Ca' Foscari
University
of Venice

Master's Degree programme

in Data Analytics for Business and Society

Final Thesis

Bayesian Neural Networks

For time series forecasting

Supervisor
Ch. Prof. Roberto Casarin

Assistant supervisor
Ch. Prof. Davide Raggi

Graduand
Marco Solari
Matriculation Number 875475

Academic Year
2023 / 2024

Table of contents

1 Introduction

Artificial Neural Networks are parametric models that mimic the mental functions of higher life forms, emulating how the brain's pathways can create connections that allow them to recognize and interpret complex patterns. This approach, while at the first stages not fruitful, eventually brought a wave of new modeling and computational approaches in the fields of machine learning and artificial intelligence, trickling down from there and expanding in all fields for which the ability of these models to approximate almost any function unlocked powerful and useful application, from self-driving cars to Large Language Models.

However, these models show an inherent weakness: working as a black box, they fail to take into account *uncertainty*; hence, they are fragile in applications that involve stochasticity and for which overfitting and lack of generalization hinder their applicability, in particular in the high-risk scenarios for which their approximation capabilities would make them an excellent and versatile approach.

This thesis explores how Bayesian inference can be applied to Neural Networks to estimate the *epistemic uncertainty* associated with their tuning and predictions: it is a powerful framework that unbinds them from the limitation of the point forecast and unlocks *probabilistic* forecasts. By applying the Bayes theorem and treating their parameters as *stochastic variables*, an *ensemble* of Neural Networks can output a whole posterior predictive distribution. Moreover, this framework is robust to overfitting and overconfidence, addressing one of the most important issues of the standard, functional approach.

The perceptron is a very versatile model: applications are found in many fields and specialized variations of the basic structure abound, allowing it to perform diverse tasks such as computer vision and reinforcement learning. This thesis will be focused on the Recurrent Neural Network, a specialized architecture that excels at learning *sequences*, and will test its applications on *time series forecasting* with both *simulated* and *real data*; the latter, sourced from Bloomberg, will challenge the Bayesian Recurrent Neural Network with forecasting dynamically the volatility of the €/$ currency exchange rage and of the S&P 500 data, using high-frequency trading data sampled at the frequency of 1 minute.

This thesis will be organized as follows: Section 2 introduces the main concepts and history of the base model, the Artificial Neural Network and in particular the multi-layered perceptron (Section 2.1), its strengths and limitations (Section 2.2), and the often overlooked sources of uncertainty that this framework generates (Section 2.3).

Section 2.4 will argue that sequential data, such as time series, need a specialized architecture to respect the *ordering* of the time-dependent variables; moreover, Section 2.5 and Section 2.6 analyze the Long Short-Term Memory perceptron and its ability to store past information into hidden states, allowing it to effectively model longer-time dependencies, thus making it the undiscussed champion of time series forecasting, being the most used model for a variety of economic and financial applications.

Section 3.1 delves deeper into the limits of the standard, functional architecture by introducing the Bayesian Neural Network and the stochastic models that underpin all the following analyses, while Section 3.2 and Section 3.3 address the necessary *approximation techniques* that allow computing the posterior predictive of the network, which otherwise would be tractable only in the simplest of models and scenarios, therefore allowing applications to multidimensional models and big data.

While the first applications of the Bayesian Recurrent Neural Network can be traced to Fortunato, Blundell, and Vinyals (2017), Section 2.7 and Section 3.4 explore the more recent development of Bayesian Neural Networks, focusing on time series forecasting. This literature review is followed by a battery of simulations (Section 3.5) that aims at visualizing and summarizing the theoretical properties of these stochastic models and the approximation methods, exploring deep learning architectures and applying them to two different stochastic processes.

The last section exits from the theoretical and simulated perspective to challenge the Bayesian Neural Network with a demanding scenario: forecasting dynamically high-frequency time series. Section 4.2 is based on €/$ exchange rate data, based on samples starting from 01/12/2023 and ending on 03/02/2024, while Section 4.3 will be about forecasting the volatility of the S&P500 index using data from 02/02/2020 to 26/03/2020: both settings have their challenges, and in particular the latter is focused on the days following the systemic shock of COVID-19, a window in which volatility spiked and sudden downward shifts brought havoc in the market.

Section 5 will summarise the result of all forecasts, linking the theoretical exposition to the results and presenting the conclusions of this work.

All the code for this thesis has been written in `R` (R Core Team 2023) using `RStudio` (Posit team 2024); in particular, all charts are made with the graphic library `ggplot` (Wickham 2016), while the networks diagrams are coded in LaTeX with `TikZ` (Tantau 2013).

All the Bayesian Neural Networks use `Julia` (Bezanson et al. 2017) as their backend and in particular the package `BayesFluxR` (Wegner 2023).

2 Neural Networks

This chapter introduces the main concepts behind Neural Networks, with a focus on turning them into statistical models. Section 2.1 and Section 2.2 present their structure and functional model, showing their limits and in particular their inability to quantify uncertainty. Section 2.3 presents the sources of uncertainty. Section 2.4, Section 2.5, and Section 2.6 introduce a specialized Neural Network architecture, the logic behind it, and why it excels in *sequence* modeling, thus being the most natural candidate for forecasting time series, in which the *time* dimension has a natural ordering that yields fundamental information that should not be discarded. To conclude, Section 2.7 reviews more recent architectures developed for this application.

2.1 Multi-layered perceptron structure and functional model

The first approximation of a "*neuron*" as a computational unit able to solve complex tasks was proposed in the 1940s with the paper "Logical Calculus of Ideas Immanent in Nervous Activity" (McCulloch and Pitts 1943), breaking new grounds:

> "What was novel [...] was a theory that employed logic and the mathematical notion of computation - introduced by Alan Turing (1936-37) in terms of what came to be known as Turing Machines - to explain *how neural mechanisms might realize mental function*". (Piccinini 2004, 175)

At its core, a neural network is a parametric model capable of capturing functions with arbitrary accuracy: it is a non-linear function $f(X)$, where $X = \{X_1, X_2, ..., X_p\} \in \mathbb{R}^p$ is a vector of p variables, trained to predict a response variable Y. What sets this mathematical model apart from many mathematical models is its peculiar *layered* structure, as represented in Figure 2.1.

Figure 2.1: *An abstract representation of a single-layered Neural Network architecture*, with $K = 5$ neurons and $p = 4$ input variables. The *input* layer contains as many neurons as the number of independent variables fed into the activation functions A_k, which compose the second *hidden* layer. The last *output* layer might also contain many neurons: as a matter of fact, the multilayered perceptron is among the few models that natively can output a multivariate response.

Each "*neuron*" or "*perceptron*" represents a function that behaves as a computational unit; a set of perceptrons is densely stacked to make a *layer*. As an example, a basic structure such as the one depicted in Figure 2.1 is made of an *input layer*, to which each of the *variables* or *features* is fed and is passed forward through the connected network of one or more *hidden layers*, also composed of K neurons stacked together. This computational architecture can be expressed mathematically as:

$$\begin{aligned} f(X) &= \beta_0 + \sum_{k=1}^{K} \beta_k A_k^{(p)}(X) \\ &= \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j) \end{aligned} \tag{2.1}$$

where $A^{(j)} = g(A^{(j-1)})$, for $j = 2, \dots, p$ and $A^{(1)} = g_1(X)$.

Figure 2.2 represents the mathematical operation happening during the flow of information from one layer to a single activation, also introducing the common notation used for these models: the superscript p, as in $A^{(p)}$, indicates the layer whom the neuron belongs to, while the subscripts of the weight, as in $w_{m,n}$, points to the destination layer m and the original activation n.

The functions $A_k^{(j)}$, for $K = 1, \dots, K$, are the so-called nonlinear "*activations*". The nonlinearity of $g(X)$, with g_j being a map from $\mathbb{R}^{m_j} \to \mathbb{R}$, for $j = 0, 1, \dots, p$, is essential; otherwise, the model will revert to a multivariate linear regression. Figure 2.3 shows some examples of well-known and used activation functions. They receive as input the *weighted sum*, adjusted by a *bias*, of each of the preceding layer neuron's output, and return a single number. The artificial neural network learns by *scaling* the connections and applying *biases* to them as the stream of data is passed forward into the activations, resulting in an output that travels from one layer to the next: the final prediction Y is fed to a *loss function* $J(w_t; X, Y)$ that allows to iteratively learn patterns and structure in the data. The choice of the loss function depends on the application: for example, a *mean squared error* function will lead to forecasting the *mean* value of the distribution, while the *mean absolute error* forecasts the *median*.

Stated differently, this means that the network discovers the optimal weights and biases to accomplish the given task by minimizing the loss. The chosen loss function is algorithmically minimized through *gradient descent*[1] and *backpropagation*.

[1]In a *frequentist* interpretation of the learning process, this is equivalent to an MLE or MAP estimate.

Figure 2.2: *Feedforward activation* for a single neuron. This figure unpacks the notation of Equation 2.1 for a single activation. The weights w_{lj} and biases $\beta_j^{(0)}$ are applied to each connection and fed to a nonlinear function $g(\cdot)$. Some connections matter more than others: hence, the weights can be conceptualized as a signal of the relative importance of a particular connection, while the biases make the activation meaningful only if it reaches a certain threshold.

Figure 2.3: *Activation functions* examples. For the original perceptron, Rosenblatt (1957) chose the *sign* function; a choice discarded by later development because of its derivative being equal to 0 (Goan and Fookes 2020, p 6). When using the *logistic function* (*sigmoid*) in a single-layered network as Figure 2.1, Equation 2.1 is equivalent to a logistic regression. This function has been a common choice, along with the *Hyperbolic Tangent*, as the activation $g(\cdot)$ in the earliest investigations of neural network training, a choice justified by its grounding in probability theory and statistical learning; however, it was discarded because of the so-called *vanishing gradient* problem, caused by having gradients in the range $[-1, 1]$, which caused inefficient and slow learning. They have been replaced by the *REctified Linear Unit* (RELU) and its *leaky* variation in gradient-based learning such as backpropagation.

The *gradient descent* consists of an iterative update of the parameters as the networks learn from the available data by minimizing the error. In practice, the output of the model is computed for the current parameter settings (step t): partial derivatives for all parameters are found and then used to update each parameter, which is then used to repeat this algorithm in the following step (step $t+i$), until a given precision threshold is met and the performance of the model is not improving anymore. Mathematically:

$$w_{t+i} = w_t - \alpha \frac{\partial J(w_t; X, Y)}{\partial w_t} \tag{2.2}$$

Where α defines the *learning rate*, which is the magnitude of the step of that particular iteration of the gradient descent.

The *backpropagation*, introduced by Rumelhart, Hinton, and Williams (1986), is a *supervised learning* algorithm that updates the weights and biases for each layer of the network, from the output to the first hidden layer. The derivative of Equation 2.1 is computed through the application of the *chain rule*: the gradient calculation in the network moves *backward*, starting from the final layer's weights and ending with the first layer's weights. Partial gradient computations from one layer are recycled to compute gradients for the previous layer. This backward flow of error information enables efficient gradient computation at each layer, contrasting with the less efficient method of computing gradients for each layer separately.

Activations A_k with a value close to zero are *silent*, while those close to one are *firing*: all the nonlinear transformation happening inside this architecture allows to capture complex patterns and this is where the terminology refers specifically to the conceptual model that led to this computational structure: we are modeling the mechanism of a *biological* neural network, which learns by making connections and reacting to a specific phenomenon and by identifying complex, often nonlinear, patterns.

Metaphorically, each layer is an abstraction that allows mimicking the biological neurons that *activate* if its connection reaches a *meaningful threshold* and the learned pattern of *connections* recognizes learned patterns, creating specific sequences of activations that identify structure in the data.

2.2 Deep learning models as universal approximators and their limits

The initial model, pioneered by Rosenblatt (1957) is notably a hardware implementation of these concepts, designed to perform a basic binary classification task.

While the abstraction seemed powerful and the application promising, the fact that such a robot could not be trained to recognize a wide variety of patterns was immediately established: a paper by Minsky (1969) demonstrates that the single-layered network (the most basic structure) is only capable of learning *linearly separable patterns*. The initial intuition that brought together this peculiar computational architecture met a seemingly unbeatable obstacle, standing aside and letting more powerful and less expensive methods be at the forefront of research. Because of this, the field of neural networks stagnated for many years before it was realized that feed-forward neural networks with *two or more layers*, often known as *multilayer perceptrons*, can classify *non-linearly separable patterns* classes.

Hornik, Stinchcombe, and White (1989) demonstrated that a *standard multilayer feedforward network* utilizing *just one hidden layer* with a finite number of neurons is a universal approximator: provided that sufficiently many hidden units are available, this kind of model can approximate general mappings from one finite dimensional space to another, approximating continuous functions on a compact subset of $\mathbb{R}^n$.

This huge step forward led to the development of *deep learning*, named to indicate the multilayered structure of modern applications of neural networks, able to establish a new benchmark of accuracy on many different tasks, disrupting the fields of language modeling and computer vision: applications are endless.

Hornik, Stinchcombe, and White (1989, 360) attribute failures in applications, given that sufficiently many hidden units are available, to *"inadequate learning, insufficient number of hidden units, presence of a stochastic, rather than deterministic, relation between input and target."*. However, these are not the only shortfalls of deep learning.

When fed randomly generated noise, such a network will confidently output a result, even though the input contains only noise: while the metaphor of "picking up edges and patterns" is indeed useful in conceptualizing what is happening inside the algorithm, it is just an abstraction. A deep learning perceptron is a so-called *black box*: the connections it makes and the structure it learns are not a guarantee of precision and performance and are not interpretable; often, they are incomprehensible.

As an example, a multilayered feed-forward neural network trained on the MNIST dataset (Deng 2012), which can be considered the `"Hello World"` of deep learning applications consistently recognizes digits even if fed data randomly generated from a standard Gaussian distribution, notwithstanding the model accuracy on the test set of 97.46%. In other words, it seems that it recognizes a structure, the abstract "edges" and "patterns", even if there are none.

Figure 2.4: From Goan and Fookes (2020, 4), a comparison of a neural network to traditional probabilistic methods for a regression task, with no training data in the purple region. *On the left*: Regression output using a neural network with 2 hidden layers; *on the right*: Regression using a Gaussian Process framework, with a grey area representing $\pm 2\sigma$ from the expected value.

What is even worse, *it does not give any indication of the uncertainty associated with its prediction*; as a matter of fact, it is incapable of quantifying it. Quoting Jospin et al. (2022, 31):

> *"The final model might also generalize in unforeseen and overconfident ways on out-of-training-distribution data points. This property, in addition to the*

> *inability of ANNs to say 'I don't know', is problematic for many critical applications."*

While Hornik, Stinchcombe, and White (1989) proved the endless possibilities, they correctly pointed out that the model might often fail in the presence of a stochastic relationship between predictors and response: however, uncertainty and stochasticity are a natural aspect of many important phenomena for which such a powerful tool could be applied for predicting out-of-sample.

The lack of an indication of predictive uncertainty and its inability to approximate the underlying probabilistic data-generating process is an inherent pitfall of the *functional model* nested inside the deep layered perceptron: what is needed is a *stochastic model*.

2.3 Sources of uncertainty

Missing the capability to reason about uncertainty, ANNs show to possess an Achille's heel; while powerful and apt to a plurality of challenging applications, mimicking the computational abilities of higher life forms and their biological neurons, it is necessary to mutate the mathematical, algorithmic approach into a statistical model to give these models reliable uncertainty estimates, interpretability, and robustness.

Quoting Goan and Fookes (2020, 36):

> *"It is not possible to model all variables within [...] a highly complex system. This is accompanied by imperfect models and reliance on approximate inference, it is important that our models can communicate any uncertainty relating to decisions made. We must acknowledge that in essence, our models are wrong. This is why probabilistic models are favored for such scenarios; there is an underlying theory to help us deal with heterogeneity in our data and to account for uncertainty induced by variables not included in the model."*

Uncertainty can be understood as *aleatoric* or *epistemic*. In an example from Spiegelhalter (2019), the difference can be understood as the alea that exists in two different situations: before one coin is thrown and after when the result is hidden from the observer. In the first case, the outcome is determined by chance: aleatoric uncertainty. In the second case, the outcome is *determined*, it already happened: it is just unobservable.

Charnock, Perreault-Levasseur, and Lanusse (2020, 2) point out many different sources of uncertainty:

1. the choices in network architecture;
2. the methods for fitting networks;
3. the cuts in sets of training data;
4. the uncertainty in the distribution of realistic data;
5. and lack of knowledge about the physical processes that generate such data.

Some of these, are unavoidable and cannot be reduced through greater understanding: they are *aleatoric* sources of uncertainty, intrinsically linked to the stochastic nature of the observed phenomena. Some other aspects however can be investigated through a process of observation and understanding: for example, we can learn the property of the data-generating process, or find a distribution that describes how likely a particular outcome is, and test whether such a model is likely or not given the data at our disposal. This kind of uncertainty is *epistemic*, caused by a *lack of data*, and it is *reducibile*. An often misunderstood source of epistemic uncertainty is linked to the *hyperparameters* and overall design choices of the model: the design of the neural network, the way it is trained, the choice of the cost function, and all other aspects that determine the capabilities and architecture of the ANN.

While it is easy to conceptually separate these two classes, this does not happen with the network output:

> "*There is no separation between aleatoric and epistemic uncertainty and no knowledge of how likely (or well) a new example of data is to provide a realistic prediction. It is possible, though, to quantify this epistemic error caused by our lack of knowledge about the properties of a neural network, and characterizing this uncertainty can allow us to perform reasoned inference.*" (Charnock, Perreault-Levasseur, and Lanusse 2020, 5)

Treating ANNs as *statistical models* yields a powerful framework that allows such networks to make statements of inference, "*reducing the lack of trust that is inherent in the standard deep learning setup*". A seminal paper by Tishby, Levin, and Solla (1989) introduced two powerful concepts: firstly, it gave a statistical interpretation of the loss function, showing that minimizing the MSE is equivalent to finding the

MLE of a Gaussian. Secondly, it showed how the *Bayesian* paradigm provides a rigorous framework to analyze and train uncertainty-aware neural networks by specifying a *prior* distribution over the weights and learning the *posterior*; more generally, it paved the way to applying both traditional inferential techniques and specialized algorithms to support the development of probabilistic neural networks (Jospin et al. 2022, 30). An introduction to Bayesian Neural Networks is presented in Section 3.

2.4 Specialized architectures for time series

The deep multi-layer perceptron is a *general-purpose* architecture. While it can learn to forecast time series, it suffers from several limitations: in particular, the size of the input directly affects the number of parameters of the network, which might negatively affect learning speed and overall performance (Sezer, Gudelek, and Ozbayoglu 2020, 3). Moreover, time dependencies cannot be efficiently modeled because data are treated as *independent* observations and the initial data are *transformed*[2] during the learning process: the network is not able to replicate dynamic behavior over a time dimension as each observation is not sequentially processed and therefore fundamental information is lost. For example, as addressed by the papers from Ahmad et al. (2014) and Deb et al. (2017) regarding modeling daily energy consumption, it would be relevant to capture repetitive patterns that happen at a fixed frequency such as seasonality, or another time-dependent behavior such as a trend; the ANN processes the information in *batches*, or *blocks*, hence it is not able to recognize that the position of a particular observation in the time-line sheds light on an aspect of the phenomenon that needs to be captured to enhance the quality and precision of forecasts. Other applications of ANNs to economic time series analysis include *finance* (Chen, Pelger, and Zhu 2024), *insurance* (Kiermayer and Weiß 2021), and *macroeconomics* (Teräsvirta, Van Dijk, and Medeiros 2005).

Specialized architectures have been devised to properly capture the peculiarities of learning patterns from a dynamic, ordered, sequence of events. The most used is the Long-Short Term Memory (LSTM), which is built upon the Recurrent Neural Network. The latter is in itself a powerful framework for sequential data, but:

[2] In other words, the input is discarded as only its transformations are passed through the sequence of layers.

"LSTM and its variations along with some hybrid models dominate the financial time series forecasting domain. LSTM, by its nature, utilizes the temporal characteristics of any time series signal; hence, forecasting financial time series is a well-studied and successful implementation of LSTM." (Sezer, Gudelek, and Ozbayoglu 2020, 8).

However, being a variation of RNNs, an introduction to this foundational architecture is necessary, before venturing into the specifics of LSTM functional models.

2.5 Recurrent neural networks

Recurrent Neural Networks are a basic specialized functional model built for sequence modeling thanks to the introduction of *"hidden units"* or *"states"*, which store the output data that is fed to the following one in addition to the new input. In other words, the output of a layer can affect the input of the following one through *feedback*, creating a storage vector of internal states that allows this architecture to process sequences by accumulating information about the past.

This concept can be formalized mathematically through Equation 2.5, which is stylized graphically by Figure 2.5. A *third* dimension is appended to the bi-dimensional array of features (columns) and observations (rows); one of them represents the number of observations, with each characterized by a first dimension representing the *length* of the sequence of data and a second dimension for the number of feature for each step. This sequences of vectors $\{X_t\}_{t=1}^{T}$, with $l \in \{1, 2, \dots, p\}$ as the number of features and T representing a time index, can be written as:

$$\text{Time}\Big\downarrow \begin{bmatrix} x_{1,1} & \cdots & x_{p,1} \\ \vdots & \ddots & \vdots \\ x_{1,T} & \cdots & x_{p,T} \end{bmatrix} \tag{2.3}$$

As in Equation 2.1, A_k represents a non-linear *activation function* $g(\cdot)$ (Figure 2.3) and Y is the *final output* of the network; h_t, instead, is the *hidden state* at t that stores the output (a sort of "hidden prediction"), which is fed into the following activation at $t+1$ along with the corresponding input $\mathbf{x}_{t+1}$.

$$h_t = f(h_{t-1}, \mathbf{x}_t; \mathbf{W}, \mathbf{B}, \mathbf{U}) \tag{2.4}$$

Figure 2.5: *Recurrent Neural Network*. On the left-hand side, is the so-called *"folded"* representation, and on the right-hand side, is the *"unfolded"* structure of the recurrent processing happening inside this specialized architecture. The unfolded representation should not be confused with a sequence of layers, as in Figure 2.2; they are instead a representation of the *same network* but at *different steps in time*. The input is a sequence of vectors having a single component $\{X_t\}_{t=1}^{T}$ and the target is a single response. The same collections of weights are used as each element is processed and the output layer produces a sequence of predictions Y from the current activation A_T, but typically only the last of these is of relevance.

W, **U**, **B** are matrices that store the network parameters: the first for the *weights* of the *input* layer, the second for the *hidden-to-hidden* layers, the third for the *output* layer. In this architecture all parameters are shared and updated when new observations are available, and therefore used to process each element of the series: this introduces an element of *persistence*. We can write the activation function as:

$$A_{lt} = g(w_{t_0} + \sum_{j=1}^{p} w_{jt}x_{jt} + \sum_{t=1}^{T} u_{lt}A_{l-1,t}) \tag{2.5}$$

In this function, an *accumulation* mechanism allows capturing any dependence between the observations, learning contextual information that can be used to capture information that enhances the forecast produced by the last occurrence, which is what we usually define as Y and that can be a sequence itself, allowing forecasts of longer time windows:

$$Y = \beta_0 + \sum_{t=1}^{T} \beta_{lt}A_{lt} \tag{2.6}$$

All the network parameters are updated using a recurrent algorithm called *Back-Propagation Through Time*: as in a standard ANN, a gradient descent iteratively minimizes a pre-selected *loss function*, following an algorithm similar to what has been described in Section 2.1, with the main difference being that the gradient is updated following a reverse sequential order, in parallel to what happens in the RNN when data are processed in the forward-pass of the learning phase.

2.6 Long-Short Term Memory

Introduced by Hochreiter and Schmidhuber (1997), Long-Short Term Memory (LSTM) is widely cited as the most common DL model for time series forecasting by Han et al. (2021), especially in the context of predicting financial data. Quoting Sezer, Gudelek, and Ozbayoglu (2020, 21):

> *"According to the publication statistics, LSTM was the preferred choice of most researchers for financial time series forecasting. LSTM and its variations utilized time-varying data with feedback-embedded representations, resulting in higher performances for time series prediction implementations.*

> *Because most financial data, one way or another, included time-dependent components, LSTM was the natural choice in financial time series forecasting problems.*"

This variation of Recurrent Neural Networks architecture has succeeded in many fields of applications because of two limiting weaknesses of the standard structure regarding an "insufficient, decaying error backflow" (Hochreiter and Schmidhuber 1997, 1):

1. *Vanishing gradient.*
2. *Exploding gradient.*

They result in the inability to model *long range* dependencies: over extended periods, the gradient for the weights, which are repeatedly multiplied by the hidden state at each time step, tends to either diminish significantly (known as the *vanishing gradient* problem) or increase enormously (known as the *exploding gradient problem*).

Han et al. (2021, 7836) states that RNNs often fail to learn information over 5-10 timesteps. However, many useful applications depend on the ability to model long-term relationships. The feedback connection that stores recent representations of events in the form of activations corresponds to a *short-term* memory, while gradually updating the weights over time is akin to a *long-term* memory. Both are present in the Long-Short Term Memory architecture (1997), which is recurrent, gradient-based, and (most importantly) enforces a *constant* error flow through the internal states of specialized units, the *memory cells*, composed of *input*, *output* and *forget gates*. This is also known as the Costant Error Carousel (CEC).

A separate *state* vector is used to determine whether to remember or forget what is inside the *hidden* state: the memory cells are stacked together to form an LSTM layer, which regulates the information flow, allowing the architecture to feedback the information over *arbitrary* time intervals. The input gate manages the information incorporated into the cell, the output gate controls the flow of information outward to the rest of the network, and the forget gate diminishes the activation of the preceding timesteps (Han et al. 2021, 7836).

The functional form of the forward pass can be described through the following set of equations: let f_t be the forget gate's activation vector, i_t the input gate's activation vector, o_t the output gate's activation vector, c_t the cell state vector, and keeping the same terminology of Equation 2.1 and Equation 2.5[3]:

[3] $\otimes$ represents the element-wise (Hadamard) product.

Figure 2.6: *Diagram of a Long-Short Term Memory unit Costant Error Carousel*, which replaces the ordinary recurrent node illustrated in Figure 2.5, with a detailed inside information flow at time t. A *hidden* state h_{t-1}, along with a *cell* state c_{t-1}, as computed one lag before, are combined to x_t and fed in parallel to different activation functions, such as the *sigmoid* σ and the Hyperbolic Tangent $Tanh$, to update the cell and hidden states at t, to be used in the next step of the recurrent model. Each flow is labelled accordingly to equations from Equation 2.7 to Equation 2.11.

$$f_t = g(W_f x_t + U_f h_{t-1} + \beta_f) \tag{2.7}$$

The *forget* gate determines if the internal state should be flushed.

$$i_t = g(W_i x_t + U_i h_{t-1} + \beta_i) \tag{2.8}$$

The *input* gate plays a crucial role in deciding whether a specific input should influence the internal state of the network. It governs the extent to which the value of the input node should be incorporated into the current memory cell's internal state.

$$o_t = g(W_o x_t + U_o h_{t-1} + \beta_o) \tag{2.9}$$

The *output* gate determines if the internal state of a given neuron should be allowed to impact the cell's output.

$$c_t = f_t \otimes c_{t-1} + i_t \otimes g(W_c x_t + U_c h_{t-1} + \beta_c) \tag{2.10}$$

$$h_t = o_t \otimes g(c_t) \tag{2.11}$$

This formulation creates a mechanism to determine when a hidden state should be updated and when it should be reset; this allows us to avoid updating the hidden state after if necessary and to skip irrelevant observations.

LSTMs are a specialized version of RNN: all the other aspects of this functional model are the same, starting from the hyperparameters to the use of BPTT to learn the correct weights and biases for the prediction task. Even if they are more advanced, they process data sequences in the same way and hence have the same structure to their layers, differing only in the specific operations it applies at each time step as detailed by Equation 2.7, Equation 2.8, Equation 2.9, Equation 2.10, Equation 2.11, while outperforming both the ANN and standard RNN in sequence modeling applications.

Deep learning models often use LSTM networks to tackle complex tasks such as speech and handwriting recognition. They are most commonly used in time-series data contexts such as Natural Language Processing (NLP), language modeling, translation, speech recognition, sentiment analysis, predictive analysis, and financial time series analysis (Sezer, Gudelek, and Ozbayoglu 2020, 8). By integrating both short and long-term memory, LSTM networks improve their efficacy in time series data interpretation, as demonstrated by their success in language translation: this is the reason why it is the most logical choice as the functional model for a Bayesian Neural Network able to effectively forecast time series.

2.7 Review of recent developments for Neural Networks

Along with these models, other structural variations of neural networks have been applied for forecasting time series; in this section, a brief examination of the characteristics of the most common models will be provided, following the taxonomy of Sezer, Gudelek, and Ozbayoglu (2020) and Han et al. (2021), along with a review of a more recent development represented by the *multi-head attention*-based architecture known as the *transformer* (Vaswani et al. 2017).

Convolutional Neural Networks (CNNs), developed by LeCun, Bengio, et al. (1995) building upon the work in neurobiology of Hubel and Wiesel (1959), are a class of ANNs in which specialized layers based on *convolutional operations* are used for feature extraction. As with many classes of NNs, their overall architecture follows the same structure of Figure 2.1; however, the introduction of *convolutional filters* and *pooling* layers allowed this model to set new benchmarks in the field of *computer vision*, arguably being the main protagonists along with LSTM for the so-called *Deep Learning revolution* of the late 2000s, in which DL-based models started to outperform other machine learning models.

While they have been applied to time series forecasting in both simulated and applied scenarios, they excel in classification tasks. Quoting Han et al. (2021, 7836):

> *"To the best of our knowledge, we found that more works utilize the CNN architectures for classification rather than prediction of the next value with multivariate time series. [...] Compared with the extensive application of image recognition, using CNN for time series prediction still requires further studies to demonstrate its superiority in learning deep features to obtain better performance."*

Figure 2.7: *Topic-models heatmap* from Sezer, Gudelek, and Ozbayoglu (2020, 21). On the horizontal axis is a list of topics related to financial time series forecasting; on the y-axis is a list of models; the cell color gradient details the number of applications of a given model for a specific topic. Recurrent Neural Networks (RNN) include the LSTM architecture and are by far the most used. *DMLP* stands for *Deep Multi-Layered Perceptron*, which is another term for the models described in Section 2.1 and Section 2.2.

The *Deep Belief Network* (DBN) consists of a stacked architecture of *Restricted Boltzmann Machines* (RBMs). RBMs are a class of stochastic models typically used in *unsupervised learning* (Qiu et al. 2014) for tasks such as dimension reduction, classification, feature learning, and collaborative filtering (Salakhutdinov, Mnih, and Hinton 2007). Their power comes from their capacity to *identify latent patterns without supervision*.

However, RBMs represent a training difficulty because there are no cost-effective methods for calculating the log-likelihood, despite the existence of good gradient estimators (Sezer, Gudelek, and Ozbayoglu 2020, 5); while they have been more widely applied than other models for predicting time series (Hrasko, Pacheco, and Krohling 2015), *"optimization techniques are still highly demanded to reduce the computational cost this deep model, especially for ones exhibiting ensemble structure"* Han et al. (2021, 7839).

In the paper *"Attention is all you need"*, Vaswani et al. (2017) created a novel architecture that, unlike the LSTM, does not have a recurrent structure; instead, it utilizes the positional encoding added in the *input embeddings* to model the sequence information in a so-called *"self-attention"* module, allowing it to capture long-range dependencies and extract *semantic correlations* among elements in a long sequence. The transformer is among the most promising models for sequence processing and it is already setting new benchmarks in *natural language processing*, computer vision, and *speech modeling* (Wen et al. 2022).

Still, as detailed in Zeng et al. (2023, 11126), its self-attention mechanism is *permutation invariant*, resulting in a loss of temporal information: the ordering of the data points, as argued in this chapter's introduction, contains information that should not be discarded. As a result, this very advanced model underperforms even simpler linear models in the context of long-term time series forecasting: hence, further research and experimental validation are needed before applying this model in real-world scenarios, as this thesis will do in Section 4.

3 Bayesian Neural Networks

These powerful functional models paved the way for many great achievements in the field of Artificial Intelligence. They are, however, still missing the ability to effectively model uncertainty. *Bayesian inference* is a powerful framework to transform multi-layered perceptrons into statistical models able to quantify uncertainty; furthermore, using a prior to integrating out the parameters to average across many models during training gives a *regularisation* effect to the network to avoid *overfitting* issues, which are common with deep learning models.

The following sections detail the main stochastic model behind the Bayesian Recurrent Neural Network, created by Fortunato, Blundell, and Vinyals (2017), before investigating the main applied Bayesian methods to train Artificial Neural Networks, allowing them to express uncertainty through their parameters.

Section 3.1 reviews the literature about Bayesian Neural Networks and details their statistical model. Section 3.2 introduces the Markov Chain Monte Carlo (MCMC) class of methods and their relevance to Bayesian deep learning, while Section 3.3 details one of the fundamental tools for obtaining approximate Bayesian estimates of the intractable evidence, the variational inference based *Bayes-by-backprop* algorithm.

While the original model employs variational inference to train the network (Fortunato, Blundell, and Vinyals 2017, 2), MCMC will also be used in the simulated scenarios to compare the relative performance and learning time, *ceteris paribus*. Section 3.4 reviews more recent developments in the Bayesian modeling of NNs and Section 3.5 concludes this section by showing the performance of a trained model in predicting both dynamically and statically with simulated datasets, before testing the model in a realistic scenario in Section 4.

3.1 From functional to stochastic models

A method for comprehending and measuring the uncertainty related to deep neural network predictions is provided by *Bayesian statistics*. In this new setting, the parameters of the network are treated as *random variables*: it is a logical choice as we do not know their true value, but we can use the available information, in the form of the observed data, to learn and discover a *distribution* of these parameters. Inferring the values of unknown model weights based on the available information involves solving the problem of inverse probability, which is accomplished through the application of the Bayes theorem.

Figure 3.1: *Comparison between regular and Bayesian Neural Networks.* While the overall architecture stays the same and is similar to the ANN represented in Figure 2.1, the main difference is in the way the parameters are interpreted. While in the regular setting the observed data are considered as generated from a random process and the parameters have a true value to be discovered, the Bayesian approach treats these latent parameters as random variables of which we want to learn a distribution, conditional on the what we can observe in the training data.

Since Bayesian and regular NNs share the same architecture, or *functional model*, activations, cost functions, layers, and overall modeling choices; (Figure 3.1), then we are still dealing with a non-linear, highly complex function. To avoid confusion, we will refer to the parameters of the BNN with the symbol θ.

Goan and Fookes (2020, 9) explains how the Bayes theorem allows us to learn the distribution of these parameters: a *stochastic model* is chosen when the weights are treated as *hidden* or *latent* variables. While we cannot directly observe their distribution, we can represent a distribution of a possible parameter choice θ in terms of observed probabilities, resulting in the distribution of model parameters *conditional* on the data we have seen. This is the *posterior distribution*: mathematically, $p(\theta|D)$. $D = (D_x, D_y)$ represents the observations, where D_x, D_y denote the training *inputs* and *labels*, respectively.

What we can observe *before* training the model is the *joint distribution* of data and weights, which is defined by our *prior beliefs* $p(\theta)$ in the predictive power of the model over the hidden variables and the *likelihood* $p(D_y|D_x, \theta)$ and can be written as:

$$p(\theta, D) = p(\theta)p(D_y|D_x, \theta) \tag{3.1}$$

The choice of architecture and prior comes into play by defining the likelihood term: in particular, the prior distribution should be specified to incorporate the belief as to how the weights should be distributed, before seeing any data. Putting everything together and assuming independence between the model parameters and the input, the Bayes theorem is then applied to obtain the posterior distribution over the model weights:

$$\pi(\theta|D) = \frac{p(\theta)p(D_y|D_x, \theta)}{\int_\theta p(\theta')p(D_x|D_y, \theta')d\theta'} \propto p(\theta)p(D_y|D_x, \theta) \tag{3.2}$$

where $\propto$ denotes the proportionality symbol. $\int_\theta p(\theta)p(D_x|D_y, \theta)d\theta$ represents the *evidence* that the parameterized distribution accurately describes the distribution of some event, or $p(D)$.

Bayesian Neural Networks hence are *stochastic artificial neural networks*: since they have *random variables* θ workings as parameters, they can obtain uncertainty estimates by comparing the predictions of multiple sampled model parametrizations; this peculiarity also makes them a special case of *ensenble learning*. The uncertainty will be higher when these predictions disagree; most importantly, it solves the conundrum detailed in Section 2.3, as Equation 3.2 offers a natural framework to distinguish between *aleatoric* uncertainty, or the probability of observing a given output given the input and the chosen parametrization of the model $p(D_y|D_x, \theta)$, and *epistemic* uncertainty, or the probability of the chosen parametrization given the available observations $p(\theta|D)$.

There is however a huge issue: the Bayesian posterior for a neural network architecture is a highly dimensional and non-convex function. In particular, the *evidence* is *intractable* both from an analytical and a computational point of view for all but the simplest cases, as it requires computing the marginal likelihood of every possible parametrization of the model.

Notwithstanding the intractability of $\int_\theta p(\theta')p(D_x|D_y, \theta')d\theta'$ that could potentially hinder the feasibility of this kind of stochastic modeling, *approximate* solutions can be obtained through specialized techniques. These techniques borrow either from traditional Bayesian statistics, such as:

1. Markov Chain Monte Carlo,
2. Variational Inference,
3. Empirical Bayes.

or from a re-conceptualization of deep learning algorithms, fine-tuned to allow for stochastic modeling; in particular, the following Section 3.3 and the model applied in Section 3.5 will use the Bayes-by-backprop, which adapts the gradient backpropagation of Equation 2.2 to a stochastic setting. Even if these solutions are indeed simplifications and lead to only approximate solutions, as detailed by Charnock, Perreault-Levasseur, and Lanusse (2020, 42), "*recent research has also shown that being only approximately Bayesian is sufficient to achieve a correctly calibrated model with uncertainty estimates*" (Jospin et al. 2022, 40).

Empirical Bayes, which in synthesis consists of learning the prior and posterior *afterward*, "*is a valid approximation when the dimension of the prior parameters being learned is significantly smaller than the dimension of the model parameters*" (Jospin et al. 2022, 42). It mainly deals with adaptations of algorithms such as the Bayes-by-backprop; hence, the following sections will only deal with VI and MCMC.

3.2 Markov Chain Monte Carlo and the Metropolis-Hasting algorithm

One of the main differences between BNNs and ANNs is in the *learning* phase. The performance of the non-probabilistic models is strictly dependent on it and, especially for sequence modeling, proper tuning is essential. As an example, choosing the dimension of the *window* of input data that are fed through a recurrent structure such as the one depicted in Figure 2.5 is fundamental in learning the correct weights and dramatically improves the performance of the model. Even for a standard feed-forward *dense* network a learning phase consisting of training the network on *batches* of data is an essential part of model fitting and requires time and effort.

With BNNs, instead, the learning phase is *not* required; the only requirement is sampling the posterior and doing model averaging (Jospin et al. 2022, 39). The following pseudocode details the steps of the algorithm which translates the statistical framework detailed in Section 3.1:

Algorithm 1 Inference procedure for a BNN

```
  Define $p(\theta|D) = \frac{p(\theta)p(D_y|D_x,\theta)}{\int_\theta p(\theta)p(D_x|D_y,\theta)d\theta}$;
  **for** $i = 1 \to n$ **do**
      Draw $\theta_i \sim p(\theta|D)$;
      $\mathbf{y}_i = \Phi_{\theta_i}(\mathbf{x})$;
  **end for**
  **return** $Y = \{y_i|i \in [0, N)\}, \Theta = \{\theta_i|i \in [0, N)\}$;
```

Because the *evidence* term can only be approximated, reliable predictions should be based on accurate approximations of the posterior (Goan and Fookes 2020, 17).

The most important approach to this solution is the Markov Chain Monte Carlo, a general class of methods for sampling arbitrary and intractable distributions. While Variational Inference (VI) prevails as the most common approach for BNNs specialized in time series forecasting, as can be seen from Table 3.1, it does not allow sampling from the *exact* posterior distribution, as detailed in Section 3.3; hence, a gain in efficiency and performance is traded with a lack of knowledge about how close the chosen approximation is from the desired target (Charnock, Perreault-Levasseur, and Lanusse 2020, 31). As argued in one of the papers that first proposed variational methods (Hinton and Van Camp 1993, 12), the best tool to sample from the *exact* posterior is the MCMC.

The main idea behind the MCMC family of algorithms is based on building a Markov chain of independent samples with such properties that they can be attributed as belonging to the posterior distribution, then applying Monte Carlo integration for prediction. *Ergodicity* is fundamental: it should be possible to move from any possible state to another in some finite number of transitions and no long-term repeating cycles should occur. However, each consecutive sample is correlated with the next; the transition probability is simply dependent on the prior state, and the chains exhibit long-range correlation. States from some target distribution can only be retained as samples if they are physically uncorrelated; hence, the initial steps are usually out of distribution and thus should be discarded.

In other words, convergence is *not* guaranteed unless an infinite number of steps is taken to ensure stationarity; this is the reason why we can only use this technique to obtain numerical approximations of the evidence.

Among all the MCMC methods, the most relevant one for BNNs is the Metropolis-Hasting algorithm (Chib and Greenberg 1995): its popularity derives from the fact that it only requires a function $f(x)$ that is proportional to the exact probability distribution $P(x)$ from which to sample from. This property allows us to deal with the intractable part of the elsewhere easy-to-compute $\pi(\theta|D)$ (Equation 3.2).

This algorithm starts with a random guess θ_0 and then samples θ' around it from a proposal distribution $Q(\theta'|\theta)$, usually a Gaussian, which works as a candidate point: if this point is more likely according to the target distribution, it is accepted; in the other case, it is either rejected or accepted with a certain probability (Jospin et al. 2022, 40). As the algorithm proceeds, it adapts its proposal distribution based on the past samples it has seen. If the proposed samples are consistently accepted or rejected, the algorithm adjusts its proposal distribution to explore the space more effectively: the algorithm repeats this process for many iterations, gradually exploring the space of possible values and generating samples from the target distribution. If Q is chosen to be symmetric, the acceptance probability p can be simplified and the formula of the acceptance rate becomes:

$$p = \min\left(1, \frac{f(\theta')}{f(\theta_n)}\right) \tag{3.3}$$

The following algorithm details the step of the iterations:

Over time, the samples generated by the algorithm become increasingly representative: the algorithm is designed to converge to the correct distribution, meaning that as the number of iterations increases, the samples become more accurate representations of the target.

The main limitation of this algorithm is that *"above a handful of parameters the computational time of Metropolis-Hasting becomes a limitation, meaning that it is not efficient for sampling high dimensional distributions such as the posterior distribution of neural*

Algorithm 2 Metropolis-Hasting algorithm.

```
Draw $\theta_0 \sim$ Initial probability distribution;
while $n = 0 \to N$ do
    Draw $\theta' \sim Q(\theta'|\theta_n)$;
    $p = \min\left(1, \frac{Q(\theta'|\theta_n)}{Q(\theta_n|\theta')}\frac{f(\theta')}{f(\theta_n)}\right)$;
    Draw $k \sim$ Bernoulli$(p)$;
    if $k$ then
        $\theta_{n+1} = \theta'$;
        $n = n + 1$;
    end if
end while
```

network parameters" (Charnock, Perreault-Levasseur, and Lanusse 2020, 19). This lack of scalability explains the popularity of variational methods for the approximate computation of the posterior, which will be introduced in the following section.

3.3 Variational Inference and probabilistic backpropagation

Even though MCMC algorithms are, at least in principle, the better approach, as they allow to sample the posterior and approximate it exactly given enough time, two motives explain the lack of widespread adoption of this class of algorithms.

The first consists of the requirement of an *initial burn-in time* before the Markov chain converges to the desired distribution. All samples until the Markov chain is *stationary* should be discarded because they can be out of distribution (Jospin et al. 2022).

The second is the fact that successive samples might be autocorrelated; to gather roughly independent samples from the underlying distribution, a large set must be created and subsampled. Moreover, This final collection of samples must be stored after training: this weighs negatively on the feasibility of these methods because of the computational resources required and undermines scalability to larger models.

Variational inference dominates the field of probabilistic Neural Networks for time series forecasting (Y. Wang et al. 2019, 10); the Bayesian Recurrent Neural Networks (Fortunato, Blundell, and Vinyals 2017) along with all more recent models listed in Table 3.1 use this approach to obtain estimates of the *evidence*.

Beal (2003) presents a unified variational Bayesian framework that approximates the intractable computations in models with latent variables using a lower bound on the marginal likelihood.

Quoting Goan and Fookes (2020, 13), *"the machine learning community has continuously excelled at optimization-based problems"*; VI frames the problem of *performing inference* as an *optimization problem*: having an intractable distribution as the one in Equation 3.2, variational approaches will attempt to solve an optimization problem over a class of tractable distributions Q to identify a $q_\phi \in Q$ that is most similar to π. Then, q_ϕ, called the *variational posterior*, is used instead of π to obtain an approximative answer: in practice, this is achieved by *assuming the form of the posterior distribution* and *performing optimization* to find the assumed density that is closest to the true posterior.

This *"closeness"*, or *similarity*, is measured using *relative entropy* through the Kullback-Leibler (KL) divergence. Based on Shannon's information theory (Shannon 1948), it measures the loss of information when approximating the posterior distribution $\pi(\theta|D)$ with the variational posterior q_ϕ and works as the *objective function* that needs to be minimized with respect to the set of parameters ϕ that characterizes a specific variational posterior. In the context of Bayesian inference, the KL-divergence is computed as:

$$\mathbb{KL}(q_\phi \| \pi) = \int_\theta q_\phi(\theta') \log \left(\frac{q_\phi(\theta')}{\pi(\theta'|D)} \right) d\theta' \tag{3.4}$$

where $\mathbb{KL} = 0$ implies that there is no loss of information, hence the two distributions are equivalent, and $\mathbb{KL} > 0$ indicates a degree of information lost. While computing the posterior is required to obtain this measure, in practice this is avoided thanks to the following derivation of Equation 3.4:

$$\int_\theta q_\phi(\theta') \log \left(\frac{q_\phi(\theta')}{\pi(\theta'|D)} \right) d\theta' = log(P(D)) - \mathbb{KL}(q_\phi \| \pi) \tag{3.5}$$

The resulting quantity is known as the *evidence lower bound* (ELBO): the KL-divergence is *non-negative*, hence the ELBO represents the lower bound to the evidence $P(D)$. Being $\log(P(D))$ only dependent on the prior and constant with respect to the parameters θ, maximizing the ELBO is equivalent to minimizing the KL-divergence.

Optimization of the ELBO can be conducted with various methods, the most popular being the stochastic adaptation of the gradient descent to variational inference known as *Stochastic Variational Inference* (SVI) (Jospin et al. 2022, 40); introduced by Graves (2011), it can be applied to most ANNs functional models, as it integrates with their standard architectures and training methods. However, further adaptations are required for functioning in deep-layered neural networks, as stochasticity stops backpropagation from functioning at the internal nodes of a network (Jospin et al. 2022, 41).

The solution lies in a *probabilistic backpropagation* algorithm, also known as *Bayes-by-backprop* (Blundell et al. 2015), which can be applied to differentiable functions such as neural networks. The basic premise for this algorithm is a "*reparametrization trick*" that ensures that backpropagation (Equation 2.2) works as usual, even though *uncertainty* is introduced in the weights of the network, allowing to optimize a pre-defined objective function to learn a distribution on the weights of a neural network.

The main idea is to use a random variable $\epsilon \sim q(\epsilon)$ as a source of *noise* not caused by variational inference. As detailed in Jospin et al. (2022, 41), the parameter space Θ is not sampled directly, but obtained through a deterministic transformation $t(\epsilon, \phi)$ such that $\theta = t(\epsilon, \phi)$ follows $q_\phi(\theta)$. ϵ is sampled, changing at each iteration, but can be considered a constant with regard to other variables: because all other transformations are non-stochastic, backpropagation works as usual for the variational parameters ϕ, which means that it uses gradient updates as in Equation 2.2 and it can readily be scaled using optimization schemes and readily implemented with GPU-based processing (Blundell et al. 2015), as it is common practice in the latest DL software libraries to greatly speed up the learning phase.

The general formula for the ELBO (Equation 3.5), becomes:

$$\int_\epsilon q_\phi(t(\epsilon, \phi)) \log \left(\frac{P(t(\epsilon, \phi), D)}{q_\phi(t(\epsilon, \phi))} \right) |\det(\nabla_\epsilon t(\epsilon, \phi))| d\epsilon \tag{3.6}$$

Blundell et al. (2015) use the fact that, if $q_\phi(\theta)d\theta = q(\epsilon)d\epsilon$, $t(\epsilon, \phi)$ is invertible with respect to ϵ, the distributions q_ϕ and $q(\epsilon)$ are not degenerated; being $f(\theta, \phi)$ a differentiable function, then the ELBO gradient can be rewritten as:

$$\frac{\partial}{\partial \phi} \int_{\phi} q_{\phi}(\theta') f(\theta', \phi) d\theta' = \int_{\epsilon} q(\epsilon) \left(\frac{\partial f(\theta, \phi)}{\partial \theta} \frac{\partial \theta}{\partial \phi} + \frac{\partial(\theta, \phi)}{\partial \phi} \right) d\epsilon \quad (3.7)$$

The Bayes-by-backprop algorithm can be summarized by the following pseudocode:

Algorithm 3 Bayes-by-backprop

```
$\phi = \phi_0$
**for** $i = 0 \rightarrow N$ **do**
    Draw $\epsilon \sim q(\epsilon)$;
    $\theta = t(\epsilon, \phi)$
    $f(\theta, \phi) = \log(q_{\phi}(\theta)) - \log(p(D_y|D_x, \theta)p(\theta))$;
    $\Delta_{\phi} f =$ **backprop**$_{\phi} f$;
    $\phi = \phi - \alpha \Delta_{\phi} f$;
**end for**
```

3.4 Review of recent developments for Bayesian Neural Networks

To introduce this brief examination of more recent probabilistic neural networks applications to time series forecasting, the following quote taken from H. Wang and Yeung (2020) survey's abstract highlights again how they play a fundamental role in the more general push to integrate human-like intelligence in automatic systems:

"A comprehensive artificial intelligence system needs to not only perceive the environment with different "senses" [...] but also infer the world's conditional (or even causal) relations and corresponding uncertainty".

As addressed in the conclusion of Section 2.3 it is indeed fundamental to integrate stochasticity into the neural network functional model if these powerful tools are to be integrated into critical "intelligent" systems.

Bayesian Deep Learning models have flourished trying to bridge the advances of deep learning and probabilistic inference on random variables and, focusing on the main theme of this thesis, Table 3.1 presents a taxonomy of the leading more recent models in the field. It can be noted that some of the most important Big Tech giants' research centers have participated and are active in this research[1]: this is an important indicator that this combination is ripe with practical business-related applications, along with the theoretical insights.

Table 3.1: List of the more recent Bayesian Neural Networks specialized in time series forecasting.

Model	Reference
DeepAR	Salinas et al. (2020)
DeepState	Rangapuram et al. (2018)
Spline Quantile Function RNN	Gasthaus et al. (2019)
DeepFactor	Y. Wang et al. (2019)

All these models follow the same approach explored in this thesis, merging an RNN-based perceptron component and a stochastic model able to handle the conditional relationships among variables: *"an ideal forecasting model requires both efficient processing of high-dimensional data and sophisticated modeling of different random variables, either observed or latent."* (H. Wang and Yeung 2020, 32).

The Deep AutoRegressive (DeepAR) from Salinas et al. (2020) is an LSTM-based recurrent forecasting algorithm that belongs to the family of probabilistic deep learning models, specifically designed for time series and tailored to forecast inventory demand, even with just a few data points. It works specifically to learn patterns and dependencies between multiple series, such as many similar ones across a set of cross-sectional units, by leveraging not only their historical values but also any relevant contextual information, such as categorical features or time-related covariates; moreover, it generates a full probability distribution over future time steps in the form of Monte Carlo samples (Salinas et al. 2020, 3) that can be used to compute consistent quantile estimates for all sub-ranges in the prediction horizon allowing users to quantify uncertainty and make informed decisions.

Deep State Space Models (Rangapuram et al. 2018) combine the flexibility and expressiveness of deep learning architectures with the structured and interpretable representations of classical state space models (SSM) by parametrizing a per-time-series linear SSM with a jointly-learned recurrent architecture. The most interesting aspect of this model is its combination of the ability to process complex and highly dimensional datasets, typical of deep learning, with the interpretability and

[1] In particular, Google (Fortunato, Blundell, and Vinyals 2017; Vaswani et al. 2017) and Amazon (Salinas et al. 2020; Rangapuram et al. 2018; Gasthaus et al. 2019; Y. Wang et al. 2019).

the capability of explicitly incorporating structural assumptions, such as seasonality, typical of SSMs.

The probabilistic forecasting method developed by Gasthaus et al. (2019) combines the modeling capacity of RNNs with the flexibility of a spline-based representation of the output distribution. The main advantage of this approach is that learning a non-parametric quantile function avoids having to specify a parametric form of the observed data distribution beforehand (Gasthaus et al. 2019, 10); especially because choosing the wrong parametrization can have a profound effect on the model performance, the quality of its forecasts, and its ability to handle randomness.

The Deep Factor Model (Y. Wang et al. 2019), similarly to the DeepAR model, is tailored to producing probabilistic forecasts for large collections of similar and/or dependent time series, trying to marry the ability to handle complex patterns and scalability to large datasets; to do so, it represents each time series, or its latent function, as a combination of a *global time series*, which is handled by the deep learning part of the architecture, and a corresponding *local model*, which is stochastic, tasked with capturing the individual random effects associated with each series (Y. Wang et al. 2019, 2).

To conclude, the variety of structures and combinations that these four models show the *flexibility* of Bayesian Neural Networks; as examined in Section 3.1, Bayesian modeling can be used to adapt different functional perceptron-based architectures to perform inference.

The following section will test the performance of one of the combinations of deep learning and probabilistic modeling with simulated data from an AR(1) process and a latent factor model.

3.5 Simulated examples

This section is dedicated to putting the proposed model to the test: *is the Bayesian Recurrent Neural Network able to produce accurate and sensible probabilistic forecasts*?

Also, different functional variations within the same stochastic models and posterior approximations will be tested, along with the two different approximation frameworks detailed in Section 3.3 and Section 3.2, to understand their respective influences on the models' performances, therefore improving the understanding of the model selection process and its hyperparameter tuning in preparation for the following chapter, in which real data will be used.

This performance for point forecasts will be evaluated on both datasets using the Root Mean Square Error (RMSE) and Mean Average Error (MAE); these indicators will also be computed for two classical benchmark methods, *mean* and *naïve*, to provide grounding to the BRNN performance, along with the Winkler score (Winkler 1972) aimed at evaluating the computed intervals[2]. Moreover, *coverage* statistics will be included to test not only the performance of point estimates, as is the usual case with Neural Networks, but also to conduct a *posterior predictive check*, as the Bayesian framework (Section 3.1) allows to generate a full distribution of estimated values that can then summarized through scale or shape indexes, such as the mean or a set of quantiles. We are interested in comparing the credible intervals associated with every point estimate, computed using the posterior with $\alpha = 5\%$ as the significance level, and the out-of-sample data points. In particular, we would expect $\approx 95\%$ of these test observations to fall within the computed intervals if the model is appropriate.

The simulations will be based on stochastic processes, which will be processed beforehand: after generating the synthetic data, they have been *scaled* and *centered*, a standard practice in deep learning that ensures a speed-up of the learning process. Both data-generating processes underpinning the simulated datasets have been chosen to test whether the chosen BNN can capture complex signals while excluding the noise components that can skew its prediction in different scenarios that are common in time series forecasting; as anticipated, the insights derived from the cross-validation and forecasting in this simulated setting will be used in section Section 4 to choose the best structural and probabilistic form of the model that will forecast with real data.

The first simulation involves an $AR(1)$ process defined as such:

$$y_t = 0.8y_{t-1} + \epsilon_t, \qquad \epsilon \sim WNN(0,1) \tag{3.8}$$

with $t = 1, \ldots, 1000$. This process generates a *stationary* time series, as can be seen in Figure 3.2; the autocorrelation (ACF) and partial autocorrelation (PACF)

[2]When observations are within the specified interval, the Winkler score equals the length of that interval, implying narrower intervals correspond to lower scores. However, if an observation lies outside the interval, a penalty is incurred proportional to the deviation of the observation from the interval.

Figure 3.2: *AR(1) simulation plots*: on the top, a simple line chart representing the resulting time series, with different colors to represent the train-test split of the data points; on the bottom, the AutoCorrelation Function (ACF) plot on the left-hand side and the Partial AutoCorrelation Function (PACF) plot on the right-hand side, with blue dashed lines corresponding to confidence interval for the white noise hypothesis.

functions are also included.

Figure 3.3: *Latent factor simulation*, as described by Equation 3.9: on the top, a simple line chart representing the resulting time series, with different colors to represent the train-test split of the data points; on the bottom, the AutoCorrelation Function (ACF) plot on the left-hand side and the Partial AutoCorrelation Function (PACF) plot on the right-hand side, with blue dashed lines corresponding to confidence interval for the white noise hypothesis.

The second simulation, instead, uses a latent factor model to generate a sequential set of observations, as described by the following equation:

$$y_t = \underbrace{0.1 \times Z_t}_{\text{Random Walk}} + \underbrace{0.3 \times \sin(1.5t + U)}_{\text{Sinusoidal process}} + \underbrace{\epsilon_t}_{\text{White Noise}} \tag{3.9}$$

where $Z_t \overset{iid}{\sim} N(0,1), U \sim \mathcal{U}(0,\pi)$ and $\epsilon_t \overset{iid}{\sim} N(0,2)$. The resulting series is plotted in Figure 3.3, along with the relative ACF and PACF.

The stochastic model underpinning the BNNs will be the same in every learning process: first, the prior distribution has to be specified; for every following model, a scale mixture of Gaussians for each network parameter has been chosen:

$$\xi\mathcal{N}(0,\sigma_1) + (1-\xi)\mathcal{N}(0,\sigma_2) \tag{3.10}$$

The models have been trained with $\sigma_1 = 1, \sigma_2 = 2$, and $\xi = 0.5$. For the priors of the parameters introduced by the likelihood, a Gamma distribution with $\Gamma \sim (2, 0.5)$ has been chosen to sample from, and the values for all parameters θ have been initialized with a standard Normal distribution $\mathcal{N}(0, 1)$.

Both Bayes-by-backprop (Section 3.3) and the Metropolis-Hasting (Section 3.2) algorithms have been employed to obtain several samples from the predicted posterior distribution; the first ran for either 10^4 (for multi-layered models) or 10^5 (for single-layered models) epochs with the ADAM algorithm (Kingma and Ba 2014), which optimizes the stochastic gradient descent (Equation 2.2) learning required to minimize the ELBO (Equation 3.5) function and is commonly used in deep learning for large datasets and very noisy and sparse gradients.

The number of samples to obtain has been determined through cross-validation: as foretold in Section 3.2 and Section 3.3, this sampling process is of foremost importance in determining how well the exact posterior is approximated by the different approximation techniques. Hence, keeping the same basic functional form constant[3], these effects have been studied for both sampling processes.

Figure 3.4 shows the evolution of a Variational Inference-approximated predictive posterior and Figure 3.5 for a Markov Chain Monte Carlo approximation for an increasing number of samples. The difference inherent within these approaches is made evident: while the VI-based approach reaches a good approximation in fewer steps, the MCMC takes 10^5 samples to converge to a proposed distribution. While in the former case, the overall computational time is impacted by the fact that the underlying NN has to be trained, the latter quickly consumes higher resources, considering also that the posterior has to be analyzed to obtain quantiles, interval bounds, indexes and, if optimizations are not implemented while performing the calculations on matrices that can quickly grow to contain millions of records this phase could weigh heavily in the overall process.

Choosing a Recurrent Neural Network with layers composed of Long-Short Memory Term units (Section 2.6) implies also another fundamental hyperparameter: the overall structure of the model. While a simple single-layered model will always be trained in the following analysis, serving as a point-of-reference for the trade-off between performance and computational resources, another cross-validation is

[3]A single-layered Bayesian Recurrent Neural Network.

Figure 3.4: *Evolution of the approximated posterior with an increasing number of samples* with a Variational Inference-based model. In each chart, grey lines represent all forecasts, while the red line is the actual objective distribution that we wish to forecast. Dashed lines, color-coded in the same way, represent the *median* point of both distributions. The same BRNN has been trained and the resulting predictive posterior distribution has been sampled to obtain a snapshot of the effect of increasing the number of samples. It can be seen that a quick convergence to a good approximation is reached within a range of 10^3 to 2.5×10^3 samples. Increasing the number of samples is not improving dramatically the overall shape of the distribution; however, the forecasted median converges almost exactly at 10^4 samples.

Figure 3.5: *Evolution of the approximated posterior with an increasing number of samples* with a Markov Chain Monte Carlo-based model: as expected from a theoretical point of view (Section 3.2), increasing the number of samples dramatically increases the accuracy of the prediction. In each chart, grey lines represent all samples, while the red line is the actual objective distribution that we wish to forecast. Dashed lines, color-coded in the same way, represent the *median* point of both distributions. A good approximation is reached only for 10^5 samples, which increases significantly the overall computation time, and the approximated distributions and their quantiles demonstrate a wide range of shapes and scales.

needed to examine whether deeper functional models can outperform simpler architectures, to understand whether adding complexity will lead to increased performance or might overfit the training data, weighting eventual benefits against the increased costs of training such a model.

Table 3.2: *Performance indicators for different deep learning architectures of a Bayesian Recurrent Neural Network*. The number of *activations* indicates the number of LSTM (Section 2.6) units that compose the network's topology. The increasing number of *parameters* has no significant effect on the network performance on the cross-validation set.

Activations	Parameters	RMSE	MAE	Winkler Score	Coverage
1	30	0.5759098	0.4711644	2.723399	0.970
2	83	0.5779981	0.4733379	2.749886	0.976
3	160	0.5698022	0.4662353	2.728590	0.976
4	261	0.5803210	0.4741019	2.767175	0.970
5	386	0.5764124	0.4692083	2.787441	0.970
6	535	0.5845347	0.4760758	2.738011	0.964
7	708	0.5761679	0.4711341	2.763580	0.970
8	905	0.5769288	0.4755576	2.729739	0.970
9	1126	0.5763785	0.4714887	2.744778	0.970
10	1371	0.5782066	0.4754351	2.752156	0.976

Table 3.2 shows how increasing the number of activations, and consequently, the number of parameters, has little effect on the network performance: this has a twofold implication. On the one hand, *complicating the structure of the model is not worth the extra computational effort required*, as it will lengthen the fitting process without bringing a noticeable result. On the other hand, this confirms (Section 3) that complex Bayesian Neural Networks *do not show the tendency to overfit the training data*: their non-stochastic counterparts, instead, can dramatically overfit, bringing overconfidence and variance, which leads to a failure in generalization, especially with layered architectures.

A deep learning model will nevertheless be trained to test this hypothesis: *Are the complications and extra effort that a more articulated network topology brings to the learning process worth it, even when confronted with static and dynamic forecasts*? It is important to compare if and how including hidden layers and augmenting the dimension of the hidden states makes the network able to capture more information; if this is not the case, the cost in computational resources required to learn the higher number of parameters and the accuracy improvement might lead to prefer simpler models. The variations on the recurrent LSTM structure that will be trained and their characteristics are summarised in Table 3.3, along with an identifier that will allow us to point out their performance. Comparing the results of Table 3.2, a number of 3 activations seems the best cut-off point, as it performs slightly better than its peers, and will be used to forecast.

Table 3.3: Structure of the two models used to forecast the simulated datasets of this section. The notation of the structure represents the type of unit used in the layer, such as `LSTM`, along with the count of inputs and outputs as `(number of inputs, number of outputs)`.

Identifier	Structure	Number of parameters
Simple BRNN	`LSTM(1, 1)`	14
	$\rightarrow$ `Dense(1, 1)`	
Deep BRNN	`LSTM(1, 3)`	232
	$\rightarrow$ `LSTM(3, 3)`	
	$\rightarrow$ `LSTM(3, 3)`	
	$\rightarrow$ `Dense(3, 1)`	

To summarise, a basic model with a single LSTM input cell will be pitted against a deep learning model with two additional hidden layers, the best performer of Table 3.2: the input layer will be composed of a single input unit and either one or three output units, while the two intermediate hidden layers for the DL-based network are composed of either one or three LSTM units for both inputs and output connections. The final output layer will be a standard *dense* layer (Figure 2.2) that summarises its input linearly through the *identity* function as the final, single activation; since the LSTM unit already performs a set of non-linear transformations and we need to use the model in a regression setting, it is common practice to simply combine and scale the result of the second-to-last layer to obtain the required prediction.

It is now time to test the two models and the different approximation methods. Both *dynamic* and *static* forecasts on a test set have been performed, with a train-test split of $N_{\text{train}} = 825$ and, respectively, $N_{\text{test}} = 75$ for the out-of-sample forecasts and $N_{\text{test}} = 175$ for the *dynamic* scenario, in which the actual value of the lagged independent variable is used for each subsequent forecast. The difference in the dimension of the test sets underlines the more challenging environment presented by the out-of-sample scenario, in which the ability of the network to generalize is truly challenged, as none of the test sets is used until the final assessment.

In the *dynamic* setting, both models outperform their benchmark, regardless of the approximation used, showing exceptional capabilities to react to sudden shifts and jumps of the simulated data.

Starting from the $AR(1)$ process simulation (Equation 3.8), Figure 3.6 charts their behavior, along with 95% credible intervals: of great interest is the fact that these intervals range *shifts* depending on both lagged and actual values at time t and the different models and approximations used. This is reflected by both the Winkler scores and coverage; also, the visual analysis shows how almost all of the target values fall within these intervals.

Table 3.4: *Performance indicators for the AR(1) simulation.* In particular, both simpler Bayesian networks using either Bayes-by-backprop (Section 3.3) or MCMC (Section 3.2) methods performance was on par with their more complex counterparts, while having many fewer parameters and therefore requiring much less fitting time (see Table 3.3).

Model	Set	RMSE	MAE	Winkler Score	Coverage
Naïve	test	1.07361	0.84565	5.51448	1.00000
Mean	test	1.17975	0.95516	20.40839	0.92000
Simple, VI	train	0.55567	0.43581	2.75992	0.96000
Simple, VI	test	0.59302	0.46427	2.91426	0.93939
Simple, MCMC	train	0.56462	0.44287	2.81924	0.96000
Simple, MCMC	test	0.61801	0.48028	3.03202	0.92727
DL, VI	train	0.55659	0.43544	2.86081	0.96970
DL, VI	test	0.60652	0.47086	2.91418	0.96364
DL, MCMC	train	0.56257	0.44110	2.80094	0.95758
DL, MCMC	test	0.62241	0.48300	3.11164	0.91515

The resulting forecast performance is summarised in Table 3.4. All performance indicators on both training and test sets highlight how all versions of the Bayesian Recurrent Neural Network performed with good accuracy, generalizing very well to capture the series dynamics and showing consistency in their credible intervals.

In addition, the deep learning Variational Inference-based network required more fine-tuning and hence failed the first tests: the resulting output is shown in Figure 3.7.

Figure 3.6: *AR(1) mean forecasts with 95% credible intervals*, with the simple model output on the first row (subplots 1 and 2) and the deep learning model output on the second row (subplots 3 and 4). The output on the first column (subplots 1, 3) derives from a variational approach that involved training the models with Bayes-by-backprop (Section 3.3) for 10^5 epochs before sampling their variational posteriors. The output on the second column (subplots 2, 4) has been obtained through the Metropolis-Hasting algorithm, hence it did not require training the model (Section 3.2). The functional architecture details are shown in Table 3.3, but the only difference column-wise between the models is the approximation technique. All models show remarkable performance on this simple simulated dataset.

Figure 3.7: *Comparison between models, one that has been properly trained, and another in which the training process failed*, for a simulated $AR(1)$ process. The Variational Inference framework (Section 3.3) requires a standard learning phase, in which batches of training data are fed to the network, performing backpropagation in its Bayesian version. Even when this process fails, the resulting forecast is comparable to the benchmarks; moreover, the overall uncertainty captured by the intervals longer width shows that the model signals its lack of generalization through the distribution of its output.

Table 3.5: *Comparison between learned and unlearned models,* $AR(1)$ simulated dataset. The VI-based deep learning architecture requires more fine-tuning, hence the first standard test runs failed to produce performant dynamic forecasts; however, the results are still comparable with the benchmark forecasts displayed in Table 3.4.

Model	Set	RMSE	MAE	Winkler Score	Coverage
Learned	train	0.55659	0.43544	2.86081	0.96970
Learned	test	0.60652	0.47086	2.91418	0.96364
Not learned	train	0.97509	0.78372	4.47801	0.96000
Not learned	test	1.10122	0.87663	5.62730	0.91515

Even when the learning process fails, the resulting forecasts are still comparable with the benchmarks (Table 3.5), showing that even in such a worst-case-scenario the model still performs acceptably, without critical failures or returning absurd or over-confident predictions; nevertheless, it can be noted that this failure has an impact on the posterior ability to correctly estimate the associated uncertainty, as only 91.5% of observations fall within the computed intervals. Coverage notwithstanding, this is also the case in which the prediction intervals are wider, ≈ 3.87.

It can also be noted by comparing all the other panels that dilation of the intervals corresponds to the points in the time series in which its dynamics reach wider deviations from the mean: in particular when a downward spike that almost reaches -3σ before moving steeply upward in just a few time steps occurs in the simulations.

In this second setting, testing the BRNN against a latent factor model as detailed in Equation 3.9, all models show outstanding performances, regardless of the architecture or approximation technique, even if the time series shows a more pronounced cyclical dynamic mixed with a random walk.

The simple, single-layered recurrent MCMC-based structure, in particular, while not being the best performer, shows that a BRNN *mean* forecast can follow the test data point quite closely, mimicking spikes, trends, and cycles consistently, even if they are accompanied by an increase in the overall uncertainty, as shown by the wider shape of the resulting predictive posterior; this combination suggests that the number of samples from the posterior is not enough to correctly approximate the test set distribution.

Figure 3.8: *Test set mean forecasts with 95% credible intervals for the latent factor model simulated data*, with the simple model output on the first row (subplots 1 and 2) and the deep learning model output on the second row (subplots 3 and 4). In this case, both the simple BRNN with VI and the Deep Learning BRNN with MCMC forecast performance was approximately similar, following the test series dynamics more closely and having narrower intervals.

Table 3.6: *Performance indicators for the latent factors model simulation* described by Equation 3.9. All other models performed consistently well, outclassing the simpler forecasting approaches. In this second setting, choosing the best performing model is not clear cut.

Model	Set	RMSE	MAE	Winkler Score	Coverage
Naïve	test	0.85925	0.73131	4.07134	1.00000
Mean	test	0.91664	0.78840	16.02283	0.99000
Simple, VI	train	0.36244	0.29101	1.68654	0.96121
Simple, VI	test	0.37517	0.29545	1.87148	0.94545
Simple, MCMC	train	0.37598	0.30230	1.85831	0.96970
Simple, MCMC	test	0.37570	0.29052	1.89329	0.96364
DL, VI	train	0.35715	0.28654	1.69972	0.97697
DL, VI	test	0.38145	0.29978	1.86120	0.95758
DL, MCMC	train	0.36075	0.28863	1.65353	0.96000
DL, MCMC	test	0.37091	0.28858	1.89660	0.93333

The benchmarks were outperformed by all the trained models and, while the learning process still failed for the VI-based deep learning model (Figure 3.9 and Table 3.7), it is also the simulation test result that has the widest prediction intervals, which this time include 99% of the out-of-sample data. This reactive dilation of the intervals to the differences in the inherent uncertainty is more evident in this simulation than in Figure 3.6.

Table 3.7: *Comparison between learned and unlearned models*, latent factor simulated dataset. The VI-based deep learning architecture requires more fine-tuning, hence the first standard test runs failed to produce performant dynamic forecasts; however, the results are still comparable with the benchmark forecasts displayed in Table 3.6.

Model	Set	RMSE	MAE	Winkler Score	Coverage
Learned	train	0.36059	0.29018	1.72909	0.97818
Learned	test	0.38448	0.30366	1.93666	0.95152
Not learned	train	0.98091	0.76940	4.93985	0.94182
Not learned	test	0.84926	0.72472	3.89396	0.99394

Another fundamental aspect of the capabilities of a Bayesian Neural Network, as detailed in Section 3.1, is its ability to output a *full predictive distribution*, from which

Figure 3.9: *Comparison between models, one that has been properly trained, and another in which the training process failed*, for the simulated latent factor dataset. The Variational Inference framework (Section 3.3) requires a standard learning phase, in which batches of training data are fed to the network, performing backpropagation in its Bayesian version. Even when this process fails, the resulting forecast is comparable to the benchmarks; moreover, the overall uncertainty captured by the intervals longer width shows that the model signals its lack of generalization through the distribution of its output.

a mean, median, or whatever quantile forecast desired can be computed through simple formulas. Hence, we can compare this output with the *actual* test data by using descriptive statistics such as the *mean*, *median*, *skewness*, and *kurtosis*.

Table 3.8: *Descriptive statistics for the predictive posterior and the actual test distribution*, simulated $AR(1)$ dataset.

set	Mean	Median	Skewness	Kurtosis
Test	0.0339625	-0.0383083	-0.1214046	3.130651
VI, simple	0.0281679	0.0615775	-0.2064915	2.911297
VI, DL (learned)	0.0018364	0.0359133	-0.2077651	2.885629
VI, DL (unlearned)	-0.0035615	-0.0035613	-0.0003094	3.007421
MCMC, simple	0.0377277	0.0731352	-0.1987595	2.803121
MCMC, DL	0.0176742	0.0593319	-0.2105090	2.747941

Table 3.9: *Descriptive statistics for the predictive posterior and the actual test distribution*, simulated latent factor model dataset.

set	Mean	Median	Skewness	Kurtosis
Test	-0.5674783	-0.5920102	-0.0035522	2.789960
VI, simple	-0.5893648	-0.6074777	0.1145625	2.760384
VI, DL (learned)	-0.5857342	-0.5942244	0.0612518	2.790477
VI, DL (unlearned)	-0.5975992	-0.6049365	0.0436146	2.755918
MCMC, simple	-0.5691130	-0.5872220	0.3268309	5.264646
MCMC, DL	-0.5830328	-0.5995767	0.1462454	2.920661

Table 3.8 and Table 3.9 show this comparison. While some features of the target distribution are not being captured exactly, they are still a good approximation, not far off from the actual values. In particular, for both simulations, the autoregressive process simulation characteristics (left-skewed and leptokurtic) are found in the approximation of the predictive posterior. The same is not happening for the simulated latent factor model, in which, while the mean and median forecasts are more precise and the approximated posterior has a kurtosis in almost all iterations that differs by the 10^{-2} order of magnitude, the skewness is not captured properly.

To conclude this section, the following paragraphs put the two structural variations of the BRNN and the two different posterior predictive approximations to the hardest test, which is forecasting out-of-sample.

Figure 3.10: *Actual and predicted densities comparison*, $AR(1)$ dataset.

Figure 3.11: *Actual and predicted densities comparison*, simulated latent factor model dataset.

For these last experiments on simulated data, the output of the trained BRNNs has been fed back, retraining the network iteratively for several steps; afterward, the usual performance indicators have been computed using the test set. As expected, the performance significantly worsens, but *not all models perform worse than the benchmark*.

Figure 3.12: *Static forecast*, $AR(1)$ simulation. The quality of the forecast is not good, as can be foretold by observing the chart, in which all models fail to pick up relevant patterns, reverting to what is an approximated *mean* forecast. Moreover, the credible intervals are not reactive, hence a relevant part of the target data points fall outside them.

Figure 3.12 charts the static forecasts, while Table 3.10 shows the relative performance indicators, for the dataset corresponding to the simulated $AR(1)$ process. In this case, all models perform *on par*: there is no clear-cut winner, but the extra effort and sophistication of the BRNN have to be taken into account; hence, it seems that the simplest *mean* forecast would be a better choice than the complex network. In particular, the probabilistic aspect of the forecast, summarized by the Winkler Score and the coverage statistics, reveals that the network fails to correctly capture the overall variation and shape of the target distribution, as too many of the test data points fall outside the 95% credible intervals.

Table 3.10: *Performance indicators* for $AR(1)$ simulation. While no clear winner is looking at these statistics, given the extra computational effort involved, the mean method seems to be outperforming the more sophisticated BRNN.

Model	RMSE	MAE	Winkler Score	Coverage
Mean	1.285586	1.0260118	7.106446	0.850
Naïve	1.488923	1.2419381	13.431981	1.000
VI, simple	1.389106	1.1484818	14.907949	0.573
VI, DL	1.295991	1.0411589	8.535286	0.760
MCMC, simple	1.297111	0.9911647	13.506259	0.640

This changes with the simulated latent factor model: the BRNNs, while not outclassing completely the simpler method as they did in the *dynamic* forecast setting, are still outperforming the benchmarks, with the MCMC-approximated single-layered recurring network as the winner. Even if the coverage is still not able to match the nominal value associated with the credible intervals, a 3% difference is not as problematic as what happened with the AR simulated process.

Table 3.11: *Performance indicators* for latent factor model simulation. In this case, all networks outperform the benchmarks, with the MCMC being the best performance in each indicator for both point and probabilistic forecast quality.

Model	RMSE	MAE	Winkler Score	Coverage
Mean	0.6009647	0.4636015	4.082119	0.920
Naïve	0.6461243	0.5047261	10.545585	1.000
VI, simple	0.5713207	0.4375569	3.716109	0.853
VI, DL	0.5849795	0.4494713	4.135341	0.907
MCMC, simple	0.4909852	0.3740334	2.804273	0.920

To conclude, it seems that better tools exist to deal with time series that can be modeled as autoregressive processes: in this case, the BRNN is not able to outperform the simplest methods, hence they are not a viable choice in a forecasting scenario.

However, with more complex time series that can be modeled by latent factors models, even a simple, single-layered LSTM recurrent network that samples from the predictive posterior using the MCMC technique for approximation can withstand

Figure 3.13: *Static forecast*, latent factor model simulation. In this case, the BRNNs outperform the benchmarks, with the simple MCMC-based network taking the crown, having lower errors, Winkler Score, and higher coverage, without it being too computationally expensive to test with a simple dataset of size $T = 900$. While neither of the other models captures the kind of patterns that the target series is showing, it exhibits slight variations that approximate the original dynamics, further reducing the errors of its point forecasts. Moreover, wider credible intervals allow it to cover most of the original data points.

and outperform the benchmarks on point-forecasts out-of-sample, capturing the uncertainty associated with its predictions.

4 Real data applications

This section explores the application of the powerful framework unveiled in the previous chapter, testing the Bayesian Recurrent Neural Network in an applied scenario, forecasting dynamically on real-world data.

Section 4.1 will present the dataset and the volatility estimator that will be targeted and forecasted, the *realized range* (Martens and Van Dijk 2007), while Section 4.2 and Section 4.3 the Bayesian Recurrent Neural Network resulting performances.

4.1 Presenting the datasets and the realized range

The two experiments reported in the following sections will perform a *dynamic* forecast of *volatility* of *high-frequency trading data* in challenging scenarios, sourced from Bloomberg. Both datasets contain the *closing price*, *high*, *low*, and *net change*, each recorded at intervals of *one minute*. First, Section 4.2 targets the realized range of the €/$ exchange rate, starting from 01/12/2023 and ending on 03/02/2024, while Section 4.3 will be about forecasting the volatility of the S&P500 index using data from 02/02/2020 to 26/03/2020, a time frame in which COVID-19 brought havoc on the financial markets after a relatively stable run, causing sudden spikes and falls in the overall volatility compared to the previous periods, as manifested by a steep descent of the index value followed by a timid bounce back around March, a month in which the maximum volatility value is observed.

In both cases, this thesis will build on Martens and Van Dijk (2007) and use the *scaled realized range* to estimate volatility. Volatility is both time-varying and, to a certain extent, predictable (2007, 181), and the realized range works as an efficient estimator for this feature of financial time series. Mathematically, it is defined as:

$$RR_t^{\Delta} = \frac{1}{4\log 2}\sum_{i=1}^{I}(\log H_{t,i} - \log L_{t,i})^2 \tag{4.1}$$

where $H_{t,i}$ and $L_{t,i}$ are the high and low values of the price at time t. To make it more robust to *market microstructure frictions*, a typical characteristic of intraday data caused by the high-frequency sampling and the noise it casts, a bias-correction procedure develops Equation 4.1 further, consisting of scaling the result by the ratio or the average level of the daily range and the average level of the realized range of the previous trading days. The scaled realized range $RR^{\Delta}_{S,t}$ can be written as:

$$RR^{\Delta}_{S,t} = \left(\frac{\sum_{l=1}^{q} RR_{t-l}}{\sum_{l=1}^{q} RR^{\Delta}_{t-l}} \right) RR^{\Delta}_{t} \tag{4.2}$$

where q is the number of trading days used to compute the scaling factor, which has been set to $q = 5$ for all simulations.

Simulation experiments and empirical analysis on S&P500 index futures and S&P100 constituents reveal the realized range's superior performance over another common estimator used as a proxy for volatility, *realized variance* (Martens and Van Dijk 2007, 188), even with microstructure noise: while the dataset used does not contain *tick-by-tick* data and the aggregating at the frequency on one minute helps avoiding dirty signals and errors, proper data cleaning and preparation is still needed to properly ensure that the data fed to the BRNN are not contaminated by wrongly reported observations.

Following Corsi, Peluso, and Audrino (2015, 389), a *filter* is applied to remove wrongly reported observations that may distort the analysis: the filtering process starts with calculating a robust indicator of daily price variability, utilizing the sample standard deviation of observed prices within the 25th to 75th percentile range of the empirical distribution. Following this, all prices falling outside an interval, defined as twice this robust variability measure and centered around both the previous and subsequent price observations are eliminated. Formally, being $(Y_{i,1}, \dots, Y_{i,T})$ the T sorted observed prices, the sample standard deviation is computed on the subsample interquartile range $(Y_{i,\lfloor 0.25T \rfloor}, \dots, Y_{i,\lfloor 0.75T \rfloor})$, where $\lfloor \cdot \rfloor$ is the_floor_ function. This standard deviation d is the filter's *threshold*: if $|Y_{i,t} - Y_{i,t-1}| > 2d$ and $|Y_{i,t} - Y_{i,t+1}| > 2d$, then the sample is removed from the series and substituted with the boundary value.

The training set will consist in both experiments of most of the observations available, which will forecast a specified number of days in advance: given h the number of trading days for which the minute-to-minute forecast will be provided, $h = 3$ for the S&P500, while $h = 5$ for the FOREX trading data. For these tasks, all the insights from Section 3.5 suggest that the *simplest* architecture will offer a desirable performance, considering the trade-off with computational resources: all things considered, the simple, single-layered LSMT-based recurrent architecture is the best overall choice, taking into account that adding layers and thus complicating the network only slightly increases the performance for both probabilistic and point forecasts.

The sampling techniques will also have a huge impact on the training and testing process, adding to the computational cost of training in this intensive setting: each day consists of either 390 samples for the S&P500, corresponding to 6 and a half hours of trading, or 1440 observations for the currency rate; hence, a 5 days forecast in this second setting consists of 7200 minute-by-minute predictions. Another consideration must involve the number of posterior samples required: the low number of samples needed to obtain a stable distribution will also speed up the calculations of quantiles and other indicators. Variational inference seems to be the best approximation technique for this scenario since 10^4 samples are more than enough to obtain sufficient precision. Markov Chain Monte Carlo is not viable because, as foretold in Section 3.2, ensuring convergence is computationally intensive and expensive, without taking into account the time spent in evaluating the required statistics from the sampled posterior. Nevertheless, Section 4.3 will present an MCMC-based forecast: being opened for a limited number of days for a total of 390 minutes each day, forecasting dynamically 3 days in advance is feasible within reasonable computational costs.

To conclude this introduction to the techniques and approaches used and before presenting the datasets and the experiment results, one last issue needs to be addressed. Whenever missing observations present themselves, a solution is required, since ANNs are not capable of handling them: if placeholders such as `NA` are fed through the loss function, they will hinder the backpropagation and learning process, resulting in a stream of `NA` forecasts. *Imputation* has been chosen as the strategy to fill in the gaps caused by market closure, holidays, and other events that disrupt the continuity of data by repeating the last available observation. In the case of the S&P500 dataset, a dummy variable representing days in which the market was closed has been engineered to filter out the missing observations and obtain a single stream of values.

Figure 4.1: *Scaled realized range, closing price, and net change* for €/$ exchange rate.

4.2 €/$ exchange rate

This time series presents challenging dynamics, as represented in Figure 4.1, exhibiting a random walk of closing prices that pairs with a lack of patterns in both the scaled realized range and net change, suggesting an almost white-noise-like sequence of observations. Table 4.1 contains the relative summary statistics for the whole dataset. If we focus on the test set and isolate them in a single facet of charts representing these 5 trading days (Figure 4.2), some slight patterns are nevertheless discernible, such as those linked to the opening-closing hours of the market at the different time zones or the different days of the week.

Table 4.1: *Descriptive statistics* for €/$ exchange rate. The observed distribution is far from normal, with high asymmetry and kurtosis.

Mean	Median	Sd	Skewness	Kurtosis	Min	Max
0.0001251	9.2e-05	0.0001706	8.647287	182.1783	0	0.0054104

If we compare *point-forecast* accuracy, the model is not beating one of the two benchmarks, a simple *naïve* forecast, which is usually the case in this kind of

Figure 4.2: *5 days sample scaled realized range* for €/$ exchange rate.

scenario; there are however some key considerations to be made regarding the *probabilistic* forecasts: first of all, the *coverage* statistic and the *Winkler Score* highlight the strengths of the BRNN. In particular, notwithstanding the challenging task, the *coverage* is almost perfectly aligned with the chosen α of 5% and, looking at Figure 4.3, the point forecast pattern mimics the dynamics of the original series and, in particular, can correctly recognize the volatility spikes.

Table 4.2: *Performance indicators* for €/$ exchange rate 5-days ahead forecast. While the point forecasts of the BRNN are worse than those of a naïve model, the probabilistic indicators show a better performance; in particular, the coverage is almost exactly on point with the credible intervals 5% nominal level, meaning that the corresponding quantiles can capture the overall variability of the target distribution.

Model	RMSE	MAE	Winkler Score	Coverage
Naïve	0.9999306	0.5879688	5.895383	1.00
Mean	1.1723362	0.6587535	239.469060	0.97
Simple VI	1.0147689	0.6363179	5.121619	0.95

By examining the target distribution and the predictive posterior (Figure 4.4), it is

Figure 4.3: *Dynamic forecast, €/$ exchange rate, 29-01-2024 to 03-02-2024.* The main highlight is that the BRNN can offer a remarkable probabilistic forecast, marching the spikes and declines in volatility,

clear how hard of a task was predicting such an irregular, multimodal, highly asymmetric density; this explains how simple benchmark methods can outperform highly sophisticated neural networks. In such an unpredictable environment, simply setting all forecasts to be the value of the last observation can have a remarkable effectiveness. In further applications, fine-tuning the *stochastic* model could potentially enhance the overall performance; however, even standard choices are capable of returning a decent result.

Figure 4.4: Actual and predicted densities comparison, €/$ exchange rate.

4.3 S&P 500 index

The challenge with this setting is trying to forecast a trended index *after a systemic shock*, such as COVID-19, using only past values of that index. As it is well known (Figure 4.5), the month of February 2020 was characterized by a steep descent, a fall in the value of the index caused by the onset of the pandemic. 3 days of March have been chosen as the test set, right at the moment in which a bounce back was happening: this setting is challenging because it needs the predictive model to be able to capture sudden *shifts*, not only trends and seasonality.

Figure 4.5: *Scaled realized range, closing price, and net change* for the S&P 500 index. While the range of the y-axis variable makes it hard to notice all data points, this width is a clear signal of the overall disruption of the markets, with the net change (subfigure 3) indicating many very pronounced downward movements.

Table 4.3: *Descriptive statistics* for S&P500 index . The observed distribution is asymmetric and leptocurtic.

Mean	Median	Sd	Skewness	Kurtosis
0.0014049	0.0013423	0.0011464	0.9258913	4.214088

In this case, a shorter training set and prediction window allowed us to compare the performance of the two different posterior approximations framework; as it has shown good results in Section 3.5, it is interesting to test whether it can be replicated in a more challenging scenario and if the result outweigh the extra computational cost involved.

Table 4.4: *Performance indicators* for S&P500 index 3-days ahead forecast. The VI-based BRNN takes the crown as the best performer for all indicators, showing in particular a coverage of almost 100%. The MCMC is not able to beat the benchmarks, exception made for the MAE, and is particularly lacking in the probabilistic aspects of its outputs.

Model	RMSE	MAE	Winkler Score	Coverage
Naïve	1.2937277	1.2636941	3.919955	1.000
Mean	0.8055183	0.8054364	35.016675	1.000
Simple VI	0.3637456	0.3602665	3.592977	1.000
Simple MCMC	1.0153938	0.6756231	8.594178	0.751

The Variational Inference-based BRNN decisively outperforms the benchmarks, showing a remarkable predictive power in a highly volatile setting; it is in particular notable the coverage of $\approx 100\%$, which means that after taking into account the numerical precision of this estimate almost all of the target data points falls inside the credible intervals. This can be attributed to the fact that the model correctly forecasted the overall increase in the market volatility and consequently the predicted posterior has wide tails, able to take into account the overall variation in the target distribution, even if one of the days in the dataset contained the maximum value for the scaled realized range.

While the MCMC-based model MAE is still the second-best, its RMSE is undercut by a mean forecast, and both probabilistic indicators show that the performance of the model is sub-par. The main reason can be immediately spotted by comparing the two predictive posteriors' densities with the actual distribution of the target variable; Figure 4.7 highlights that the latter is a highly irregular distribution, far

Figure 4.6: *VI-based model dynamic forecast S&P 500 index, 24/03/2024-26/03/2024.* In this case, the model outperforms the benchmarks; a closer examination of the charts shows how the slight shifts of the mean forecast can follow the dynamic of the target time series. Moreover, the credible intervals are wide enough to accommodate almost all data points, in such a highly volatile scenario.

Figure 4.7: *Actual and predicted densities* for the S&P 500 scaled realized range forecasts. The VI-based predicted posterior shows why it is the best performer under both point and probabilistic forecasts benchmarks: the mean is forecasted almost exactly and the fat tails can express the overall variability of the target, even though it completely lacks any hint of its tri-modality. The MCMC-based is bi-modal but fails to capture the overall scale and shape of its target, hence underperforming the variational approximation.

from the Gaussian proposed as a variational approximation. However, such an approximation does a better job by being centered closer to the target and by having fatter tails that can contain the extremes; in contrast, while the MCMC-based posterior is definitely far from a normal distribution and is also multi-modal, it is not centered correctly and fails to capture the overall variability. Being that such an imprecise approximation still required 1440×10^6 samples, it is a clear indicator of how cumbersome this approach can become and how it does not guarantee a sufficiently precise result.

Nevertheless, the MCMC-based BRNN shows some very interesting aspects and a powerful, yet unexpressed, potential: while being held back by the computational cost of the sampling process, it can output irregular and ill-behaved distributions that would be otherwise complex to model apriori, but are a better match for real data; it is important to remember that all the structural and stochastic features of this models are the same except for the approximation technique, and at the same time the output distribution is completely different from the almost-Gaussian density that the Variational Inference based models all predict regularly.

Figure 4.8 shows the consequences of this behavior: its forecasts are more reactive even if imprecise, following the dynamics of the objective more closely. Further research into more efficient and cost-effective MCMC sampling could unlock the full potential of this approach, which is very powerful and versatile even with minimum fine-tuning and no training.

Figure 4.8: *MCMC-based model dynamic forecast S&P 500 index, 24/03/2024-26/03/2024*. While imprecise, the model displays a reactive behavior to shifts and patterns in the target, mimicking patterns and especially trends. The 95% credible intervals fail to capture the overall variability: the probabilistic forecast is not on point, but the lower MAE than the benchmarks points out that the point accuracy, while inferior to the VI-based forecasts, is still more than acceptable.

5 Conclusions

Bayesian inference is an estabilished method that, as shown by its results on both simulated (Section 3.5) and real (Section 4) scenarios, effectively allows to enhance the capabilites of the Artificial Neural Network. The review of the literature on the subject unveiled a wide corpus of research and methods, unveiling well-known frameworks such as Variational Inference and the Metropolis-Hasting algorithm that are readily available and have been tested to both simulated and real datasets.

Critical applications require tools that are able to generalize well, without breaking or failing due to overconfidence or lack of generalization. Such failures might happen for various reasons, such as the lack of data, spikes in the overall volatility, or even for mishaps in the algorithms training process; in the worst case, they are inherent to the tool used, which weighs negatively on its usefulness and hinder its adoption.

These methods have been paired with a specialised architecture, such as the Long Short-Term Memory perceptron on which all the simulation of experiments of this thesis revolved, to unlock the potential of returning probabilistic forecasts, based on a full ensemble of possible parametrizations of complex, multi-parametric architectures that could natively only output point forecasts of the mean or given quantiles.

While not always the best performer, the Bayesian Recurrent Neural Network consistently beat the benchmarks, with the main highlights being its ability to return sensible and reasonable probabilistic forecast, its robustness to overfitting, and its adaptability to a wide range of challenging scenario with minimal fine tuning.

In such scenarios, having a tool able to forecast even without the help of specialized domain knowledge and models such as the perceptron is indeed full of potential. Using Bayes theorem to base these parametrizations on the available *evidence*, directly addresses one of the fundamental sources of *epistemic uncertainty* that the powerful yet over-confident Neural Network inherently has.

This thesis showed how the Bayesian Neural Network can fill this role, matching the foundational specialized function with an overarching statistical model, and how its ability to perform inference makes it a viable choice in such scenarios.

While still an imperfect tool, further research is testing even more powerful architectures and models; the fifth Makridakis competition (Makridakis, Spiliotis, and Assimakopoulos (2022)) has been the first to eminently feature machine learning models, which have been consistently among the best performers.

Further research on the approximation method, aimed at fine tuning their performance and improving the efficiency of the computations, would be able to overcome the biggest limitation of the most expensive methods, such as the Markov Chain Monte Carlo, which showed great ability to model complex and highly irregular target distributions: a quicker convergence or a reduction in the time that both the sampling process and the following computations require has a huge potential for applications in time series forecasting, in particular in those areas or situations in which a general-purpose functional model might be preferred to specialized, domain specific algorithms.

Abbreviations list

- ACF: AutoCorrelation Function.
- ANN: Artificial Neural Network.
- AR: Auto Regressive model.
- BBB: Bayes-By-Backprop.
- BDL: Bayesian Deep Learning.
- BNN: Bayesian Neural Network.
- BBBTT: Bayes-By-Backprop Through Time.
- BPTT: Back Propagation Through Time.
- BRNN: Bayesian Recurrent Neural Network.
- CEC: Costant Error Carousel.
- CNN: Convolutional Neural Network.
- DBN: Deep Belief Network.
- DGP: Data Generating Process.
- DMLP: Deep Multi-Layer Perceptron.
- ELBO: Evidence Lower BOund.
- GP: Gaussian Process.
- KL-Divergence: Kullback-Leibler Divergence.
- LSTM: Long-Short Term Memory.
- MAP: Maximum A Posteriori.
- MCMC: Markov-Chain Monte Carlo.
- MLE: Maximum Likelihood Estimation.
- MLP: Multi-Layer Perceptron
- NN: Neural Network.
- PACF: Partial Autocorrelation Function.
- PGM: Probabilistic Graphical Model.
- RBM: Restricted Boltzmann Machine.
- RNN: Recursive Neural Network.
- VI: Variational Inference.

List of Figures

List of Tables

Bibliography

Ahmad, Ahmad S, Mohammad Y Hassan, Md Pauzi Abdullah, Hasimah A Rahman, F Hussin, Hayati Abdullah, and Rahman Saidur. 2014. "A Review on Applications of ANN and SVM for Building Electrical Energy Consumption Forecasting." *Renewable and Sustainable Energy Reviews* 33: 102–9.

Beal, Matthew James. 2003. *Variational Algorithms for Approximate Bayesian Inference*. University of London, University College London (United Kingdom).

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B Shah. 2017. "Julia: A Fresh Approach to Numerical Computing." *SIAM Review* 59 (1): 65–98. https://doi.org/10.1137/141000671.

Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. 2015. "Weight Uncertainty in Neural Network." In *International Conference on Machine Learning*, 1613–22. PMLR.

Charnock, Tom, Laurence Perreault-Levasseur, and François Lanusse. 2020. "Bayesian Neural Networks." *arXiv Preprint arXiv:2006.01490*.

Chen, Luyang, Markus Pelger, and Jason Zhu. 2024. "Deep Learning in Asset Pricing." *Management Science* 70 (2): 714–50.

Chib, Siddhartha, and Edward Greenberg. 1995. "Understanding the Metropolis-Hastings Algorithm." *The American Statistician* 49 (4): 327–35.

Corsi, Fulvio, Stefano Peluso, and Francesco Audrino. 2015. "Missing in Asynchronicity: A Kalman-Em Approach for Multivariate Realized Covariance Estimation." *Journal of Applied Econometrics* 30 (3): 377–97.

Deb, Chirag, Fan Zhang, Junjing Yang, Siew Eang Lee, and Kwok Wei Shah. 2017. "A Review on Time Series Forecasting Techniques for Building Energy Consumption." *Renewable and Sustainable Energy Reviews* 74: 902–24.

Deng, Li. 2012. "The Mnist Database of Handwritten Digit Images for Machine Learning Research." *IEEE Signal Processing Magazine* 29 (6): 141–42.

Fortunato, Meire, Charles Blundell, and Oriol Vinyals. 2017. "Bayesian Recurrent Neural Networks." *arXiv Preprint arXiv:1704.02798*.

Gasthaus, Jan, Konstantinos Benidis, Yuyang Wang, Syama Sundar Rangapuram, David Salinas, Valentin Flunkert, and Tim Januschowski. 2019. "Probabilistic

Forecasting with Spline Quantile Function RNNs." In *The 22nd International Conference on Artificial Intelligence and Statistics*, 1901–10. PMLR.

Goan, Ethan, and Clinton Fookes. 2020. "Bayesian Neural Networks: An Introduction and Survey." *Case Studies in Applied Bayesian Data Science: CIRM Jean-Morlet Chair, Fall 2018*, 45–87.

Graves, Alex. 2011. "Practical Variational Inference for Neural Networks." *Advances in Neural Information Processing Systems* 24.

Han, Zhongyang, Jun Zhao, Henry Leung, King Fai Ma, and Wei Wang. 2021. "A Review of Deep Learning Models for Time Series Prediction." *IEEE Sensors Journal* 21 (6): 7833–48.

Hinton, Geoffrey E, and Drew Van Camp. 1993. "Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights." In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, 5–13.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-Term Memory." *Neural Computation* 9 (8): 1735–80.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1989. "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks* 2 (5): 359–66.

Hrasko, Rafael, André GC Pacheco, and Renato A Krohling. 2015. "Time Series Prediction Using Restricted Boltzmann Machines and Backpropagation." *Procedia Computer Science* 55: 990–99.

Hubel, David H, and Torsten N Wiesel. 1959. "Receptive Fields of Single Neurones in the Cat's Striate Cortex." *The Journal of Physiology* 148 (3): 574.

Jospin, Laurent Valentin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. 2022. "Hands-on Bayesian Neural Networks—a Tutorial for Deep Learning Users." *IEEE Computational Intelligence Magazine* 17 (2): 29–48.

Kiermayer, Mark, and Christian Weiß. 2021. "Grouping of Contracts in Insurance Using Neural Networks." *Scandinavian Actuarial Journal* 2021 (4): 295–322.

Kingma, Diederik P, and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization." *arXiv Preprint arXiv:1412.6980*.

LeCun, Yann, Yoshua Bengio, et al. 1995. "Convolutional Networks for Images, Speech, and Time Series." *The Handbook of Brain Theory and Neural Networks* 3361 (10): 1995.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2022. "M5 Accuracy Competition: Results, Findings, and Conclusions." *International Journal of Forecasting* 38 (4): 1346–64.

Martens, Martin, and Dick Van Dijk. 2007. "Measuring Volatility with the Realized

Range." *Journal of Econometrics* 138 (1): 181–207.

McCulloch, Warren S, and Walter Pitts. 1943. "A Logical Calculus of the Ideas Immanent in Nervous Activity." *The Bulletin of Mathematical Biophysics* 5: 115–33.

MINSKY, ML. 1969. "Perceptrons." *MIT Press*.

Piccinini, Gualtiero. 2004. "The First Computational Theory of Mind and Brain: A Close Look at Mcculloch and Pitts's 'Logical Calculus of Ideas Immanent in Nervous Activity'." *Synthese* 141: 175–215.

Posit team. 2024. *RStudio: Integrated Development Environment for r*. Boston, MA: Posit Software, PBC. http://www.posit.co/.

Qiu, Xueheng, Le Zhang, Ye Ren, Ponnuthurai N Suganthan, and Gehan Amaratunga. 2014. "Ensemble Deep Learning for Regression and Time Series Forecasting." In *2014 IEEE Symposium on Computational Intelligence in Ensemble Learning (CIEL)*, 1–6. IEEE.

R Core Team. 2023. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rangapuram, Syama Sundar, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. 2018. "Deep State Space Models for Time Series Forecasting." *Advances in Neural Information Processing Systems* 31.

Rosenblatt, F. 1957. "The Perceptron - a Perceiving and Recognizing Automaton." 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. 1986. "Learning Representations by Back-Propagating Errors." *Nature* 323 (6088): 533–36.

Salakhutdinov, Ruslan, Andriy Mnih, and Geoffrey Hinton. 2007. "Restricted Boltzmann Machines for Collaborative Filtering." In *Proceedings of the 24th International Conference on Machine Learning*, 791–98.

Salinas, David, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. "DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks." *International Journal of Forecasting* 36 (3): 1181–91.

Sezer, Omer Berat, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. 2020. "Financial Time Series Forecasting with Deep Learning: A Systematic Literature Review: 2005–2019." *Applied Soft Computing* 90: 106181.

Shannon, Claude Elwood. 1948. "A Mathematical Theory of Communication." *The Bell System Technical Journal* 27 (3): 379–423.

Spiegelhalter, David. 2019. *The Art of Statistics: Learning from Data*. Penguin UK.

Tantau, Till. 2013. *The TikZ and PGF Packages: Manual for Version 3.0.0*. http://

sourceforge.net/projects/pgf/.

Teräsvirta, Timo, Dick Van Dijk, and Marcelo C Medeiros. 2005. "Linear Models, Smooth Transition Autoregressions, and Neural Networks for Forecasting Macroeconomic Time Series: A Re-Examination." *International Journal of Forecasting* 21 (4): 755–74.

Tishby, Levin, and Solla. 1989. "Consistent Inference of Probabilities in Layered Networks: Predictions and Generalizations." In *International 1989 Joint Conference on Neural Networks*, 403–9. IEEE.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. "Attention Is All You Need." *Advances in Neural Information Processing Systems* 30.

Wang, Hao, and Dit-Yan Yeung. 2020. "A Survey on Bayesian Deep Learning." *ACM Computing Surveys (Csur)* 53 (5): 1–37.

Wang, Yuyang, Alex Smola, Danielle Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. 2019. "Deep Factors for Forecasting." In *International Conference on Machine Learning*, 6607–17. PMLR.

Wegner, Enrico. 2023. *BayesFluxR: Implementation of Bayesian Neural Networks*. https://CRAN.R-project.org/package=BayesFluxR.

Wen, Qingsong, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. 2022. "Transformers in Time Series: A Survey." *arXiv Preprint arXiv:2202.07125*.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Winkler, Robert L. 1972. "A Decision-Theoretic Approach to Interval Estimation." *Journal of the American Statistical Association* 67 (337): 187–91.

Zeng, Ailing, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. "Are Transformers Effective for Time Series Forecasting?" In *Proceedings of the AAAI Conference on Artificial Intelligence*, 37:11121–28. 9.