Horizon 2020


Understanding Europe's Fashion Data Universe

# Software Requirements: SSM Library for Time Series Modelling and Trend Prediction

## Deliverable number: D1.4

Version 3.0

Funded by the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 732328

| | |
|---|---|
| **Project Acronym:** | FashionBrain |
| **Project Full Title:** | Understanding Europe's Fashion Data Universe |
| **Call:** | H2020-ICT-2016-1 |
| **Topic:** | ICT-14-2016-2017, Big Data PPP: Cross-sectorial and cross-lingual data integration and experimentation |
| **Project URL:** | https://fashionbrain-project.eu |

| | |
|---|---|
| Deliverable type | Report (R) |
| Dissemination level | **Public** (PU) |
| Contractual Delivery Date | 31/12/2017 |
| Resubmission Delivery Date | 27/02/2019 |
| Number of pages | 20, the last one being no. 14 |
| Authors | Alan Akbik, Duncan Blythe - Zalando |
| Peer review | Benjamin Winter - BEUTH |

## Change Log

| Version | Date | Status | Partner | Remarks |
|---|---|---|---|---|
| 1.0 | 31/12/2017 | Final | Zalando | Rejected 15/03/2018 |
| 2.0 | 30/06/2018 | Resubmitted Final | Zalando | Rejected 15/10/2018 |
| 2.1 | 15/02/2019 | Revised Draft | Zalando | |
| 3.0 | 27/02/2019 | Resubmitted Final | Zalando | |

## Deliverable Description

Most modern algorithms of State Space Models (SSM) for time series analysis and probabilistic inference will be summarised in this deliverable and will be used as a basis for future software developments in the project. The output will be available for project internal and public use.

## Abstract

This deliverable discusses the task of time-series analysis with regards to the four subtasks of forecasting, distribution modeling, dependency analysis and missing value imputation and their relevance for the fashion industry. We give an overview of classic approaches to solving these tasks and argue that these approaches are subject to restrictive assumptions that limit their applicability to fashion industry use cases. We then present a solution for time-series analysis based on recurrent neural networks (RNNs) and illustrate how they can be used to perform all four subtasks of time-series analysis and even be trained over incomplete data. Finally, we discuss an open source library for time-series modeling that we created and publicly released for this work package.

# Table of Contents

# List of Figures

# List of Acronyms and Abbreviations

| | |
|---|---|
| **AS** | Assumption |
| **DA** | Dependency Analysis |
| **DM** | Distribution Modeling |
| **EM** | Expectation Maximization |
| **F** | Forecasting |
| **LSTM** | Long Short-Term Memory |
| **MVE** | Missing Value Estimation |
| **RNN** | Recurrent Neural Network |
| **SSM** | State Space Models |
| **UDF** | User-Defined Function |

# 1 Introduction

Time-series analysis is the task of modeling sequential data where the sequence reflects the unfolding of time. In the fashion industry, these time-series may arise in various scenarios, such as in reference to customers (clicks-per-customer on types of item), in reference to market demand (purchases fluctuating over time in a given sector), or in reference to products (number of mentions over time of specific products on social media channels). This data enables many different types of analytics that are of high relevance to the fashion industry.

Following [16], we distinguish between four main tasks in time-series analysis:

- **Forecasting (F)**: The first task is to predict a subsequent value of a quantity given past values. A crucial example in the fashion industry is to forecast the popularity of individual fashion items, to make better stocking decisions and to better cope with the phenomenon of fluctuating sales.
- **Distribution Modeling (DM)**: The second task is to model the underlying processes in such a way as to produce mathematical models of time-series distributions. Such models enable important use cases in the fashion industry such as checking how well a specific customer's data corresponds to modeling assumptions. Furthermore, subsequent data analytics pipelines can leverage these models to produce better analytics.
- **Dependency Analysis (DA)**: The third task is to better understand the relationship between multiple time-series. For instance, we might have one time-series of weather data and another time-series of sales data between which we aim to better understand the inter-relationships. Typically, such dependency analysis is performed understand the inter-relationship between diverse business sectors, such as country, gender and type-of-product, and how these evolve over time.
- **Missing Value Estimation (MVE)**: The fourth task is to impute missing or unavailable time-series points in otherwise observed data. Such data cleaning steps are relevant to industry use cases since in real-world applications there are typically various issues that may cause some data to be lost or corrupted. With missing value estimation, lost or unavailable measurements of specific customers can be inferred at a range of time points, thus enabling better processing and analytics of this data.

As these examples illustrate, each of these four tasks is of high relevance to the fashion industry. Accordingly, we place high importance onto time-series analysis in the FashionBrain project, as outlined in the strategy deliberable D1.2.

**Limitations.** However, there are a number of limitations to traditional approaches to time-series analysis that make application to fashion industry scenarios prohibitively difficult. First, current time-series models are often overly simple, e.g. employing strong linear and/or Gaussian assumptions [1, 13, 21]. However, the time-series data we work with in FashionBrain, for example transactional click-data, exhibits a high-degree of non-linearity and non-Gaussianity.

Second, current models usually ignore changing trends over time, i.e. non-stationarities and cannot be adapted to model long-range dependencies [13, 18, 27]. However, fashion data is intrinsically seasonal and responds to emerging trends, meaning that non-stationarity must ideally be incorporated into models.

Finally, traditional models lack a probabilistic component, not yielding estimates with confidence estimates of their certainty [11, 21]. Weighted decisions, which allow a strategy to be applied probabilistically, in the absence of certainty, is vital to yielding financially optimal decisions.

## 1.1 Scope of this Deliverable

With this deliverable, we present a set of methods for time-series analysis designed to address the above-mentioned limitations. In particular, we use a state-of-the-art probabilistic state-space model based on Recurrent Neural Networks (RNNs) and an accompanying training framework. The presented framework is self-contained and unified as to admit plausible incorporation into MonetDB within the scope of the project. We accomplish this by using an RNN model which has has the universal approximation property (i.e. it is highly flexible) and may be trained in a unified manner to solve each of the four time-series tasks; the only difference between the tasks consists in data-preparation.

We make the framework created in the scope of this deliverable freely available as an open-source library called ProbRNN[1] and provide a set of code examples on how to execute different training and testing steps in this deliverable report. The library may be used as a basis for further work in the project, and may serve as a blueprint for MonetDB solutions in time-series analysis.

## 1.2 Placement of this Deliverable within FashionBrain

In this deliverable, *D1.4 "Software Requirements: SSMlibrary for time series modelling and trend prediction"*, we present a library of time-series modeling and trend prediction that we use as basis for our research into fashion industry time-series analytics.

[1] https://github.com/zalandoresearch/probrnn

As such, the requirements for this demonstrator are derived from the business scenarios identified in *D1.2 "Requirement analysis document"*, in particular:

- Scenario 4: Time-Series Analysis
  - Challenge 7: Textual Time Trails

## 1.3 Outline of this Deliverable

This deliverable is structured as follows: We first give an overview of classic approaches to time-series modeling and illustrate their limitations. We then propose a solution based on recurrent neural networks (RNNs) and show how they can be trained and applied to different time-series tasks. Finally, we discuss our open source library that implements the proposed approach and present code examples.

# 2 Background

A time-series [16] is an indexed sequence of random variables $X_1, X_2, ..., X_t := X_{1:t}$ which is distributed according to some underlying join probability distribution $X_{1:t} \sim P(X_1, X_2, ..., X_t)$. Typically several sample paths $X_1^i, X_2^i, ..., X_t^i$ of such a time-series are observed, for $i = 1, ..., n$ where $t$ is an arbitrary end time point. These observations are used either as training data to train time-series analysis approaches, or as evaluation data to test their ability to perform analytics.

Traditionally, time-series methods for forecasting, distribution modeling, dependency analysis and missing value estimation have typically drawn from one or more of the following restrictive assumptions:

**AS1** The relationships between data points in a time series are linear, allowing their relationships to be modeled with linear methods to produce a best estimate.

**AS2** The noise innovations around the true model are always assumed to follow a standard Gaussian distribution as opposed to other distributions.

**AS3** The marginal distributions across time are constant (*stationarity*).

**AS4** Time-dependence ranges over a finite number of time-steps, meaning that there is a hard limit on the length of the time series that is captured in models. In other words, long-range dependencies (i.e. between data points that lie far apart in time) are not captured.

**AS5** Negligible loss in performance is achieved by neglecting the confidence of an estimate.

In the following, we discuss previous work in each of the four time-series analysis tasks with respect to these restrictive assumptions.

## 2.1 Forecasting (F)

Formally, forecasting consists of producing a good estimate of $E(X_t|X_1, ..., X_{t-1})$, or more generally $P(X_t|X_1, ..., X_{t-1})$, i.e. to predict the probability of value $X_t$ based on all previous observations from time-steps $t-1$ through 1. The simplest of models for time-series forecasting is the linear autoregressive model [16]:

$$X_t = \sum_{i=1}^{k} X_{t-i} + c + \epsilon_t$$

and the moving average model [16]:

$$X_t = \sum_{i=1}^{k} \theta_i \epsilon_{t-i} + c + \mu_t$$

Here, the Greek letters apart from $\epsilon$ are parameters of the model and $\epsilon$ is typically assumed to be white noise. Both cases assume a fixed window $k$ of previous observations that is taken into account when predicting the current value, meaning that any observations that lie outside this window are not considered. These approaches thus cannot model long-range dependence (AS4). Both approaches are also clearly linear and Gaussian, meaning that they are affected by restrictive assumptions AS1 through AS3. In addition, most implementations of these approaches are optimized for quick inference and do not return confidence estimates, meaning that these approaches are also affected by AS5.

To address these limitations, previous work has proposed a number of extensions to these basic methods [16, 20, 2, 4, 22]. In particular, the approach proposed in [16] combines the advantages of linear autoregressive and moving average models, thus is more flexible, but still subject to AS1 to AS4. The approach presented in [20] extends this approach using a specific non-stationarity assumption, thus partially solving AS3, but is still subject to the remaining assumptions. [2] is another extension to [16] that partially solves AS5, but in a parametric fashion.

The approach proposed in [4] adds a property know as "autoregressive conditional heteroskedasticity" to solve the non-stationarity problem in a slightly different way tailored especially to financial data. It solves the non-linearity problem using a highly specific model of non-linearity. Nevertheless, all proposed extensions still contain an implicit Gaussianity assumption, meaning they are subject to restrictive assumption AS2.

## 2.2 Distribution Modeling (DM)

Formally, distribution modeling consists of estimating $P(X_1, X_2, ..., X_t)$, i.e. estimating the probability of occurrence of a sequence of data points. Such a probability estimation could for instance be used to determine how plausible or typical an observed sequence of data points is.

Standard approaches to distribution modeling make parametric assumptions about the joint distribution $P(X_1, ..., X_t)$. For example, one may assume that $X_{1:t}$ is distributed according to a multivariate Gaussian [12] as in $X_{1:t} \sim N(\mu, \Sigma)$. More flexible approaches relax the strict Gaussianity assumption, for instance assuming a mixture of Gaussians [3] as $X_{1:t} \sim \sum_i w_i N(\mu_i, \Sigma_i)$ or by assuming a fully non-parametric model, using for instance a kernel density estimator [23].

Although these latter methods are in principle flexible enough to handle arbitrary distributions, they do not scale well to high dimensional data. The former methods, on the other hand are overly restrictive.

## 2.3 Dependency Analysis (DA)

Dependency analysis consists in estimating values of one time-series from another $P(Y_t|X_{1:t}, Y_{1:t-1})$ to model the inter-dependencies between time-series, i.e. given time-series $Y_t|X_{1:t}$ we wish to estimate the probability of $Y_{1:t-1}$. Similar to distribution modeling, traditional approaches to dependency analysis use highly parametric assumptions about the type of of dependency, assuming linearity (correlation analysis) [6], Gaussianity (Pearson correlation analysis) [6], or specific assumptions about parameters [25, 28]. This means they are subject to restrictive assumption AS2.

Few of these dependency analysis methods fully relax the linearity assumption, allowing for arbitrary dependencies to be modeled - general methods such as mutual information require a huge dataset size to be of practical interest [7]. Moreover, few of these methods are specifically tailored to time-series analysis, where the dependency between time-points can hamper analysis of the strength of dependencies. This means that restrictive assumptions AS1 through AS4 generally hold for traditional approaches to dependency analysis.

## 2.4 Missing Value Estimation (MVE)

In missing value estimation we are given a sample path with missing values indicated by question marks in the following series:

$$X_1^i, X_2^i, ?, ?, X_5^i, ?, ..., X_t^i$$

The task is to estimate these missing values. There are two varieties of missing value estimation methods: parametric and non-parametric. In the parametric case, one typically assumes a model, and performs Expectation Maximization (EM) to estimate the missing values, performing updates using the modeling assumption [9].

In the non-parametric or semi-parametric framework typically one assumes that the data-generating distribution is degenerate in some sense and uses this degeneracy assumption to obtain estimates for missing values which are valid given for a range of models; this variety includes compressed sensing [10] and low-rank matrix completion [5].

All four tasks in time-series analysis are thus traditionally affected by these 5 limiting assumptions.

# 3 Method

We now outline our suite of time-series algorithms. Let $f(X_t, h_t)$ be the recurrence relation of a Long Short-Term Memory (LSTM) [17], with an embedding layer to the input of the LSTM. Then we make the following approximation:

$$P(X_t|X_1, ..., X_{t-1}) \approx P(X_t|X_{t-1}, h_t)$$

where:

$$h_t = f(X_{t-1}, h_t)$$

This means that instead of deriving the probability of $X_t$ from all previous observations $X_1, ..., X_{t-1}$, we only derive from the observation at the previous timestep $X_{t-1}$ together with a current "hidden state" $h_t$ of the LSTM. This hidden state is recursively produced from previous timesteps and thus implicitly captures the time-series up to the current point.

We use LSTMs to model time-series because they have several properties that allow us to avoid the restrictive assumptions AS1 through AS5: First, they are capable

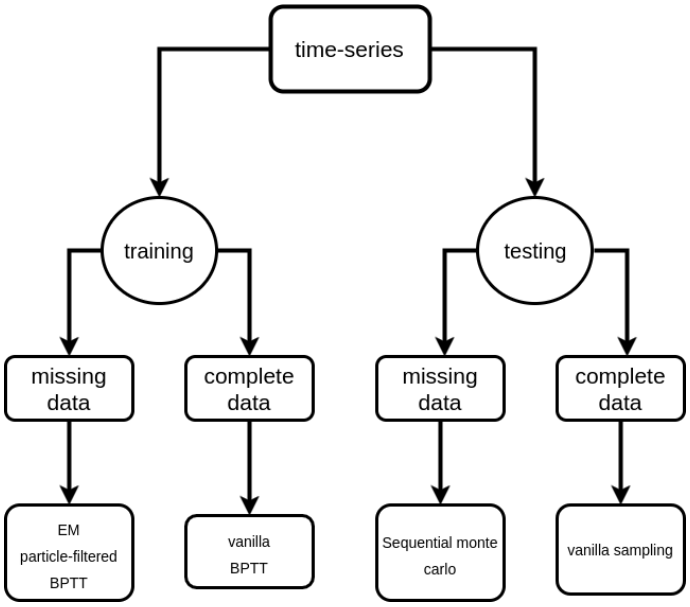


**Figure 3.1:** Training and evaluation methods depending on whether the time-series data is complete or has missing data.

of modeling arbitrary non-linear relationships [24] and thus are not restricted to simple assumptions of linearity (AS1). Second, they may cope with arbitrary noise distributions (i.e. not only Gaussian noise) by leveraging a binning approach and a softmax layer on the output [14] and so address restriction AS2. Third, they do not a priori assume stationarity, since the time-step is implicit in $h_t$, allowing non-constant distributions across time (AS3).

Importantly, LSTMs have shown that they can model long range dependence in a manner superior to traditional neural networks, potentially spanning hundreds of time-steps [17]. They are thus well suited to address restrictive assumption AS4 and model long distance temporal dependencies. Finally, they may be parameterized in a probabilistic manner as described above [19, 15].

## 3.1 Model Training

To train and test the LSTM, we use time series observations that are either complete or imcomplete, meaning they are either of the form:

$$X_1^i, X_2^i, ..., X_t^i$$

for complete observations, or of the form:

$$X_1^i, X_2^i, ?, ?, X_5^i, ?, ..., X_t^i$$

for incomplete observations, i.e. series with missing values. The schematic of this training and testing approach with regards to complete or incomplete data is shown in Figure 3.1. Depending on the data type, we use a different learning approach: If the training data is complete, we use a standard backpropagation through time (BPTT) approach. If it is incomplete, we instead use a particle-filtered version of backpropagation through time that marginalizes over missing values and thus allows training. In both cases, we train the network $f$ to model $P(X_t|X_{t-1}, h_t)$. For testing, if the data is complete, we use a standard sampling approach, while using a sequential monte carlo sampling method to deal with missing values.

With these training and testing methods, an LSTM can be applied to perform all four time-series analytics tasks seen in Figure 3.1. The application of this approach is straightforward, as follows:

**Forecasting (F)** For forecasting, the available training data is split into windows $X_1^i, ..., X_t^i$ that correspond to a maximum length of time-steps we wish to backpropagate. We train the neural network to maximize $P(X_t^i|X_{t-1}^i, h_t)$. If the windows are consecutive in time, the final hidden state from a previous window is passed to the next window as the initial state. This approach, known as "truncated backpropagation through time" ensures that training remains computationally effective, while enabling long range dependencies to

be modeled since the hidden state is passed on from window to window. The approach can thus learn dependencies that lie outside the bounds of one specific window.

**Distribution modeling (DM)** For distribution model, we train the LSTM in the same way as for forecasting, but with a learnt and constant initial hidden state. The approximate joint distribution is then given by:

$$P(X_{1:t}) = \prod_{i=1}^{t} P(X_i|X_{1:i-1}) \approx \prod_{i=1}^{t} P(X_i|h_i)$$

This approach is known as neural autoregressive distribution estimation [26].

**Dependency analysis (DA)** For dependency analysis between multiple sentences, we interleave the data points at aligned time points to create the following sequence:

$$X_1, Y_1, X_2, Y_2, ..., X_t$$

We then train the LSTM to predict the next item in a sequence based on all previous observations with exactly the same approach as used to train the forecasting model. This gives us a unified model that models a distribution that captures both time-series. In order to use this model at inference time, we then apply a particle filtering approach using sequential importance resampling [8] to predict $Y_t$ from previous values.

**Missing value estimation (MVE)** In order to use an LSTM to infer missing values from a sequence, we employ a particle filtering approach in combination with expectation maximization. Particle filtering is an efficient method for sampling probable sequences according to the trained LSTM and the partially observed data points. We use this approach to maximize the probability of the observed sample paths and thus infer the missing values that are most probable in these paths.

This illustrates that once we have trained the LSTM with either a default or a particle-filtered version of backpropagation through time (depending on whether the data is complete or incomplete, see 3.1), we can use the trained model to perform a variety of time-series analysis tasks.

## 3.2 Code Release

To better illustrate the approach of using LSTMs to perform forecasting, distribution modeling, dependency analysis and missing value estimation with time-series data, we release the software library PROPRNN[1] which we created in our FashionBrain work. In the following, we give a quick outline of how to use the library. More information can be found on the respective GitHub page.

[1] https://github.com/zalandoresearch/probrnn

To install PropRNN, simply clone the GitHub repository and execute '`make install`' on a local machine. This can be accomplished by issuing the following commands in a Unix console:

```
git clone git@github.com:zalandoresearch/probrnn.git
cd probnn/
make install
```

In order to verify if the installation was successful, execute the following lines in the console:

```
make clean
make test
```

**Generating training and testing data.** Now, open a Python environment to set up experiments. To set up a forecasting experiment, we first generate some time-series data, which can be done with the following script:

```
from probrnn import data
import numpy as np
x = np.random.randn(100000)
datastruct = data.TimeSeries(x)
```

To set up a distribution modeling experiment, we first generate some distributions, which can be done with the following script:

```
from probrnn import data
import numpy as np
x = np.random.randn(1000, 10)
datastruct = data.NadeWrapper(x)
```

Finally, to set up a distribution modeling experiment, we generate two time series. This can be done with following script:

```
from probrnn import data
import numpy as np
x = np.random.randn(1000)
y = np.random.randn(1000)
z = np.zeros(0000)
z[::2] = x
z[1::2] = y
datastruct = data.TimeSeries(z)
```

**Initalizing models.** The library offers helper methods to initialize models for forecasting or distribution modeling. The following Python script initializes a time series prediction model:

```
from probrnn import models
model = models.TimeSeriesPrediction(datastruct)
```

The following Python script initializes a distribution estimation model:

```
from probrnn import models
model = models.NADE(datastruct, params=params)
```

**Running experiments.** To execute a training run, instantiate the `models.Training` class and pass it variables to indicate where to save the model and training log files. The following script instantiates the class and executes a training run:

```
training = models.Training(model,
                           "test_model",
                           "test_log.json")
callback =
    lambda err, i, _: print "loss: {err};".format(err=err)
training.train(callback)
```

If the training data contains missing values, standard backpropagation through time cannot be used. For this, we implemented a class that performs the particle filtered version of backpropagation. It can be instantiated and used for training using the following code:

```
from probrnn import inference
imputer = lambda a, b: inference.NaiveSIS(a, b)
training = models.Training(
model,
"test_model",
"test_log.json",
imputer=imputer
)
training.train(callback)
```

If the test data contains missing values that we wish to fill in at testing time, we use a special `imputer` class to perform the missing value imputation, as discussed above. This can be run by issuing the following code:

```
x[np.random.choice(len(x), replace=False, size=50)] = np.nan
estimate = imputer(model, x).estimate()
```

Further code examples can be found on the GitHub page. This library illustrates how LSTMs can be used for various time-series tasks and may serve the consortium as a basis for discussion for further time-series research and development.

# 4 Conclusion

With this deliverable, we provided an overview of traditional approaches to time-series analysis and illustrated their limitations with regards to five restrictive assumptions. We then outlined a solution to these limitations using recurrent neural networks –in particular LSTMs– and showed how LSTMs can be used to deal with incomplete data during training, how they can be used to infer missing values in sequence data, and how they can be used to perform dependency analysis of different time series. We furthermore publicly release an open source library that implements these approaches. We argue that the simplicity and generality of this approach may serve as a blueprint for discussion of time-series technologies.

# Bibliography


[1] Hirotugu Akaike. Fitting autoregressive models for prediction. *Annals of the institute of Statistical Mathematics*, 21(1):243–247, 1969.

[2] Michael A Benjamin, Robert A Rigby, and D Mikis Stasinopoulos. Generalized autoregressive moving average models. *Journal of the American Statistical association*, 98(461):214–223, 2003.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3):307–327, 1986.

[5] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009.

[6] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[7] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[8] Nando De Freitas, C Andrieu, Pedro Højen-Sørensen, M Niranjan, and A Gee. Sequential monte carlo methods for neural networks. In *Sequential Monte Carlo methods in practice*, pages 359–379. Springer, 2001.

[9] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[10] David L Donoho et al. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

[11] James Durbin and Siem Jan Koopman. Time series analysis of non-gaussian observations based on state space models from both classical and bayesian perspectives. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(1):3–56, 2000.

[12] David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

[13] Clive WJ Granger. Causality, cointegration, and control. *Journal of Economic Dynamics and Control*, 12(2-3):551–559, 1988.

[14] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[15] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech

recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[16] James Douglas Hamilton. *Time series analysis*. Princeton Univ. Press, Princeton, NJ, 1994. ISBN 0691042896.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Søren Johansen. Statistical analysis of cointegration vectors. *Journal of economic dynamics and control*, 12(2-3):231–254, 1988.

[19] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[20] Alan Pankratz. *Forecasting with univariate Box-Jenkins models: Concepts and cases*, volume 224. John Wiley & Sons, 2009.

[21] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[22] Peter M Robinson. *Time series with long memory*. Advanced Texts in Econometrics, 2003.

[23] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pages 832–837, 1956.

[24] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

[25] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[26] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

[27] Paul Von Bünau, Frank C Meinecke, Franz C Király, and Klaus-Robert Müller. Finding stationary subspaces in multivariate time series. *Physical review letters*, 103(21):214101, 2009.

[28] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.