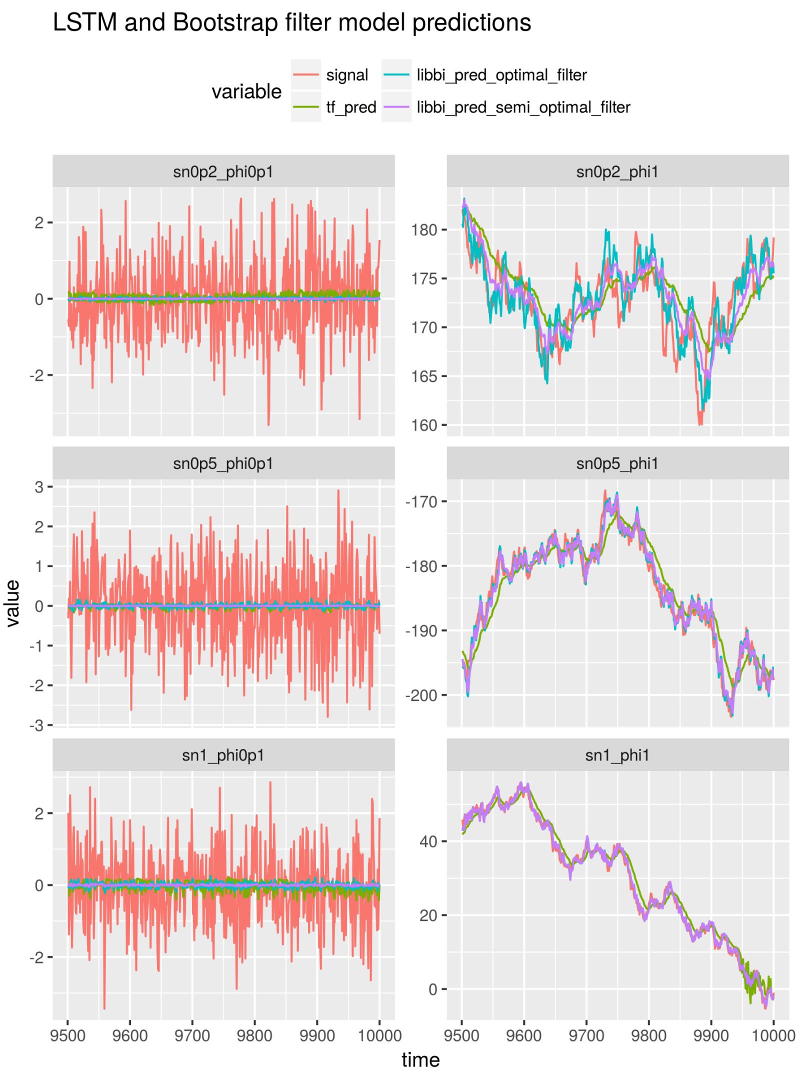

Figure 6.3: LSTM predictions plotted against Bootstrap model estimatios.

6.2 Conclusion

The main conclusion is that LSTM performs well in stationary scenarios, but not very well in non-stationary scenarios. Compared to the custom-tailored SSM models the results proved the LSTM model got worse with time. Further work will be to tune the hyperparameters from Section 4.3.2 in the direction it showed improvement. LSTM might benefit from tuning the parameters upwards in non-stationary scenarios in order to improve te prediction results where they evidently got worse with time.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
URL `https://www.tensorflow.org/`

Althelaya, K. A., El-Alfy, E.-S. M., Mohammed, S., apr 2018. Evaluation of bidirectional LSTM for short-and long-term stock market prediction. In: 2018 9th International Conference on Information and Communication Systems (ICICS). IEEE.
URL `https://doi.org/10.1109/iacs.2018.8355458`

Andrieu, C., Doucet, A., Holenstein, R., jun 2010. Particle markov chain monte carlo methods. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 72 (3), 269–342.
URL `https://doi.org/10.1111/j.1467-9868.2009.00736.x`

Ba, J., Mnih, V., Kavukcuoglu, K., 2014. Multiple object recognition with visual attention.

Brownlee, J., Aug. 2017. Multivariate Time Series Forecasting with LSTMs in Keras.
URL `https://machinelearningmastery.com/multivariate-time-series-forecasting-lst`

CAIS++, 2017. Lesson 8: Recurrent neural networks.
URL `http://caisplusplus.usc.edu/blog/curriculum/lesson8`

Cavalcante, R. C., Brasileiro, R. C., Souza, V. L., Nobrega, J. P., Oliveira, A. L., 2016. Computational Intelligence and Financial Markets: A Survey and Future Directions. Expert Systems with Applications 55, 194–211.
URL `http://www.sciencedirect.com/science/article/pii/S095741741630029X`

Chollet, F., 2015. keras. `https://github.com/fchollet/keras`.

Chollet, F., Allaire, J. J., 2018. Deep Learning with R. Manning Publications.
URL `https://www.amazon.com/Deep-Learning-R-Francois-Chollet/dp/161729554X?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=161729554X`

Chowdhury, G. G., jan 2005. Natural language processing. Annual Review of Information Science and Technology 37 (1), 51–89.
URL `https://doi.org/10.1002/aris.1440370103`

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., Ng, A. Y., 2012. Large scale distributed deep networks. In: NIPS.

Doucet, A., De Freitas, N., Gordon, N., 2001. Sequential Monte Carlo Methods in Practice.

Duchi, J., Hazan, E., Singer, Y., Jul. 2011. Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 12, 2121–2159.
URL `http://dl.acm.org/citation.cfm?id=1953048.2021068`

Gers, F. A., Eck, D., Schmidhuber, J., 2001. Applying LSTM to Time Series Predictable through Time-Window Approaches. In: Dorffner, G., Bischof, H., Hornik, K. (Eds.), Artificial Neural Networks — ICANN 2001. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 669–676.

Graves, A., 2013. Generating sequences with recurrent neural networks.

Hochreiter, S., 1991. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München.

Hochreiter, S., Schmidhuber, J., Nov. 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.
URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`

Jordan, J., Mar. 2018. Setting the learning rate of your neural network.
URL `https://www.jeremyjordan.me/nn-learning-rate/`

Karpathy, A., 2015. The unreasonable effectiveness of recurrent neural networks.
URL `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

McMahan, H. B., 2011. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS).

McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., Chikkerur, S., Liu, D., Wattenberg, M., Hrafnkelsson, A. M., Boulos, T., Kubica, J., 2013. Ad click prediction: a view from the trenches. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).

Mingas, G., Bottolo, L., Bouganis, C.-S., 2017. Particle MCMC algorithms and architectures for accelerating inference in state-space models. International Journal of Approximate Reasoning 83, 413–433.
URL http://www.sciencedirect.com/science/article/pii/S0888613X16302092

Ruder, S., 2016. An overview of gradient descent optimization algorithms. CoRR abs/1609.04747.
URL http://arxiv.org/abs/1609.04747

Weng, L., Jul. 2017. Predict Stock Prices Using RNN: Part 1.
URL https://lilianweng.github.io/lil-log/2017/07/08/predict-stock-prices-using-RNN-part-1.html#model-construction

Zaremba, W., Sutskever, I., Vinyals, O., 2014. Recurrent Neural Network Regularization.
URL http://arxiv.org/abs/1409.2329