Forecasting Lightpath Quality Of Transmission And Implementing Uncertainty In The Forecast Models

by

Somaiieh YOUSEFI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLEMENT FOR A MASTER'S DEGREE WITH
THESIS IN ELECTRICAL ENGINEERING
M. Sc.

MONTREAL, DECEMBER 13TH, 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS (THESIS M.SC.)

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Professor Christine Tremblay, Thesis Supervisor
Department of Electrical Engineering, École de technologie supérieure

Professor Christian Desrosiers, Thesis Co-supervisor
Department of Software and Information Technology Engineering, École de technologie supérieure

Professor Ouassima Akhrif, President of the jury
Department of Electrical Engineering, École de technologie supérieure

Doctor Petar Djukic, Member of the jury
Manager of AI Department, Ciena Company

Professor Nadjia Kara, Member of the jury
Department of Software and Information Technology Engineering, École de technologie supérieure

THIS THESIS WAS PRENSENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

NOVEMBER 22ND, 2021

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGMENT

I would like to sincerely acknowledge my supervisors Professor Tremblay and Professor Desrosiers, who gave me the opportunity to do this project at Ciena, and for their time to lead the project.

Moreover, I would like to express my special thanks of gratitude to Doctor Petar Djukic and his professional team at Ciena, specially to Doctor Firouzeh Golaghazadeh. Doctor Petar Djukic did a great precise mentoring leading me to learn enormously and develop my abilities beyond what I thought before joining his group. He helped me step by step to promote my competence along the project. His sense of responsibility for my degree and his management, was exceeding a supervisor, which will be never forgettable for me.

Prédiction de la qualité de la transmission des connexions optiques et mise en œuvre de l'incertitude dans les modèles de prédiction

Somaiieh YOUSEFI

RÉSUMÉ

La popularité récente de l'utilisation des modèles d'apprentissage profond pour la prédiction des séries chronologiques requiert de prédire avec exactitude la cible mais également mesurer l'incertitude de la prédiction. Ainsi, une architecture intuitive N-Beats a été implémentée dans cette thèse pour propriétés la rapidité de l'entraînement du modèle et l'adaptabilité de paramètre. De plus, cette architecture ne nécessite pas de modifier les différents paramètres du tableau de séries temporelles. L'impact des hyper-paramètres sur les performances de notre modèle a été également étudiée. Finalement, la performance de notre modèle a été comparée à d'autres modèles, à savoir Long Short-Term Memory (LSTM), Multi Layer Perceptron (MLP) et une méthode naïve. Les résultats ont montré que notre modèle N-Beats a surpassé le modèle LSTM. Toutes les implémentations ont été réalisées en utilisant le langage de programmation Python. Notre base de données utilisée comprend des ensembles de données de deux opérateurs de réseau, Microsoft et NASP. Un échantillonnage aléatoire de nos ensembles de données a été effectué pour éviter un sur-apprentissage.

L'estimation de l'incertitude est une mesure des modèles d'apprentissage automatique pour déterminer la fiabilité de la prédiction des séries chronologiques. Ainsi, pour mesurer l'incertitude et le niveau de confiance du modèle MLP, le *dropout* Monte Carlo, qui se rapproche de l'incertitude bayésienne, a été appliqué lors de l'inférence. La régression quantile a été implémentée comme modèle de référence pour prédire les intervalles de confiance et pour évaluer notre stratégie d'estimation de l'incertitude. En conséquence, les modèles d'approximation bayésienne ont indiqué la nécessité d'un étalonnage pour ajuster la probabilité prédite.

Mots-clés: Séries temporelles, Forecaster, N-Beats, LSTM, MLP, Hyper-paramètres, Incertitude, Approximation bayésienne, Intervalles de confiance, Régression quantile

Forecasting lightpath quality of transmission and implementing uncertainty in the forecast models

Somaiieh YOUSEFI

ABSTRACT

The recent popularity of using Deep Learning models for the forecasting of time series calls for methods to not only predict the target but also to measure the uncertainty of the prediction accurately. Working with time series requires reliable forecasters. An intuitive N-Beats architecture was implemented in this thesis with the desirable properties of being fast to train and adaptable, without requiring modifications for the various time series array settings. Its performance was compared with other models, namely Long Short-Term Memory (LSTM) Multi-layer Perceptron (MLP) and a naïve method. The influence of different hyperparameters were investigated as to how the parameters affected the performances of the models. Our N-Beats model outperformed well-known time series forecaster, LSTM. All the implementations were conducted in Python programming language. Random sampling was performed to avoid overfitting. Our target field data are Microsoft and NASP data sets.

Estimating uncertainty is a component of reliable machine leaning models for time series forecasting. To measure the uncertainty and confidence level of the selected MLP model, Monte Carlo dropout, which approximates Bayesian uncertainty, was applied during inference. Quantile Regression was also implemented on the MLP algorithm as a baseline to predict the confidence intervals and to evaluate our strategy for estimating uncertainty. As a result, Bayesian approximation models indicated the necessity of calibration to adjust the predicted probability.

Keywords: Time series, Forecaster, N-Beats, LSTM, MLP, Hyperparameter, Uncertainty, Bayesian approximation, Confidence intervals, Quantile regression

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABREVIATIONS

ANN	Artificial Neural Network
BER	Bit ratio
ML	Machine Learning
DL	Deep Learning
KNN	K-Nearest Neighbors
LSTM	Long Short-Term Memory
SVM	Support Vector Machine
MLP	Multilayer Perceptron
RNN	Recurrent Neural Network
MSE	Mean Squared Error
MAE	Mean Absolute Error
sMAPE	Symmetric Mean Absolute Percentage Error
SNR	Signal to Noise Ratio
TS	Time Series
GP	Gaussian Process
QoT	Quality of Transmission
Q-factor	Quality factor
KL	Kullback-Leibler
NASP	North American Service Provider
MAD	Median Absolute Deviation

INTRODUCTION

The increasing rate of data traffic due to the popularity of video and cloud applications, as well as emerging 5G and internet of things (IoT) technologies, call for higher traffic volumes. The increasing data volume makes the impact of degradations of lightpath performance and equipment failure more and more important. It is crucial to have reliable networks. The lightpath, in the optical networks, is a WDM channel carried from the source to the destination, in which no optical conversion occurs in between the source and the destination (Choi & Yang, 2008).

A survivable telecommunication network is a network that withstands the faults and attacks of the link or the equipment (Choi & Yang, 2008). The approach of the telecommunication companies to have a survivable network is to forecast the lightpath performance. Forecasting is defined as the process of predicting future (future trends) regarding the past sets of information as the history. The data from the lightpaths are in the form of the time series (TS) which are sequences of data occurring over time. Time series have a wide range of applications from monitoring industrial processes (sensor operations, optical line paths, etc.), economy, management (*11.3 Neural Network Models | Forecasting*, n.d.) to tracking business trends. Forecasting times series is important because of the number of prediction problems in critical areas like telecommunications that have a temporal component (Brownlee, 2016)(Oreshkin et al., 2020a). Therefore, having more accurate predictions can have a huge financial impact, in the order of millions of dollars on businesses (Oreshkin et al., 2020a).

Machine learning (ML) algorithms and deep learning (DL) techniques are now outperforming the classical statistical methods for time series forecasting (Oreshkin et al., 2020a). ML algorithms such as K-nearest neighbors (KNN), support vector machines (SVM) and random forest (RF) have been applied recently for forecasting the quality of transmission (QoT) of lightpaths carried in optical networks (Aladin & Tremblay, 2018)(*A Survey on Application of Machine Learning Techniques in Optical Networks*, n.d.; Choudhury et al., 2018; Wang et al., 2017). These algorithms also have been used for predicting performance metrics like signal-

to-noise ratio (SNR) or (Q-factor) for the lightpaths, as well as equipment failure (Choudhury et al., 2018; Wang et al., 2017). A long short-term memory (LSTM) DL method has been applied to predict reconfigurable add-drop multiplexer (ROADM) requirements in the network 30 minutes in advance (Mo et al., 2018). Aladin et al. (Aladin et al., 2020), by applying SVM and artificial neural networks (ANN), showed the promising application of ML for estimating lightpath QoT before establishment and for predicting the QoT of deployed lightpaths. Oreshkin et al. (Oreshkin et al., 2020a) proposed a DL algorithm which uses backward and forward residual links as well as fully-connected layers in the form of deep stacks. Their N-Beats model achieved a higher accuracy compared to the previous models on the same target time series.

The application of conventional ML or DL models to address the regression problems involves predicting the expected value of a future observation. However, it is also essential for the forecasting algorithms to output the confidence of their predictions. For instance, in making critical and safety related decisions, such as in self-driving cars (or autonomous cars), the confidence of the prediction must be taken into account (Levi et al., 2020). Model uncertainty provide us with the means to deal with the uncertain input data (Gal & Ghahramani, 2016b). Uncertain prediction in classification can also determine the decisions which require additional validation by a human (Linda et al., 2009). To estimate uncertainty, a distribution of targets over the target domain is required for each prediction during inference (Levi et al., 2020). Gal et al. (Gal & Ghahramani, 2016b) developed a mathematical tool for introducing uncertainty to the model, which is an approximation to the Bayesian probability theory as well as the Gaussian process (Rasmussen, 2004).

In this project, various types of ML algorithms are studied including feed forward, recurrent neural networks (RNN) and residual neural networks (ResNet). Commonly used ANN forecasters for time series are multilayer perceptron (MLP), LSTM and more recently the N-Beats model (Oreshkin et al., 2020a). Among these models, the MLP is simple to implement, LSTM has a library, but N-Beats is not available yet as a library. To use this algorithm, implementing it is necessary. Therefore, the first objective is to find an appropriate forecaster

for the target time series. The forecaster that we choose should have a certain degree of accuracy. This is what we investigate by comparing with the baseline results as the first objective of this research thesis. We select MLP, LSTM and N-Beats forecasters, analyze and compare their predicting performances with the baseline that we choose to be a naïve model. The results can be compared with that of Chouman et al. (Chouman et al., 2021) who have used logistic regression, MLP and LSTM models to predict the lightpath QoT and the minimum lightpath QoT for horizons up to 72 hours. According to the performance results, we select the most accurate forecaster for the next step which is estimating uncertainty using the dropout method at test time. To implement this method, dropout must be applied during the inference time. In the neural networks, dropout is used during training as a regularization technique to avoid overfitting. Using dropout while training will result in point prediction. However, deploying this method during inference time can result in variational distribution. The upper and lower bounds of the prediction can be obtained from the margins of this distribution. As the baseline for estimating confidence margins, we develop the deep quantile regression method, which is a statistical tool to estimate conditional median and demonstrate the distribution of the mass of data around the median. Finally, we provide a quantitative assessment of the methods for uncertainty estimation.

Identification of the research problem

The need to find accurate ML forecasters to predict the future values of the optical signal, with the purpose of improving reliability of the networks, is the first problem to be addressed in this thesis. The next important problem is the necessity to define the confidence level of the forecast by the selected ML model. We intend to predict the confidence margins. In other words, we also need the degree of confidence of our predictions.

Research objectives

The objectives of this project can be summarized as follows: implementing deep neural network algorithms to forecast lightpath SNR in different time horizons (e.g., 1-h, 2-h, 4-h,

etc.); comparing the implemented algorithms in terms of capability of precise forecasting by evaluating them on the Microsoft and NASP data sets; developing and validating strategies to measure the prediction uncertainty of the forecasting models; using the forecasting models and the uncertainty estimation to define a lower margin for signal drop.

Thesis structure

In this thesis, the research topics are structured in the following chapters. Chapter 1 consists of extensive background research and literature survey and the strategies that have been applied so far to address the lightpath forecasting problem. Chapter 2 describes the most efficient strategies of forecasting time series and the implementation of forecast models in more details. It also contains the results of the analysis of the performance of each forecaster. Chapter 3 describes the selected means of uncertainty in predicting lightpath SNR. The last part of the thesis includes the conclusion and the recommendations for future work.

CHAPTER 1

BACKGROUND INFORMATION AND LITERATURE SURVEY

This chapter includes important definitions and background information about lightpath QoT metrics, which can be collected in the form of time series in optical networks, as well as the methods to forecast lightpath QoT by using ML methods. In addition, this chapter addresses the uncertainty of the time series forecasting models.

1.1 Quality factor

Bit error rate (BER) and quality factor (Q-factor) are the main factors defining the quality of digital signals in the telecommunication systems. These two parameters are related together (Hui, 2019).

BER is a key factor in the quality of communication. It is also known as probability of digital bits (Hui, 2019) and is defined as:

$$BER = \frac{Bit_{error}}{Bit_{total}} \tag{1.1}$$

where number of misinterpreted bits and total number of received bits are denoted by Bit_{error} are Bit_{total}, respectively. It is important to note that both the misinterpreted bits and total bits are measured in the same time frame. The Q value (or Q-factor) is a good alternative for BER. Q value is a quality factor defined by the separation between the digital signal an Gaussian noise (Hui, 2019).

In addition, in optical communications, the signal-to-noise ratio (SNR) determines the quality of the signal. To obtain a high SNR, it is important to have low noise level (Hui, 2019).

Lightpath QoT depends on the characteristics of the transmission system, of the link, and of the WDM signal. Estimation of these metrics before lightpath establishment can be performed by using analytical models which generally require a high computation time and a precise knowledge of the system, link and channel parameters (Aladin & Tremblay, 2018). Forecasting lightpath QoT for deployed lightpaths would allow network operators to trigger proactive maintenance by detecting degradations before they actually occur. Fig. 1.1 and Fig. 1.2 show the Q-factor and SNR time series over four days of one WDM channel of Microsoft data set, and one channel from the North American Service Provider (NASP) data sets, respectively.

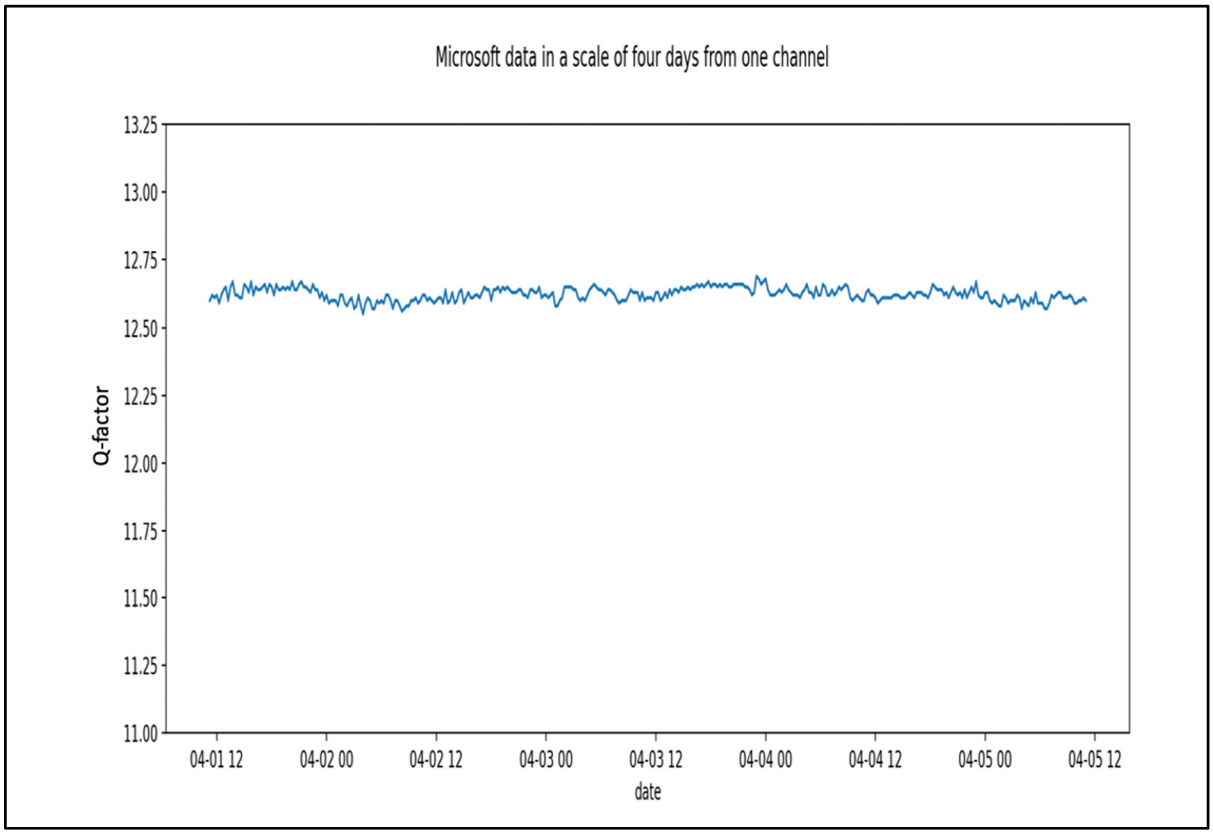

Figure 1.1 Q-factor time series for one WDM channel, Microsoft data set, four days data

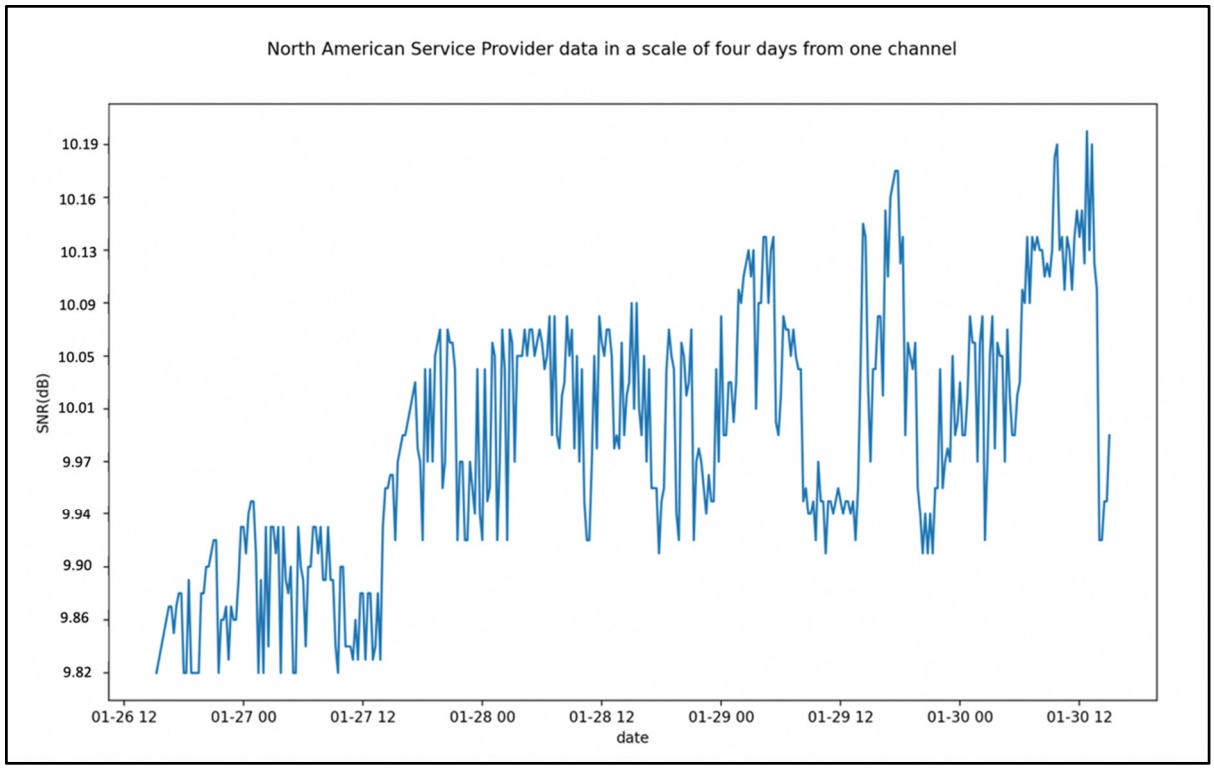

Figure 1.2 SNR time series for one WDM channel, North American Service Provider data set, four days data

Forecasting is a common statistical task in business that helps making decisions to schedule the production and guides for future planning (*1.6 The Basic Steps in a Forecasting Task | Forecasting*, n.d.) . In this project, forecasting will help us to predict and have control over the Q-factor in the optical networks.

1.2 Time series forecasting

Capturing genuine patterns and relations existing in the historical data in forecasting time series is of great importance. Forecasting can differentiate between random fluctuations in the past and genuine patterns. A good forecast can prevent replicating past events (including failure of the equipment or link, performance degradation) not to happen again (R. J. Hyndman et. al., Forecasting: principles and practice, chapter 1, section 1.1, 2014). Forecasting methods could be simple, such as considering the most recently occurred events as the forecast for the future

(naïve method), or complex, like using neural networks. The choice of the technique depends on the predictability of the quantity to be forecasted as well as the type of the available data. For instance, if there is no data available, or the data are not relevant, then qualitative forecast might be used. Quantitative forecasting is employed once two conditions are met: (a) numerical information about the past of the data are available (history). (b) It could be assumed that some aspects of past patterns will be continued in the future. In each quantitative forecasting the following basic steps are common (R. J. Hyndman et. al., Forecasting: principles and practice, chapter 1, section 1.6, 2014): (1) Problem definition: This is the hardest part of forecasting; To understand the problem well and to know the system or the organization as well as how to use forecasting. (2) Gathering information: This step requires statistical data and the expertise of the people who collect the data and will use the forecast. (3) Preliminary analysis: The analysis will start drawing diagrams and seeing if there is any meaningful trend. (4) Choosing and fitting models: Choosing the best model depends on the availability of data and the strength of correlation between the forecasted variable and other variables. To assess the validity of the forecasting results, generally, two or three models will be compared using different metrics. (5) Using the model and evaluating via some special methods: Once the model has been selected and its parameters have been estimated, the model is ready to forecast.

Visualizing of plotted time series in Python or R could help us to obtain more information about them. From the plotted time series, the frequency of the data that could be annual, quarterly, monthly or weekly, can be observed. The time series can be decomposed into their main components:

Trend

The trend is when there is a long-term increase or decrease of the data. Trend is not always linear, but it could be a path of data which is nonlinear. There could be a “changing direction” when the data changes from the increasing to the decreasing trend (*2.3 Time Series Patterns | Forecasting*, n.d.).

Seasonal

A pattern is seasonal when a time series is affected by a repeating time interval (e.g., the season of the year or the day of the week) and always has a "fixed and known frequency" (*Chapter 1 Getting Started | Forecasting*, n.d.). Fig. 1.3 shows an example of the seasonal behavior of the consumption of anti-diabetic drug in Australia. The consumption of drugs shows a considerable drop each year in February from 1992 to 2008.

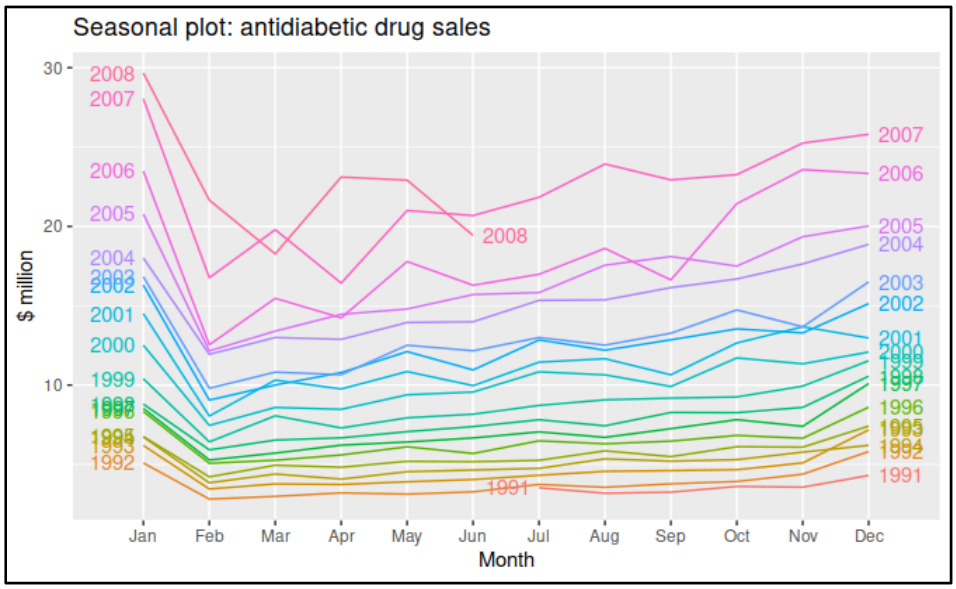

Figure 1.3 Seasonal data that shows the consumption
of anti-diabetic drug in Australia
Taken from (*11.3 Neural Network Models | Forecasting*, n.d., p. 3).

Cyclic

The cyclic trend is when the data rises and falls but not in a fixed frequency manner. These fluctuations are related to the business situations and are known as "business cycle" expression (*2.3 Time Series Patterns | Forecasting*, n.d.).

1.3 Components of the time series

Time series might have some components. The decomposition of the time series a could be written as in Eq. (1.2) or (1.3) given the seasonality and trend. It is also possible that the time series does not contain an of these characteristics.

$$y_t = S_t + T_t + R_t \tag{1.2}$$

where y_t is the data, S_T is the seasonal component, T_T is the trend component and R_T is the remainder component at time period t. It is also possible to write y_t as:

$$y_t = S_t \times T_t \times R_t \tag{1.3}$$

The first decomposition is more appropriate when the magnitude of the variation in seasonal term or the variation around the trend cycle does not change with the level of the time series. Otherwise, when the variation in the seasonal pattern or the variation around the trend cycle is in the range of the time series, the second definition is more applicable. An alternative way of the second definition is the logarithmic definition (Eq. 1.4) (*6.1 Time Series Components | Forecasting*, n.d., p. 1).

$$\log y_t = \log(S_t) + \log(T_t) + \log(R_t) \tag{1.4}$$

Having introduced time series, now the available methods for forecasting time series could be introduced:

1.4 Average method

The simplest forecaster would be taking the mean of the history of the data (Eq. 1.5):

$$\hat{y}_{T+h|T} = \bar{y} = (y_1 + y_2 + \cdots + y_T)/T \tag{1.5}$$

in this equation h denotes the forecast horizon and T denotes the history time.

1.5 Moving average method

Moving average is a classical way of time series decomposition originated in the 1920s and is still the basis of many of the decomposition methods for the time series. For more information about the calculations of the moving average, one should refer to the book (*6.2 Moving Averages | Forecasting*, n.d., p. 2).

1.6 Naïve method

In this method, we consider the last value of one variable to be the forecast of that variable. This method is practical for many economic and financial time series. Because naïve is appropriate once data has a random walk. These forecasters are also named "random walk" forecasters. (*3.1 Some Simple Forecasting Methods | Forecasting*, n.d.).

$$\hat{y}_{T+h|T} = y_T \tag{1.6}$$

In this equation, h represents the forecast horizon and T denotes the history time.

1.7 Seasonal naïve method

Naïve method could also be applied for highly seasonal data. Meaning that we can set each forecast to be equal to the "last value of the same season of the year" (*3.1 Some Simple Forecasting Methods | Forecasting*, n.d.). The seasonal naïve method for the time T+h could be written as:

$$\hat{y}_{T+h} = y_{T+h} - m(k+1) \tag{1.7}$$

Where h is the forecast horizon, T is the history time, m is the seasonal period and k is the integer part of the $(h-1)/m$ (the number of complete years before the period T+h.

1.8 Facebook prophet

Prophet is a forecasting model provided by Facebook, for univariate time series, in which nonlinear trends could be fit with daily, seasonal, weekly or yearly effects considering the holidays (*Quick Start*, n.d.). Facebook team has provided a forecaster that can predict the invitation of the events that people put in their Facebook calendar. These events are affected by the season of the year or days of the week. There are two themes of forecasting in the industry. First, completely automatic forecast, which is difficult to tune, and mostly inflexible. Therefore, it is hard to associate useful assumptions or heuristic. Second, common forecasters in the data science are specialized in a specific domain but not in the time series.

1.9 Regression models for the time series

Basically, if we forecast the time series y, assuming that it has a linear relationship with other time series x, this would be the concept of linear regression. The forecast variable y is called the "regressand" while the predictor variable is called the "regressor" (*Chapter 5 Time Series Regression Models | Forecasting*, n.d.). Regression has two types: linear regression and logistic regression.

1.9.1 Linear regression

One of the simplest and most widely used regression methods is linear regression. This implies that the response is constructed of linear function of the inputs (Murphy, 2012) (Eq. 1.8).

$$y(x) = w^T x + \epsilon = \sum_{j=1}^{D} w_j x_j + \epsilon \tag{1.8}$$

where $w^T x$ is representing the inner scalar product between input vector x and the weight's vector, and ϵ is the residual error between the linear prediction and the true value (Murphy, 2012). Commonly, we assume that ϵ has normal or Bell shape distribution.

1.9.2 Logistic regression

One can generalize the linear regression to the binary classification by applying two changes: First, by replacing the Gaussian distribution of y with Bernoulli distribution, which is more appropriate for binary cases, $y \in \{0, 1\}$: $p(y|x, w) = \mathrm{Ber}(y|\mu(x))$, where $\mu(x) = \mathbb{E}[y|x]$, x is the observed random sample, and w could be a real unknown parameter or vector of problems. Second, a linear combination of the inputs has to be calculated and be passed through a function called **sigmoid**: $\mu(x) = \mathrm{sigm}\,(w^T x)$

Sigmoid function is a form of logistic function denoted by sigm(η) (Eq. 1.9). This function has a S shape and maps every entry in the domain $(-\infty, +\infty)$ to the range (0, 1).

$$\mathrm{Sigm}\,(\eta) = \frac{1}{1 + \exp(-\eta)} \tag{1.9}$$

If the two mentioned steps are performed together, Eq. 1.10 will be generated:

$$p(y|x, w) = \mathrm{Ber}(y|\mathrm{sigm}\,(w^T x)\,) \tag{1.10}$$

This is called **logistic regression** due to its similarities to the linear regression, However, it is a form of classification not regression (Murphy, 2012).

1.9.3 Evaluation of the regression models

The difference between the observed y and the corresponding fitted values $\hat{y}$ are the training set error or the residuals that, for the linear regression could be defined as:

$$e_t = y_t - \hat{y}_t \tag{1.11}$$

$$= y_t - w_0 - w_1 x_{1,t} - w_2 x_{2,t} \ldots - w_k x_{k,t}$$

for t = 1,…, T.

In this equation e_t is the error or the residual, y_t is the true value of y as the function of x, $\hat{y}_t$ is the forecast of y and w_0 to w_k are the coefficients.

The residuals have the following properties: the sum of the residuals is zero ($\sum_{t=1}^{T} e_t = 0$) and $\sum_{t=1}^{T} x_{k,t} e_t = 0$ for all k. Meaning that the average of the residuals is zero, and also the correlation between the residuals and the observation respecting the predictor variable is zero.

After fitting the regression model, plotting the residuals helps to assess if the assumptions of the model have been satisfied.

1.10 Advanced forecasting models

In this section, some of the most popular advanced forecasting methods will be discussed:

1.10.1 ARIMA generative model

ARIMA model is one of the most well-known models in time series forecasting. ARIMA models aim to describe the autocorrelation between the data (*Chapter 8 ARIMA Models | Forecasting*, n.d.). To define how ARIMA works, some signal format and terms need to be defined:

Differencing

A time series is stationary when its properties are not depending on the time of observation (*8.1 Stationarity and Differencing | Forecasting*, n.d., p. 1). Therefore, time series with trends

or seasonality are not stationary. The process of calculating the difference between consecutive observations in the non-stationary time series, is called differencing.

Autoregressive models

A multi-regression is a combination of some predictors. Autoregressive is equivalent to use a linear combination of past values of that variable. By combining the differencing with autoregression and moving average models, the result would be non-seasonal ARIMA. ARIMA is obtained from Autoregressive, Integrated Moving Average (*8.3 Autoregressive Models | Forecasting*, n.d.).

1.10.2 Neural Network (NN) methods

The most recent method for forecasting time series is using artificial neural networks (ANN). Artificial neural networks are based on simple mathematical models of the neurons in the brain and they allow more complex relationship between the predictors (inputs) and the response variables (*11.3 Neural Network Models | Forecasting*, n.d., p. 3).

Neural networks have various structures and are the basis of ML. Prior to define the neural networks and the architectures, artificial intelligence (AI) and its relation with ML and DL will be introduced. Then selected DL models for the time series will be explained with more details.

Artificial Intelligence

AI was born in 1950s when pioneers tried to devise a machine that can think. Another definition is "The effort to automate intellectual tasks that the human normally do" (Lambrugh, n.d.). Therefore, AI is a general field that contains ML and DL in addition to some areas that basically do not involve any learning, including programming (Fig. 1.5). For instance, early chess programs, were only concrete lines of code without using machine learning. Machine learning differs from programming where input data according to certain programming rules

deliver the answer. In machine learning, the approach is to give the input data as well as the answers to the machine and the outcome is the rules (Fig. 1.6). Machine learning started flourishing in 1990s and became most popular method among all of the AI fields very quickly (Lambrugh, n.d.)

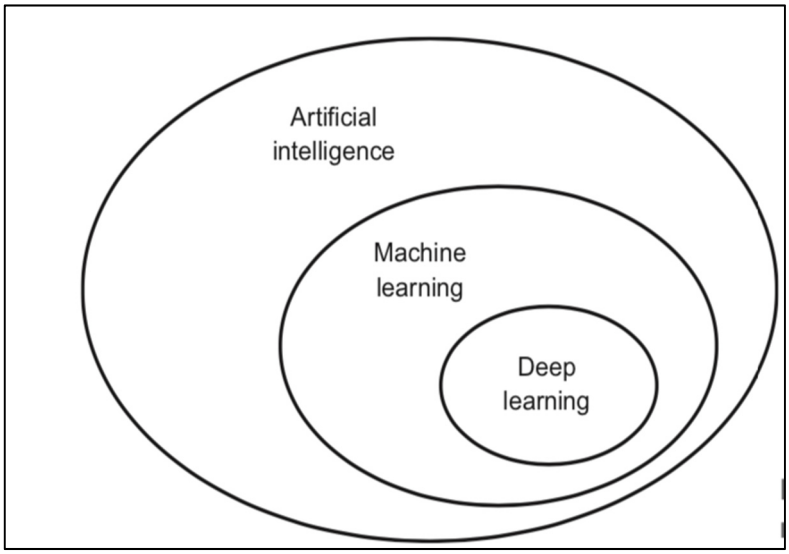

Figure 1.4 Artificial intelligence, machine learning and deep learning
Taken from Lambrugh (2020)

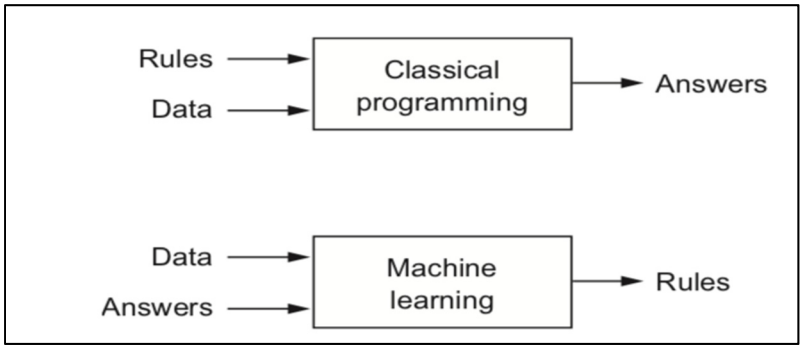

Figure 1.5 Machine learning versus programming
Taken from Lambrugh (2020)

The architecture of the neural network models

The neural network could be considered as a network of the neurons organized in some layers (*11.3 Neural Network Models | Forecasting*, n.d.). The predictors (inputs) are in the bottom layer and the forecasts (the outputs) are in the top layer. There are also some intermediate layers called hidden layers.

The simplest network doesn't consist of any hidden layer and is equivalent to the linear regression (Fig. 1.6). The coefficients of the input neurons are the weights" (*11.3 Neural Network Models | Forecasting*, n.d., p. 3).

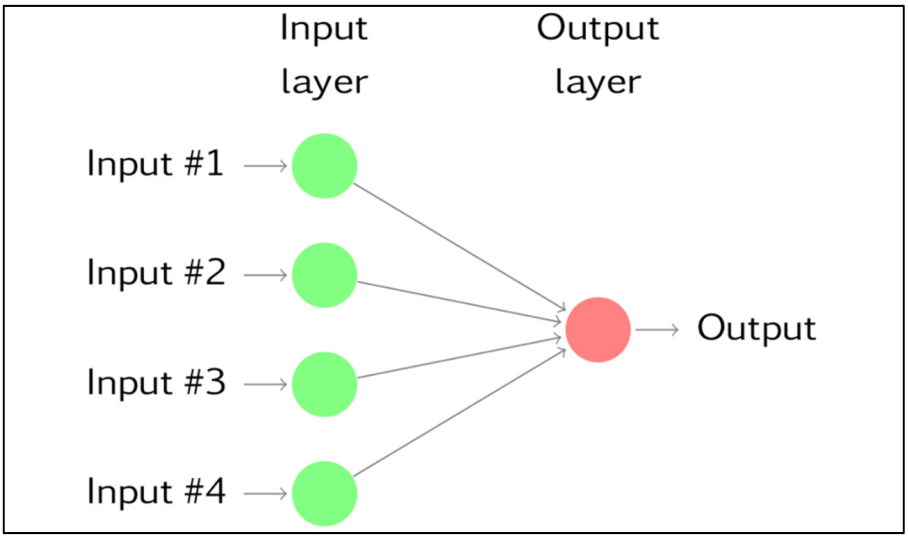

Figure 1.6 One-layer neural network with four inputs
Taken from (*11.3 Neural Network Models | Forecasting*, n.d., p. 3).

The linear regression or classification models are based on linear combination of some nonlinear basis functions $\phi_j(x)$. The output is in the form (Eq. 1.12):

$$y(x, w) = f\left(\sum_{j=1}^{M} w_j \, \phi_j(x)\right) \tag{1.12}$$

where $f(.)$ is a nonlinear activation function for classification and is the identity for regression problems (Bishop, 1994). The goal is to extend the model by making the basis $\phi_j(x)$ functions depend on some parameters and allow adjusting the parameters, together with the coefficients $\{w_j\}$, during the process of training (Bishop, 1994). Neural networks use the basis functions as denoted in Eq. 1.12.

A basic neural network governs sets of these functional transformations (Bishop, 1994). In the simplest form, a combination of the input variables $x_1, x_2, \dots, x_D$ will be constructed:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{1.13}$$

where $j = 1, \dots, M$ and the superscript (1) are indicating that the corresponding parameter (w) is in the first layer of the network (Bishop, 1994). In this equation, $w_{j0}^{(1)}$ indicates the bias parameter and a_j represents the activations. Each of these activations then will be transformed using nonlinear activation functions $h(.)$(Eq. 14). (Bishop, 1994). In neural networks these functions that receive the input from the previous layer, are called the hidden units (Fig. 1.7). The function $h(.)$ could be sigmoid function, or 'tanh'. The output of the network is the linear combination of these values (Eq.16).

$$z_j = h(a_j) \tag{1.14}$$

This function is a nonlinear function and is an input for the basis functions in Eq. 13 (Eq. 1.15).

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{1.15}$$

where $k = 1, \dots, K$, and K is the number of outputs. The superscript (2) is showing that the transformation corresponds to the second layer, where $w_{k0}^{(2)}$ is the bias parameter of this layer.

To generate the final output, the a_k should pass through another activation function. For the regression problems, the activation function is identity $y_k = a_k$. In the similar way, for the classification problems, the activation could be sigmoid function (Eq. 1.9) (Bishop, 1994).

The combination of these various stages maps the inputs to the output. In case of sigmoidal activation functions, the relation could be written as (Bishop, 1994):

$$y_k(x, w) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{1.16}$$

Therefore, according to the Eq. 1.21, the output of the neural network is the nonlinear function of the inputs, weights, and the biases.

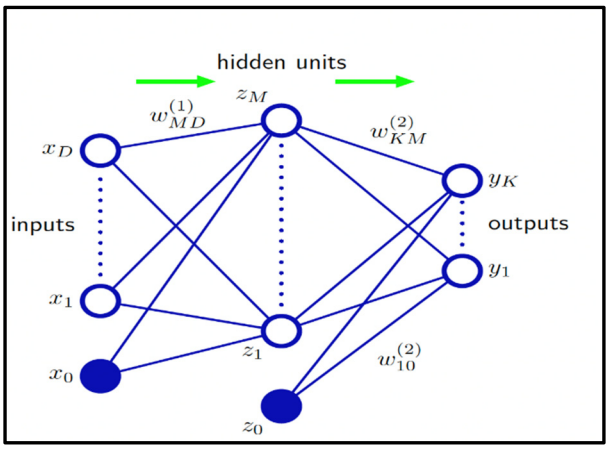

Figure 1.7 A neural network with a hidden layer
Taken from (Bishop, 1994)

1.11 Deep Learning (DL)

Deep learning is one of the specific fields of the machine learning that puts the emphasis on learning a hierarchical representation of features. The expression “deep learning” is the approach of achieving successive layers of representation in a neural network. The number of layers contributing this learning is defining the depth of the model (Lambrugh, n.d.). Other

names are “layered representation learning” and “hierarchical representation learning”. In deep learning, these neural network layers are stacked on the top of each other. In Fig. 1.8, using four layers, the network is able to define the digit in the input image.

Modern deep learning these days have tens or hundreds of successive layers. Other approaches towards machine learning focus on one or two layers which is called “shallow learning”. The standard deep learning network is called **feedforward** because the flow of information from x, the input, passes through intermediate computations to make compute the output $y = f^*(x)$ without any feed-back (*Deep Learning*, n.d.). When feedback is connected to the network, the network is called a “recurrent network” (*Deep Learning*, n.d.).

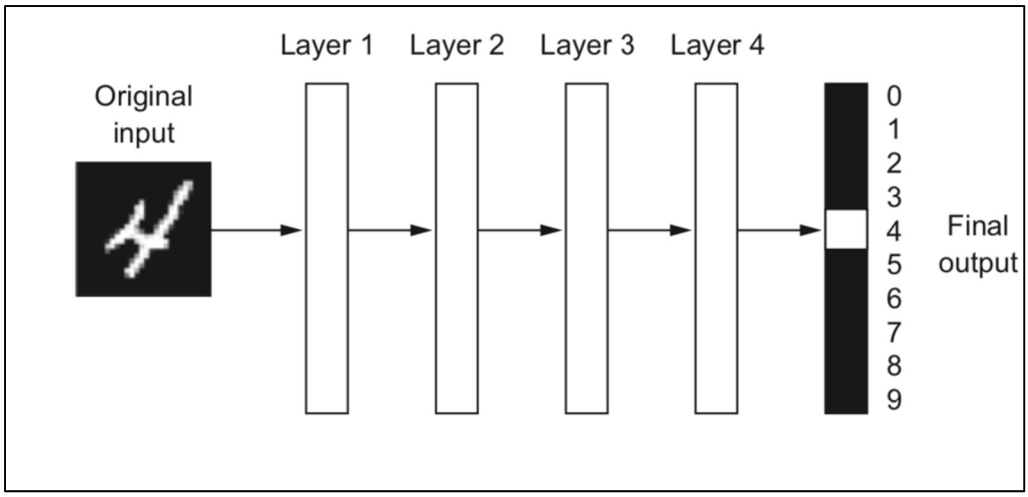

Figure 1.8 Schematic of deep neural network for classification of the digits
Taken from Lambrugh (2020)

How deep learning works?

As described above, machine learning maps inputs to the target (certain digit etc.) by observing many samples of the target (Lambrugh, n.d.). A schematic of a deep neural network is shown in Fig. 1.9. This transformation on the inputs is parametrized by the network's weights. To control how far the output is from the target, comparison between the predicted values and the true value is needed. This task is performed using a cost function ("objective function", (Lambrugh, n.d.)) in the network (Fig. 1.10). The objective function takes the prediction and the true value of the target, and by measuring the distance between the two, determines how good the performance of the network is (loss score). Learning procedure means finding the good weights that enables the network to map the inputs to the targets.

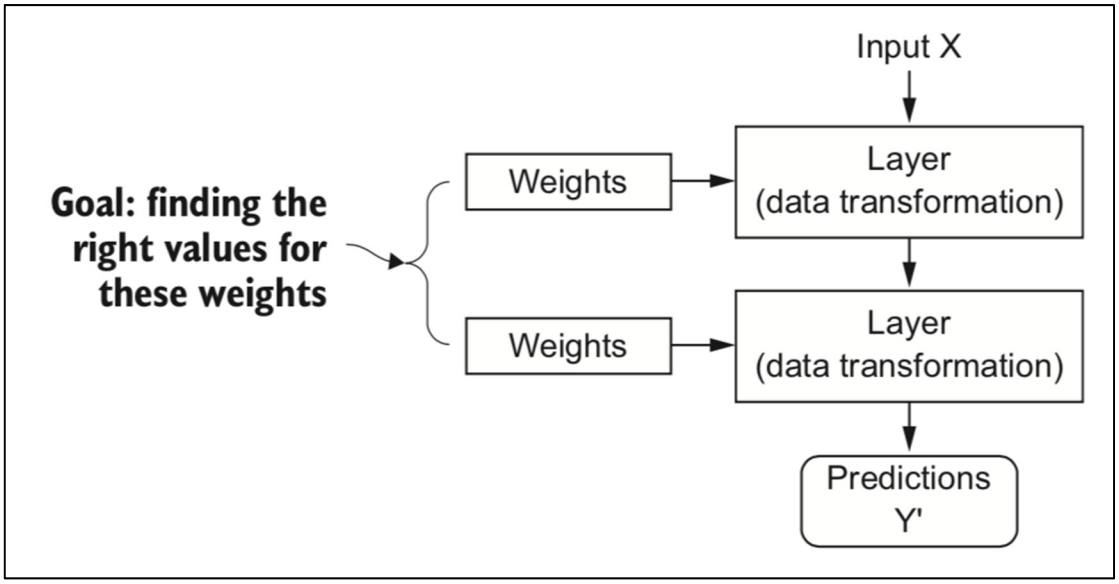

Figure 1.9 Schematic of a neural network, the output, input and the weights
Taken from Lambrugh (2020)

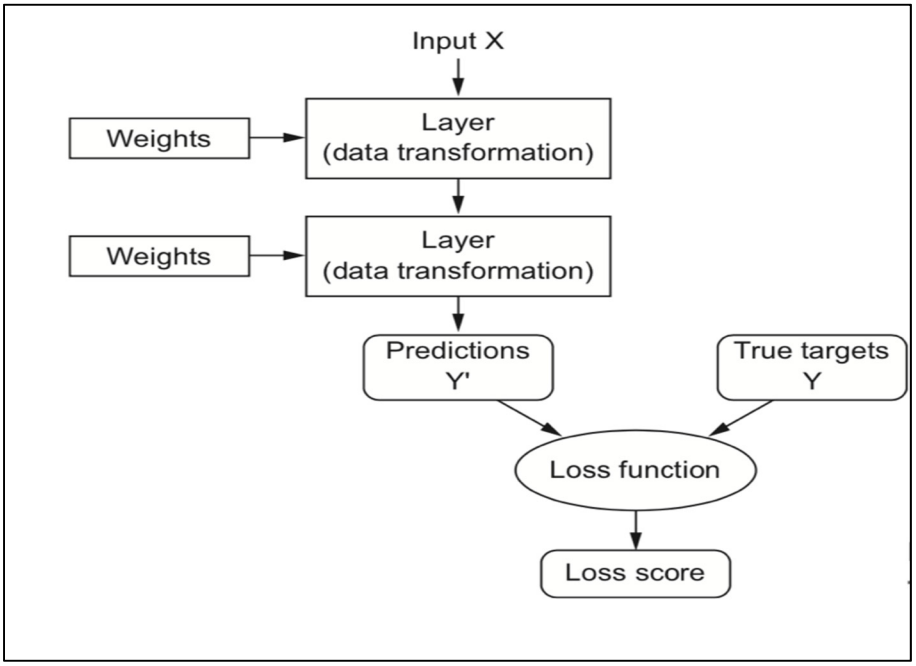

Figure 1.10 The loss function as the measure for scoring the quality of the networks
Taken from Lambrugh (2020)

Learning process

So far, the architecture of the neural network and the relationship between the inputs and the outputs is explained. In neural networks, similar to the curve fitting process, the analogy is to determine the network parameters to minimize a sum-of-squares of errors. Given an input vector $\{x_n\}$, where n corresponds to the number of inputs, and the target vector $\{t_n\}$. The error function can be written as (Bishop, 1994):

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\|y(x_n, w) - t_n\|^2 \tag{1.17}$$

This error function is also called Euclidian loss.

Neural networks tend to overfit. One technique to avoid over-fitting is to penalize the large coefficients which is called regularization by adding a regularization term to the error function (*11.3 Neural Network Models | Forecasting*, n.d., p. 3). The simplest way of regularization has the form of the sum of squares of all the coefficients, which leads to the error function as (Eq. 1.18):

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x,w) - t_n\}^2 + \frac{\lambda}{2}\|w\|^2 \tag{1.18}$$

where $\|w\|^2 \equiv w^T w = w_0^2 + w_1^2 + \cdots + w_M^2$. It is worthwhile to note that w_0 is omitted from the regularizer since having it will affect the choice of origin for the target variable (*Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd Edition.*, n.d.) or, in other words, biases the model forecasting.

Definition of training terms

It is common to divide the data set into the training and the test sets. The training data is used to determine the parameters of the forecasting method and the test set is used to determine how well the model can forecast the new data. The size of the test set is typically 20% of the whole data set (Fig. 1.11).

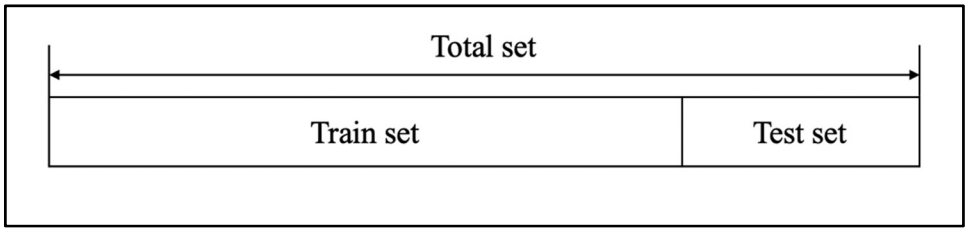

Figure 1.11 Train and test data split

Deep neural networks use the loss score as the feedback signal to adjust the weights. “This adjustment is the job of optimizer, which implements what is called the Backpropagation” (Lambrugh, n.d.). The weights are chosen randomly at the beginning. Then, during the learning

loops, the weights are adjusted in the right direction (opposite of the gradient) to decrease the loss. A small loss indicates that the network produces a minimum distance between the prediction and the actual targets (Lambrugh, n.d.).

1.12 Residual Networks (ResNets), N-Beats

N-Beats is a form of residual network (ResNet) architecture to solve univariate time series forecasting problems (Oreshkin et al., 2020b). Classical ResNets add the input of stack of some layers to the output of the stack before it is fed as an input to the next stack (He et al., 2016). Oreshkin et al. (Oreshkin et al., 2020b), implemented a new ResNet architecture with double residual topology which is capable of forecasting in discrete time. N-Beats structure improved the accuracy of forecasting of the M3 ((Makridakis & Hibon, 2000)), M4((*Mcompetitions/M4-Methods*, n.d.)) and TOURISM data sets by the maximum 11% accuracy over the best forecasting models.

N-Beats implementation

The implementation of N-Beats has outstanding characteristics. It is fast in both inference and training, it is interpretable and, more importantly, it has application in a wide range of target domains (Oreshkin et al., 2020b). The N-Beats architecture is composed of stacks of blocks, each one comprised of fully connected layers. A detailed description of the N-Beats architecture is given below.

Basic blocks

The proposed building block of their architecture (Oreshkin et al., 2020b) is as shown in the left-hand part of the Fig. 1.12. The block has a fork architecture. Each block l has an input x_l and two outputs: the block's forward forecast $\hat{y}_l$ of length H (representing the future horizon), and $\hat{x}_l$, the block's best estimate of x_l (Fig. 1.12).

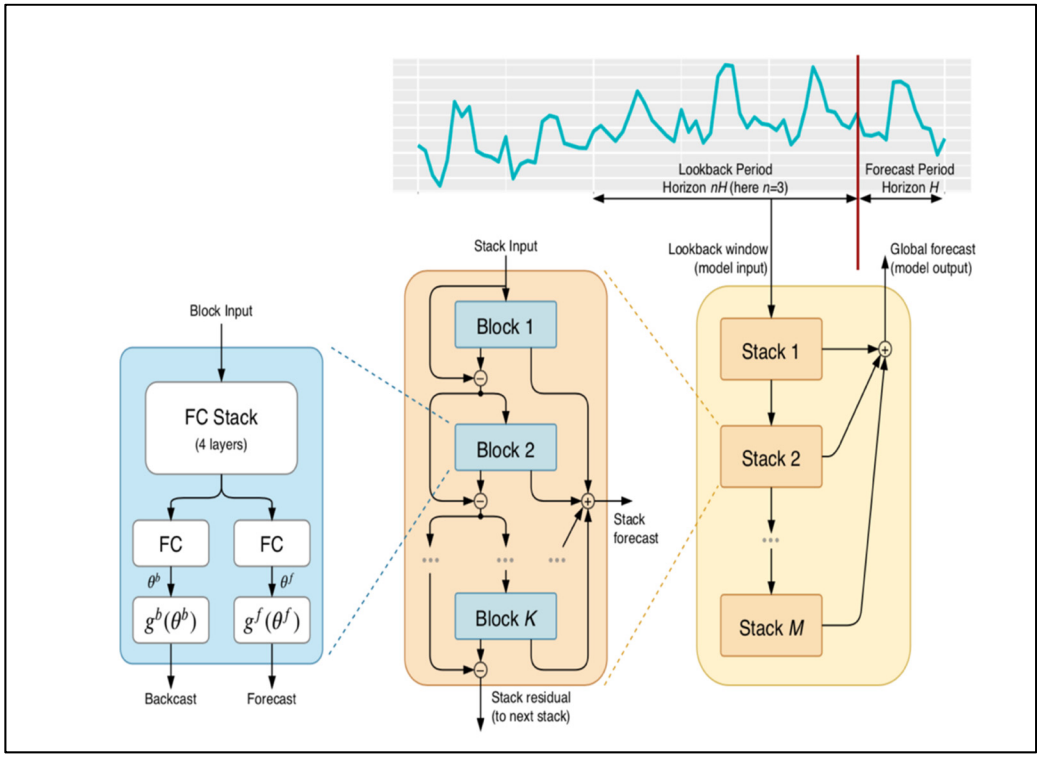

Figure 1.12 N-Beats bock diagram
Taken from Oreshkin (2020)

The basic building block consists of the following two parts: the fully connected network that produces forward theta l (θ_l^f) and the backward theta l (θ_l^b) predictors (the expansion coefficients) (Oreshkin et al., 2020b). The second part is made of two basis layers g_l^f and g_l^b that accept θ_l^f and θ_l^b, respectively as the inputs to produce the forecast $\hat{y}_l$ and the backcast $\hat{x}_l$ (Fig. 1.12 and Eq. (1.19) to Eq. (1.24)). The following set of operation is performed in block l of the N-Beats architecture. The FC layer is fully connected layers. Linear layers are the projection layers i.e., $\theta_l^f = W_l^f h_{l,4}$.

$$h_{l,1} = FC_{l,1}(x_l), \tag{1.19}$$

$$h_{l,2} = FC_{l,2}(h_{l,1}) \tag{1.20}$$

$$h_{l,3} = FC_{l,3}(h_{l,2}) \tag{1.21}$$

$$h_{l,4} = FC_{l,4}(h_{l,3}) \tag{1.22}$$

$$\theta_l^b = \text{linear}_l^b\left(h_{l,4}\right) \tag{1.23}$$

$$\theta_l^f = \text{linear}_l^f\left(h_{l,4}\right) \tag{1.24}$$

For each full layer, they used ReLU activation function ($h_{l,1} = \text{Relu}\,(W_{l,1} + b_{l,1})$). This part of the architecture is providing the expansion coefficient θ_l^b fed to g_l^b to produce the estimation for x_l. The aim is to help the downstream blocks by removing the components of their input that are not critical in the forecasting process (Oreshkin et al., 2020b).

The second part of the block performs mapping the θ_l^f and θ_l^b to the outputs as described in Eq. (1.25) and Eq. (1.26):

$$\hat{y}_l = g_l^f(\theta_l^f), \quad \hat{x}_l = g_l^b(\theta_l^b) \tag{1.25}$$

$$\hat{y}_l = \sum_{i=1}^{\dim(\theta_l^f)} \theta_l^f v_i^f, \quad \hat{x}_l = \sum_{i=1}^{\dim(\theta_l^b)} \theta_l^b v_i^b \tag{1.26}$$

where v_i^f and v_i^b are the forecast and backcast basis vectors (Oreshkin et al., 2020b).

Double residual stacking

Oreshkin et al. (Oreshkin et al., 2020b) used the extension of the ResNet architecture, recommended by Huang et al. (Huang et al., 2016). The principle of this architecture is to introduce another connection from each stack's output to the input of the following stack. This characteristic improves the trainability of the model but at the same time makes it difficult to interpret. The N-Beats architecture proposed by Oreshkin et al. (Oreshkin et al., 2020b), has double residuals (Fig. 1.12, middle and right). The operations in their model can be written as:

$$x_l = x_{l-1} - \hat{x}_{l-1} \tag{1.27}$$

$$\hat{y} = \sum_{i=l} \hat{y}_l \tag{1.28}$$

Removing $\hat{x}_{l-1}$ from the signal makes the forecast easier. It can also facilitate the "fluid gradient backpropagation". Moreover, each block provides a partial $\hat{y}$ that is accumulating first at each stack and then in the whole network and provides the final forecast (Oreshkin et al., 2020b). This architecture is capable of decomposing the time series into trend and seasonal parts.

1.13 Recurrent Neural Networks (RNN), LSTM

Recurrent Neural Networks (RNNs) are sequence models that gained a lot of success in speech recognition (Zaremba et al., 2015) due to have memory in their architecture. The interest in using these networks arises from connecting the previous information to the present task (*Understanding LSTM Networks -- Colah's Blog*, n.d.).

LSTM is a novel traffic forecaster network based on long and short-term memory (*LSTM Network: A Deep Learning Approach for Short-Term Traffic Forecast - IET Journals & Magazine*, n.d.) and it is a type of RNN (Fig. 1.13). The LSTM is different from conventional forecast methods as it considers temporal-spatial correlations within a traffic system using two-dimensional network which is constructed of memory units. The LSTM architecture is appropriate for sequence forecast because it is capable of capturing the training data within time lags (e.g., 10 min lags). In practice, standard RNNs are not able to deal with this gap because the gradient vanishes and explodes (*LSTM Network: A Deep Learning Approach for Short-Term Traffic Forecast - IET Journals & Magazine*, n.d.). Hochreiter et al. (Hochreiter & Schmidhuber, 1997) found the source of this problem and proposed LSTM. Compared to conventional RNNs, LSTM captures the time series features within longer span of time (*LSTM Network: A Deep Learning Approach for Short-Term Traffic Forecast - IET Journals & Magazine*, n.d.). The LSTM model tries to alleviate these back-flow problems. It can learn to bridge time intervals without loss of information in the short-term.

Methodology of LSTM

As mentioned earlier, LSTM has time and space complexity (Zhao et al., 2017). The forecast predicts the next moment and is based on the current state as well as the previous knowledge of the system.

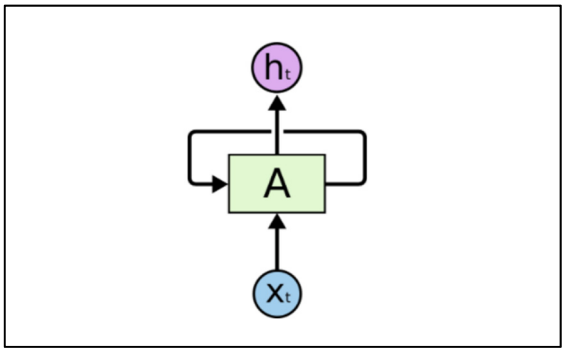

Figure 1.13 Recurrent Neural Network
Taken from ("Understanding LSTM Networks -- colah's blog," n.d.)

In the diagram in Fig.1.13, the network sees the input x_t and provides the output h_t. A loop allows the information to pass to the next step. This new neural network could be seen like several consecutive neural networks. LSTM is a special type of recurrent neural networks (RNN).

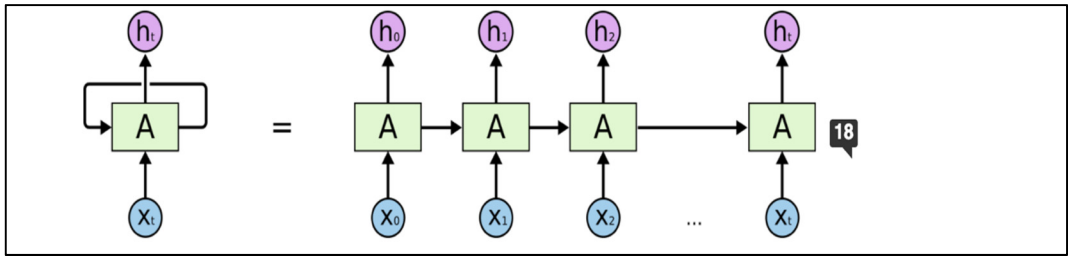

Figure 1.14 Several Neural Networks can make an RNN
Taken from ("Understanding LSTM Networks -- colah's blog," n.d.)

LSTM networks

LSTM networks are designed to avoid long-term dependency problems. RNNs have a repeating chain module. This repeating module has a very simple structure such as tanh layer (*Understanding LSTM Networks -- Colah's Blog*, n.d.).

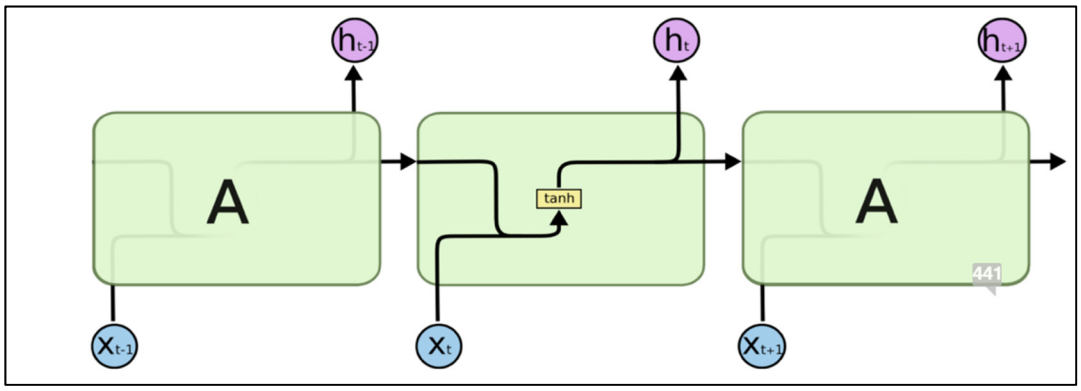

Figure 1.15 RNN repeating modules
Taken from ("Understanding LSTM Networks -- colah's blog," n.d)

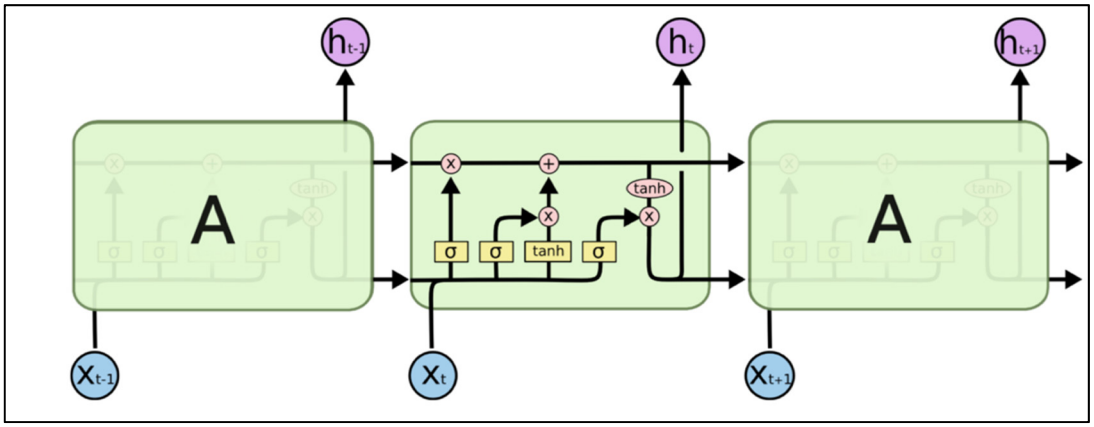

Figure 1.16 The repeating module in LSTM has four interacting layers
Taken from (Understanding LSTM Networks -- colah's blog," n.d)

LSTM has a similar chain structure (Fig. 1.16) but the repeating module is different in LSTM. Instead of having one-layer neural network, LSTM contains four interacting layers in a very special way. Fig. 1.17 shows the notation in the diagram above (Fig. 1.16).

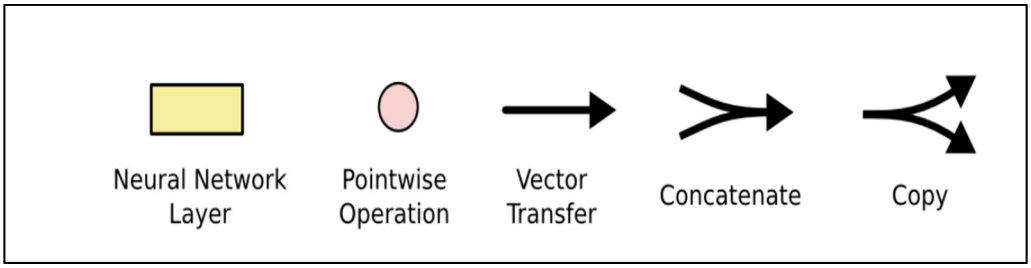

Figure 1.17 The notations in Fig. 1.13
Taken from ("Understanding LSTM Networks -- colah's blog," n.d)

A key characteristic in LSTM is a horizontal line that runs through the top of the diagram. It runs linearly along the change with only some minor interaction along its way. Therefore, the information can move forward without change. However, LSTM can add or drop information. Gates allow the information to go through. They are composed of sigmoid neural net layer and a multiplication operator. The sigmoid output defines how much of the input could go through it. For instance, zero means that no input should pass (Fig. 1.18) (*Understanding LSTM Networks -- Colah's Blog*, n.d.).

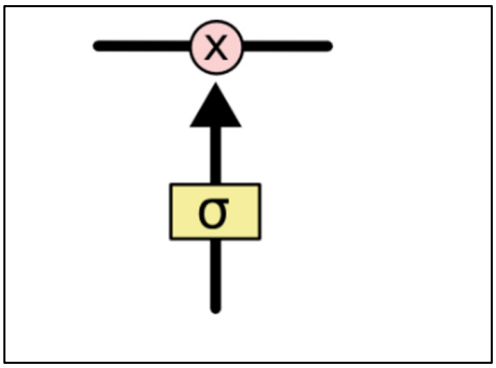

Figure 1.18 Sigmoid neural net layer and a multiplication operator
Taken from ("Understanding LSTM Networks -- colah's blog," n.d)

Steps in LSTM forecasting

The learning steps in LSTM is first to decide which information we would like to throw away and the next step is which information we intend to keep and update. The decision of forgetting

some data is made by the sigmoid layer which is called “forget gate layer”. By looking at $h_{t-1}$and x_t, it can decide for the output to be zero or one for each number in the cell state C_{t-1}. One means completely keep the data and zero means throw the data (Fig. 1.19).

Coming back to our example of predicting the next word based on the learned previous word. The cell state is including the current gender of the subject. Therefore, when a new subject is seen, the cell forgets the old subject.

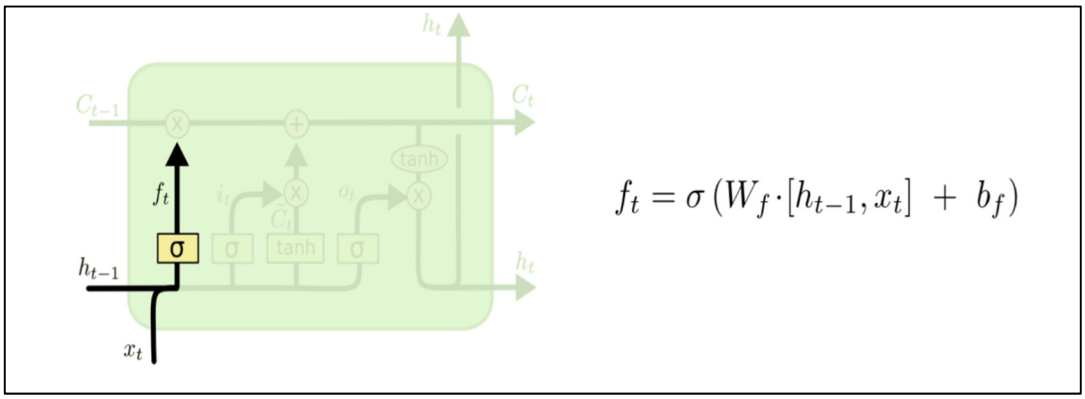

Figure 1.19 LSTM sigmoid layer as the “forget gate layer”
Taken from (“Understanding LSTM Networks -- colah’s blog,” n.d)

Now, the old cell will be updated; C_{t-1} to the new one C_t (Fig. 1.20). We already have the decision of what to do. We multiply the old state by f_t and forget the data we intended to forget. Afterwards we add $i_t * \tilde{C}_t$. This new value is scaled by how much we intend to update each state.

In the language detection model, this is where we decide to drop some information of the old values and add the new information.

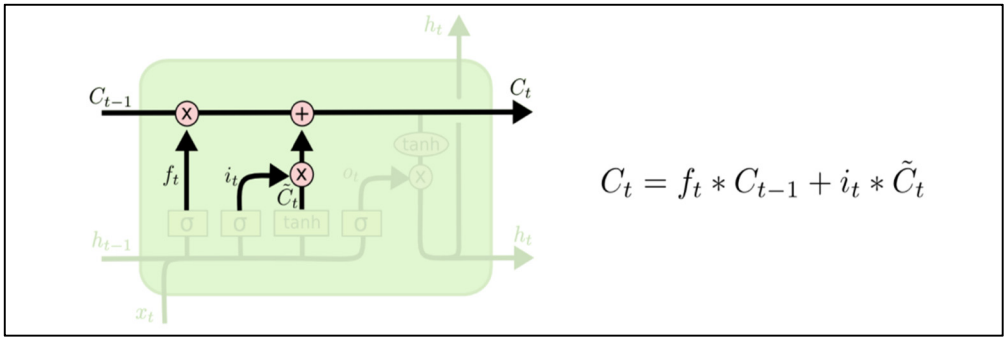

Figure 1.20 The state of forgetting and adding information
Taken from ("Understanding LSTM Networks -- colah's blog," n.d)

The final step is generating an output. The output must be based on the cell state. However, it would be the filtered version of the output. Therefore, firstly, the sigmoid layer decides what parts of the cell state is to be selected for the output. Then the cell state would be put through the tanh, which could push the values between -1 and 1 (Fig. 1.21).

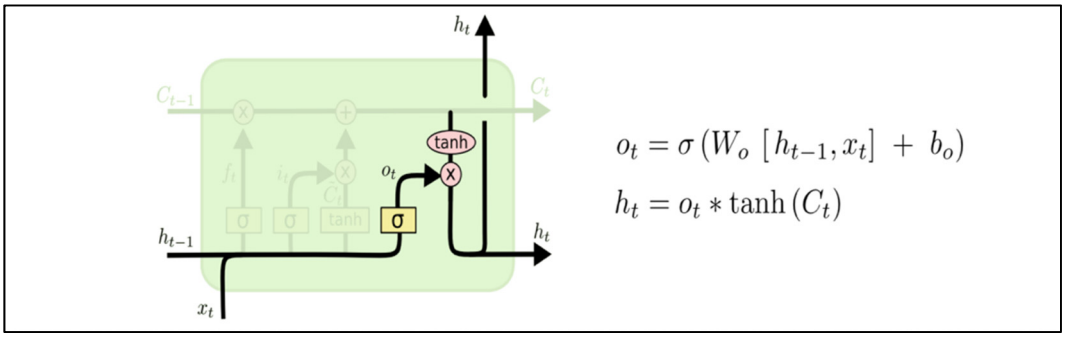

Figure 1.21 The output of LSTM
Taken from ("Understanding LSTM Networks -- colah's blog," n.d)

After reviewing the most common strategies in forecasting time series, the next section deals with the second objective of this thesis, uncertainty.

1.14 Uncertainty

Despite numerous researches in ML and its fast deployment in the measuring systems, there are fewer works on implementing uncertainty in the models. The uncertainty calculation is a necessity to adopt ML to the commercial services and products because prediction is not the only important task but also the accuracy of the prediction (Levi et al., 2020). Designers and

producers are encouraged to report the uncertainty as it gives the user a certain degree of confidence (Al Osman & Shirmohammadi, 2021; Myslín, 2020). Here, the quantification of uncertainty will be surveyed in the literature to address the second objective of this thesis. After having an overview on the viable methods conducted by the researchers in this domain, a method of introducing uncertainty to the deep learning models, based on the sets of criteria such as feasibility of implementation for a large amount of data, will be selected.

Traditional forecasting aims to predict the expected value (mean) of a time series as accurately as possible (Koochali et al., 2019). There is a wide range of mean regression methods in the literature, e.g., ARIMA and machine learning methods including Support Vector Machines (Du et al., 2016; Yan & Chowdhury, 2015), ANNs (Assaad et al., 2008; Ogunmolu et al., 2016). However, in methods that forecast the future with mean regression, the fluctuation around the mean is missing. Therefore, in some cases, the results are not reliable.

Probabilistic forecasting quantifies the variances in the prediction (Gneiting & Katzfuss, 2014) with different approaches (Collins, 2007; Gneiting & Raftery, 2005; Palmer, 2002). Stigler (Stigler, 1975) denoted for the first time the transformation from the point estimation to distribution estimation (Koochali et al., 2019). The two most markedly approaches in distribution estimation are conditional Quantile regression and conditional expectile regression (Koochali et al., 2019).

Quantile regression is a statistical method to estimate and perform inference about conditional quantile functions (Koenker, 2017). Expectile regression is similar to the quantile regression but is based on asymmetric piecewise quadratic functions (*Efron, B. (1991). Regression Percentiles Using Asymmetric Squared Error Loss. Vol.1, No.1.*, n.d.). Forecasting a collection of points instead of one point for a particular quantity can be performed using ensemble learning (Gneiting & Katzfuss, 2014).

Statistical post-processing methods such as non-homogeneous regression (NR) and ensemble model output statistics (EMOS) have been proposed by Gneiting et al. (Gneiting & Raftery,

2005). Moreover, ensemble Bayesian averaging approach (BMA) is another post-processing statistical method (Gneiting & Raftery, 2005).

Bayesian probability theory has been employed to provide probabilistic forecasting. This method offers mathematical tool to model uncertainty, however, the drawback is its complex computation and high cost (Gal & Ghahramani, 2016b). Several approaches have been developed to solve the complexity of the Bayesian methods by proposing approximations such as Markov chain Monte-Carlo (MCMC) methods, variational approximations, sequential Monte-Carlo and expectation-propagation (Doucet et al., 2001; Jordan et al., 1999). Still, this method suffers from high computational cost hinders its use in practical applications (Gal & Ghahramani, 2016b). Gal et al. (Gal & Ghahramani, 2016b) used dropout method as an approximation for the Bayesian theory. Dropout in machine learning is used to prohibit over-fitting. They showed that applying dropout before every weight-layer in a network with any arbitrary depth and non-linearity is mathematically equivalent to the probabilistic Gaussian process. Dropout is normally used in NN as a regularization tool. Applying dropout in the inference time produces different outputs for the same input because, in each run, the input is passing through a slightly different network as a result of some dropped units (Fig. 2.22).

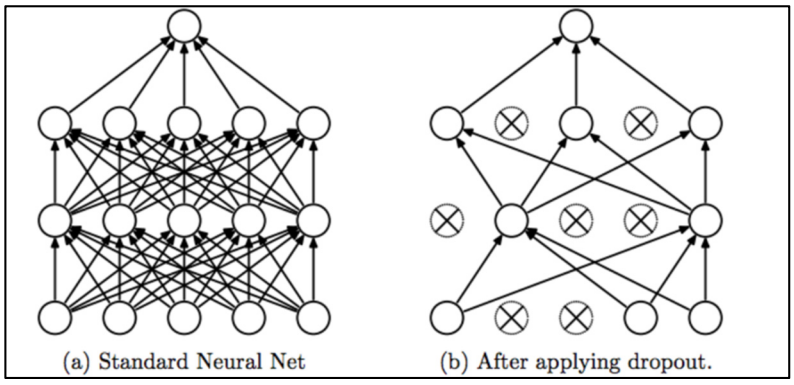

Figure 1.22 The NN without and with dropout
Taken from Budhiraja (Budhiraja, 2018)

Monte-Carlo dropout is not the only common method for regularization. There are other methods namely Batch Normalization for regularization (Teye et al., 2018). Ioffe et al. (Ioffe & Szegedy, 2015) used this method to introduce randomness to the model. Teye et al. (Teye

et al., 2018) showed that the Batch Normalization, similar to dropout could be used as an approximation to the Bayesian model. This method can lead to variational output for the ML models and as a result can enable us to calculate uncertainty estimation.

Girard et al. (Girard et al., 2003) considered multi-step ahead prediction with time-series. They used non-pragmatic Gaussian process model. The method of their prediction was k-step ahead forecasting of discrete-time of a non-linear system by repeating the prediction of one-step ahead. A prediction of y at time t+k is based on the previous point estimations. They used a Gaussian approximation to compute the uncertainty about the intermediate values. They showed that their propagation approach is comparable with the Monte-Carlo simulations and concluded that both approaches can achieve more realistic error bars than Naïve approach. Because Naïve ignores the uncertainty on the current state.

Koochali et al. (Koochali et al., 2019) developed a model named forecasting Generative Adversarial Network (ForGAN) based on LSTM and applied probabilistic approach. GAN is a network which lets us learn unknown probability distribution from samples of the distribution (Koochali et al., 2019). They implemented their probabilistic model based on the LSTM algorithm.

Brando et al. (Brando et al., 2019) applied probabilistic treatment including Bayesian approach in Deep Learning (Bishop, 1994; Blundell et al., 2015). Their method is a formulation of Deep Regression Networks that output the distributions of the estimated targets and an input-dependent variance of the target estimate. They implied that, depending on the source of the noise coming from the model parameters or directly from the output, the type of uncertainty is different. They fit such a network by maximum likelihood estimation and considered two methods: (i) adding Dropout (Gal & Ghahramani, 2016b) (0.5 probability parameter, Densedrop) and (ii) Bayes-By-Backprop variation (Blundell et al., 2015), which is denoted as Dense BBB. Given a new input, they obtained a set of output samples, rather than receiving one output. The mean of these values was considered as the final estimation of the target, and the standard deviation served as the uncertainty score. It is also worth mentioning their

baselines to validate their approach. They first evaluated three simple predictors: the mean of the series, Zero ($\hat{y} = 0$) and the last seen value (previous month). In all three cases, they used the variance of the input as the uncertainty score var(z). They also implemented Random Forest, General Additive Model (GAM), and some Deep Learning models. GAM is a generalized linear model which has linear predictor and is the summation of the smooth functions of the covariates (Brando et al., 2019; Hastie & Tibshirani, 1987) and its related uncertainty score is denoted as SE. They applied their recommended uncertainty calculation methods on LSTM.

DeVries et al. (DeVries & Taylor, 2018) applied out-of-distribution method for considering uncertainty in classification. They added a confidence branch for each class to vote from 0 to 1 (sigmoid function). The more confident the model about an output, the more its confident score is close to 1.

Salinas et al. (Salinas et al., 2020) proposed a Deep AR method that resulted in accurate probabilistic forecast. Their RNN architecture incorporates a negative Binomial likelihood. Their models have four key characteristics: (i) It can learn the seasonal behavior and, with the learned dependencies on the given covariates of the time series, a minimum amount of feature engineering is required. (ii) Their Deep AR model provides probabilistic forecast based on Monte-Carlo samples that can be used for computation of the quantile estimates for different horizons. (iii) Their method is capable of learning from similar items and predict even when there is no history for the unseen data. (iv) Although they do not consider Gaussian noise, they incorporate wide sets of likelihood functions to allow the user to select the proper approach for the statistical properties of the data. Additionally, they considered Gaussian likelihood for the real-value data and negative-binomial likelihood for positive count data. They parametrized the Gaussian likelihood using the mean and standard deviation and the negative binomial distribution by its mean and the shape parameter (Snyder et al., 2012).

Zhang et al. (Zhang & Patras, 2018) proposed a deep learning architecture named spatio-temporal neural network (STN) for precise mobile forecasting. They showed that their Double

STN (D-STN) architecture can make 10-hours ahead prediction with high accuracy. They implied that the uncertainty may grow over-time. By incorporating the prior knowledge of averages into the model, they decreased the uncertainty and improved the performance of forecasting.

According to the literature, there are also some methods to calculate the calibration error for classification (Expected Calibration Error (ECE) (Naeini et al., 2015)), and another calibration method for the regression problems developed by Kuleshov et al. (Kuleshov et al., 2018) which uses the intuition from the classification calibration error.

Boot-strap and ensemble learning has been used as the baseline with three different strategies: with different random seeds (similar with what we have already implemented), different models (diverse; randomly initialized Nets), adding diversity as a hyper-parameter that yields better cross-entropy and better accuracy (*NeurIPS 2020 : (Track2) Practical Uncertainty Estimation and Out-of-Distribution Robustness in Deep Learning*, n.d.).

1.14.1 Selected means of uncertainty based on literature survey

To summarize, the main methods for implementing uncertainty in our algorithms are listed in table 1.1. Accordingly, the dropout method used by Gal et al. (Gal & Ghahramani, 2016b), which is a recognized as an appropriate approximation for the Bayesian probabilistic approach could be selected as the method to apply uncertainty to our neural network algorithms (feedforward and ResNets). Moreover, to assess the quality of uncertainty prediction, baselines such as quantile regression or ensemble learning, are feasible methods to be implemented.

Table 1.1 Uncertainty methods

Ensemble Learning (Baseline)	**Likelihood**	**Dropout**	**Bayesian Approach**	**Quantile Regression**	**Monte-Carlo Approximation**
Brando (2019)	Brando (2019)	Gal and Gharamani (2021)	Brahim (2004)	Koenker (2017)	Jordan (1999) Doucet (2001) Salinas (2020)

CHAPTER 2

SHORT-TERM LIGHTPATH QoT FORECASTING

In this chapter, we explain the methodology that was used to implement and evaluate the forecasters. Three algorithms (models) were investigated: N-Beats, MLP and LSTM. We implemented the N-Beats model from scratch using the description of the model developed by Oreshkin et al. (Oreshkin et al., 2020a). We used a naïve method as our baseline to compare the performance of the forecasters. Naïve method returns the last seen value in the time series as future time-series value, as described in Chapter 1.

Before training the models, data preparation is essential. Special data preprocessing for time series is explained in section 2.2. After explaining data preparation and preprocessing, hyperparameter optimization, which is key step in training will be described. In the end of this chapter, the measured metrics of the test sets, as the result of training each model will be presented.

2.1 Methodology : Algorithms

The methodology of implementing N-Beats, MLP and LSTM models is described in sections 2.1.1 to 2.1.3.

2.1.1 Implementation of N-Beats

Using the structure of N-Beats presented in Chapter 1 (Fig. 1.12) (Oreshkin et al., 2020a), we found the optimum implementation in terms of the number of blocks and stacks by evaluating the forecasting results. One version of our N-Beats implementation code is mentioned in Appendix I.

N-Beats forecaster is based on the residual deep neural network (DNN) shown in Fig. 1.12. These networks are called “residual” because at each stage of the network some output of the stage is subtracted from the input to the stage and the difference (the “residual”) is forwarded to the next. Our N-Beats model is composed of three stacks. Each stack contains four blocks. Each block includes three dense layers and two theta layers each corresponds to the forecast and backcast followed by the forecast and backcast layers, respectively, and the residual layer at the end (Fig. 1.12). A schematic of the stacks and blocks of our N-Beats model is presented in Fig. 2.1. In our N-Beats architecture, all the dense and theta layers contain 64 neurons, the forecast layers contain 1 neuron and the backcast layers also have 64 neurons, which corresponds to 3,468 neurons in total. The reason for selecting this structure is based on the experiments on the performance of the model. Since the N-Beats model is hard coded, we find the best number of neurons with conducting experiments with the data.

We implement this model using TensorFlow without any dropout layers for regularization during the training process.

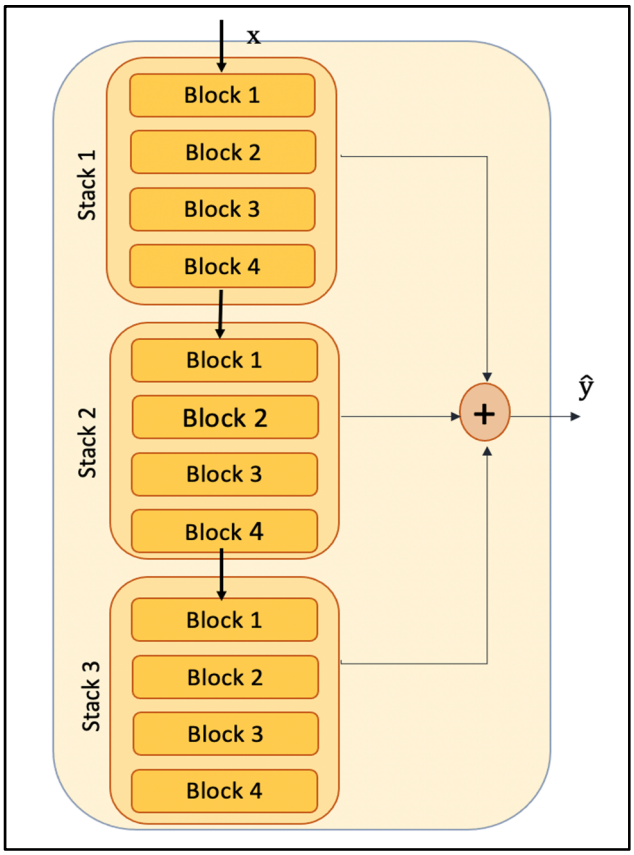

Figure 2.1 Schematic of our N-Beats implementation

2.1.2 MLP model

The MLP model contains two dense layers of 16 neurons, with ReLU activation function, and a single neuron in the final layer with the linear activation function, which corresponds to 33 neurons in total. No dropout layer was employed for regularization during training. The reason of selecting this structure for our MLP model was based on the experiments on the performance of the model (more explanations in the hyperparameter optimization section).

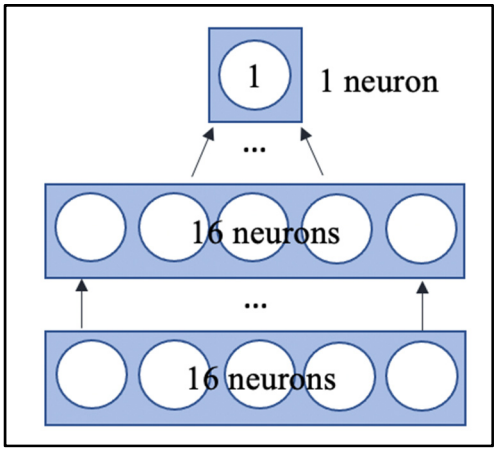

Figure 2.2 Schematic of the MLP model

2.1.3 LSTM model

We implement a LSTM algorithm with three LSTM layers of 256, 128, 64 and 32 neurons and an output dense layer of 1 neuron, respectively. The numbers of neurons are selected based on the evaluation of the results and kept the same for most of the experiments and adopted according to the hyper parameter tuning for other experiments. The numbers of neurons in the LSTM layers are selected based on the hyper parameter tuning.

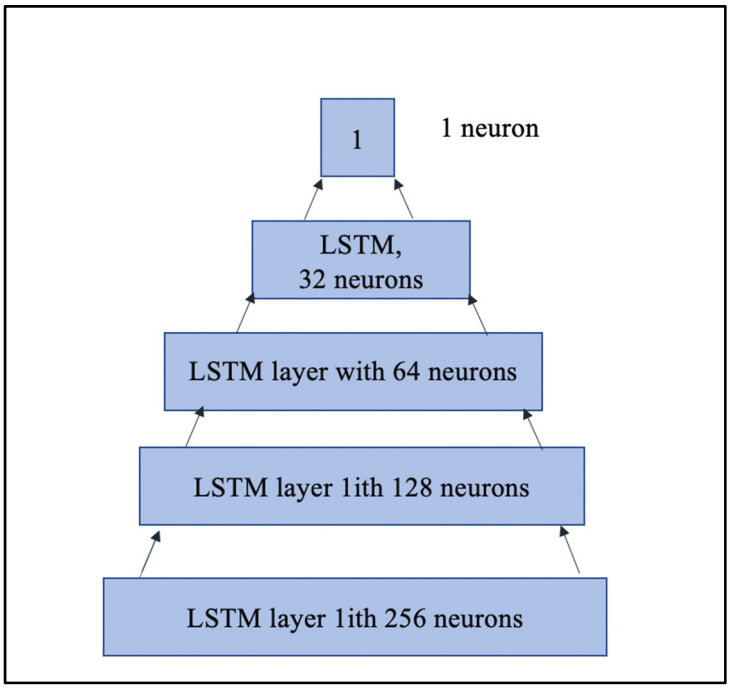

Figure 2.3 Schematic of the LSTM model

2.2 Data sets

In this project, the forecasters have been built using performance metrics (PMs) collected in two production networks in North America, Microsoft and NASP. The Microsoft data set includes 14 months of PMs from their backbone network in North America. The PMs were collected at 15-minute intervals for 4000 optical channels ("Optical Data,"). The target lightpath performance metric in this data set is the estimated Q-factor. Each channel contains nearly 32,000 Q-factor data samples. The evolution of the Q-factor for one Microsoft WDM channel over a period of four days is shown in Fig. 1.1. The NASP data set contains PMs for 148 channels. Each channel includes 12,000 SNR data have been collected in 15 minutes intervals over one year. Fig.1.2 in Chapter 1 represents the evolution of SNR for one WDM channel of the NASP data set over a period of four days. Comparing Fig. 1.1 and Fig. 1.2, the WDM channel of Microsoft Q-factor data, in the same scale of time, is more stationary, while NASP WDM channel exhibits more dynamicity (or variations over time).

2.3 Data preparation

Training the models needs data preparation. Cleaning the data is essential for any type of data to train ML models. The data should be cleaned and missing data have to be replaced by the appropriate values. Another key step for preparing time series data is windowing the data. This step will be explained in section 2.3.1.

2.3.1 Data windowing

To work with the time series, data windowing is essential to generate a sequence of data as the history and define the target within desired time interval right after the history sequence. We select the forecasting setting to be windows of 192 instances (2 days) as the history, predicting a target in 1-h, 2-h to 16-h in the future (Fig. 2.4). Prior to select this setting, we tried different other settings and found out shorter history time than 2 days have not resulted in good forecasting results.

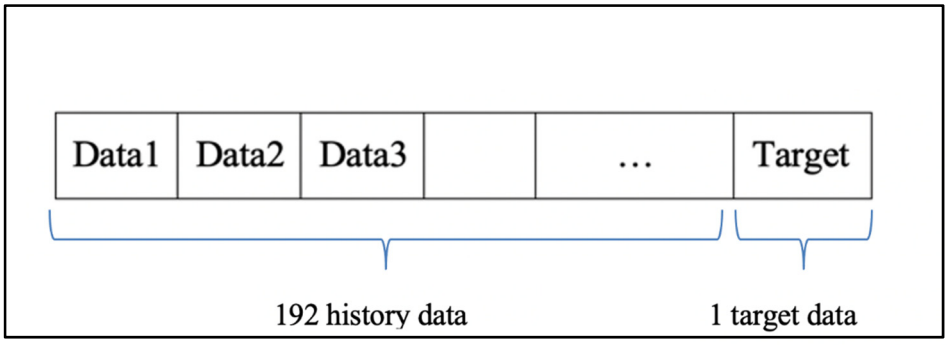

Figure 2.4 History and target data, history 192 SNR (Q-factor) data, target 1 data within desirable time interval

2.3.2 Train, validation, and test sets preparation

The input data set is generated using random sampling (will be discussed in the next section) over 500 channels to produce 14,648 windows of data (samples) with the following data split: training set of 10,253 samples, validation set of 2,198 samples and testing set of 2,199 samples,

which correspond to 70% training, 15% validation and 15% test sets from the generated windows of data, respectively (Fig. 2.5). According to our survey over all the 4000 channels of this data set, all the channels show similar behaviors with the difference in the Q-factor level. By sampling over 500 channels, we ensured sampling from different Q-factor levels (more detailed information is presented in section 2.7).

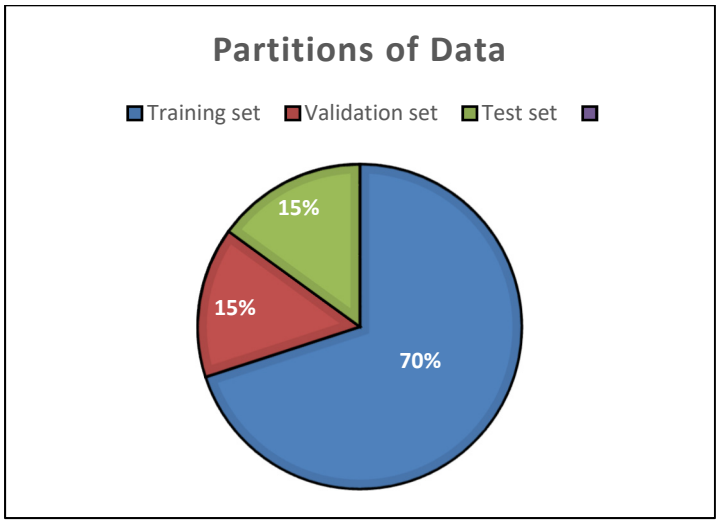

Figure 2.5 Pie chart, knowledge base split ratio; training set 70%, validation set 15% and test set 15%

We sampled 140 channels of SNR data from NASP data set with the same strategy as we sampled the Microsoft data set. Like the Microsoft data division, we used 70%, 15% and 15% of data corresponding to the train, validation and the test sets (Fig. 2.5), and each set contains 3,300, 910 and 910 data windows respectively. We generated the train, validation, and the test sets separately for both data sets, so that none of the instances from validation and test sets are seen during training.

2.3.3 Random sampling

To prevent overfitting for the two data sets used in our experiments, we randomly sample the data. Since in the two time-series (Microsoft and NASP data sets), the range of data

is narrow and the data is monotone, if we randomly select the windows of data, we can prevent over-fitting by adding diversity to the data. To do that, we randomly select a sequence (192 data in our case which corresponds to two days of history) of Q-factor or SNR. The output value is the next value in 1-h, 2-h to 16-h horizons right after the sequence. Once we sampled all the data set from Microsoft data set, for instance, we shuffle the data while training to ensure the data is randomly set. The same procedure was applied for the NASP data set.

It is worth noting that we consider the same window size for all the algorithms as our goal is assessing the performance of the models for the next step with the same settings, which is the implementation of the uncertainty on the selected model, with the same settings.

2.3.4 Data preprocessing

For time series, there are more steps to prepare the data. They need special care because it is necessary to have stationary data without seasonality or long-term trends using common methods such as differencing (Yadav et al., 2020). Plotting is a good way to evaluate if the time series has the seasonality. By plotting various channels and comparing them in the same period of time from both data sets, we didn't recognize any significant seasonality pattern, including daily, weekly and monthly in each of the data sets. Therefore, our data preprocessing was limited to filling the missing data with the expected appropriate data. For both data sets, we chose the mean value of each of the time series data files and replaced missing values by the means. Moreover, for some data sets, using the Min-Max Scaler from Scikit-Learn library might be helpful. Specially if the data are not in the same range. Since both data sets that we chose are time-series and have limited range, we did not use the Min-Max scaler for the most parts of experiments; namely, we did not use this scaler for N-Beats and MLP algorithms because the results were not impacted by this preprocessing. However, we used the Min-Max scaler method to scale the data in training and validation sets for the LSTM algorithm and we performed the inverse scaling at the test time to ensure LSTM is well set up.

2.4 Set of metrics

In this section, we briefly mention the set of metrics that are used to evaluate the results of our simulations as well as the loss function (will be discussed in the next section). In the following metrics, N is the number of data points, y_i is the true value and $\hat{y}_i$ is the prediction.

2.4.1 Mean Absolute Error (MAE)

This metric is the absolute average of the difference between each prediction and its actual value. Because of the absolute, the signs of the residuals are not considered. If the absolute is removed, it becomes Mean Bias Error (MBE), which is applicable to measure the model bias. MAE is defined with Eq. (2.1).

$$\mathrm{MAE} = = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.1}$$

MAE is a metric that takes the same unit as the variable (here dB).

2.4.2 Mean Absolute Percentage Error (MAPE)

The Mean Absolute Percentage Error is a very popular metric to evaluate the performance of the forecasters and is defined by Eq. (2.2):

$$\mathrm{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{y_I} \tag{2.2}$$

There are two shortcomings contributed with the MAPE. The first one is that MAPE is asymmetric and considers more penalty for the negative errors (Lewinson, 2020). The second one is that when the actual value is zero, MAPE becomes indefinite. MAPE is the percentage error.

2.4.3 Symmetric MAPE

The metric symmetric MAPE (sMAPE) is like MAPE and supposed to combat the shortcoming of MAPE. This is a percentage error measurement and has the upper and lower bounds of 0% and 200%, respectively and is formulated by Eq. (2.3).

$$\text{sMAPE} = \frac{1}{\text{N}}\sum_{\text{i=1}}^{\text{N}}\frac{|\text{y}_\text{i} - \hat{\text{y}}_\text{i}|}{(\text{y}_\text{i} + \hat{\text{y}}_\text{i})/2} \tag{2.3}$$

sMAPE is a percentage error.

2.4.4 Root Mean Square Error (RMSE)

Root mean square error (or deviation) is one of the most common metrics to evaluate the predictions. This metric demonstrates the Euclidian distance from the true values.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}\|\text{y}_\text{i} - \hat{\text{y}}_\text{i}\|^2}{N}} \tag{2.4}$$

RMSE takes the same units of the predicted variable (here dB).

2.5 Hyperparameter optimization

In neural networks, the training is being performed with gradient decent, which is the loss function. The choice of the loss function could be diverse; to name of a few: mean absolute error (MAE), mean squared error (MSE), etc. Gradient decent is giving an estimate of the errors so that the weights could be updated (Brownlee, 2019a). This process is called hyperparameter optimization (or tuning). It involves choosing the weights to minimize the discrepancy between the true values and the prediction. The process of hyperparameter tuning

is illustrated in Fig. 2.6. We start training the model with a set of hyperparameters. After running the model, the error between the prediction and the actual value would be evaluated.

Hyperparameters could be selected from the model (e.g., number of hidden layers, activation function, etc.) or from the learning process (e.g. learning rate, batch size, learning rate, etc.). Some of the optimizers have more parameters (e.g., Adam optimizer has alpha, beta 1, beta 2 and epsilon). Our choice of activation function is ReLU and linear. The optimizer is RMSprop, and number of hidden layers has been selected based on experiments to ensure not having over-fitting or under fitting while training. The number of hidden layers in the MLP model, as well as the number of LSTM layers, are mentioned in the implementation part of each model (previous section). Moreover, we find the optimum structure for N-Beats and perform all the experiments with the same structure. Then we keep those hyperparameters constant and conducted hyperparameter tuning by changing the batch size, number of epochs and learning rate. Among all the models, we only perform tuning for the number of LSTM layers because LSTM is very sensitive to the choice of hyperparameters. Selecting hyperparameters with precision for LSTM model is also mentioned in the previous studies (Yadav et al., 2020).

2.5.1 Batch size

Batch size defines the number of training examples used to estimate the error gradient. Batch size has a significant impact on the accuracy of the error gradients. Moreover, it can control the speed of the learning process. The range of batch sizes for all the models is presented in table 2.1.

2.5.2 Learning rate

Learning rate is defining the amount of change during each step of the error gradient. It could be defined as the step size and is probably one of the most important hyperparameter to tune the neural network (Brownlee, 2019b). Table 2.1 shows the range of learning rates for all the models.

2.5.3 Epochs

Epochs is another hyperparameter in the gradient decent process in estimating error gradient. It is defined as the number of complete passes through the data set while training (Brownlee, 2018). The range of epochs for all our models is shown in table 2.1.

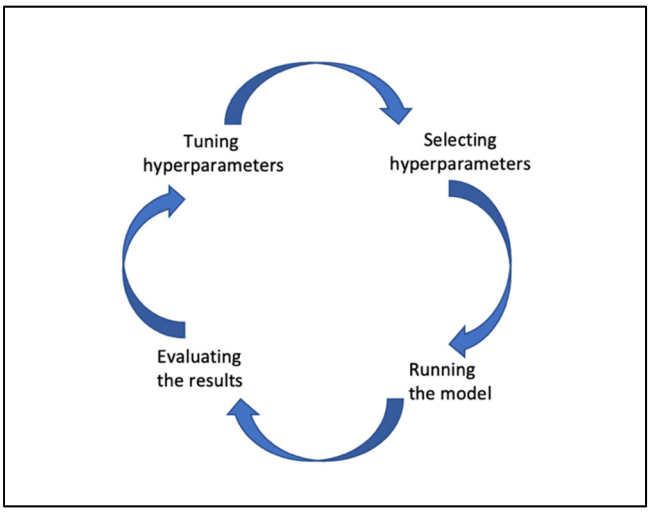

Figure 2.6 The cycle of hyperparameter tuning, starting with selecting a set of hyperparameters, then running the models, after evaluation the results, tuning the hyperparameters

The sets of hyper-parameters for N-Beats algorithm for each time interval, for the two data sets (Microsoft and NASP data sets), are presented in Tables 2.2 and 2.3, respectively.

Table 2.1 Range of hyperparameters for all the models for all the horizons

Model	Learning rate	Epochs	Batch size
N-BEATS	10^{-6} to 10^{-4}	50, 70, 90, 120	128, 256, 512, 1024
MLP	10^{-6} to 10^{-4}	30, 50, 70, 90, 120	128, 256, 512, 1024
LSTM	$10^{-4}, 5 \times 10^{-4}$	50, 70 ,90	256, 512

Table 2.2 N-Beats: optimum hyper-parameter set for Microsoft data set

N-Beats	**1-hour**	**2-hour**	**4-hour**	***8-hour***	***16-hour***
Learning Rate	5×10^{-4}	1×10^{-6}	5×10^{-5}	5×10^{-5}	1×10^{-6}
Batch Size	1024	128	512	128	128
Number of epochs	120	90	120	70	90

Table 2.3 N-Beats: optimum hyper-parameter set for NASP data set

N-BEATS	**1-hour**	**2-hour**	**4-hour**	**8-hour**	**16-hour**
Learning Rate	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}
Batch Size	128	512	512	128	128
Number of epochs	120	90	70	120	90

The sets of hyper-parameters for MLP algorithm for each time interval for the two data sets (Microsoft and NASP data sets) are presented in Tables 2.4 and 2.5, respectively.

Table 2.4 MLP: optimum hyper-parameter set for Microsoft data set

MLP	**1-hour**	**2-hour**	**4-hour**	**8-hour**	**16-hour**
Learning Rate	1×10^{-6}	1×10^{-6}	1×10^{-6}	5×10^{-6}	5×10^{-6}
Batch Size	128	512	512	256	1024
Number of epochs	90	120	120	90	30

Table 2.5 MLP: optimum hyper-parameter set for NASP data set

MLP	1-hour	2-hour	4-hour	8-hour	16-hour
Learning Rate	5×10^{-6}	5×10^{-6}	1×10^{-6}	5×10^{-6}	5×10^{-6}
Batch Size	512	512	256	512	256
Number of epochs	120	90	120	70	70

The sets of hyper-parameters for LSTM algorithm for each time interval, for Microsoft data set are presented in Tables 2.6. Since the LSTM model was tuned at learning rate 10^{-4}, batch size 256, and epochs 50, these settings were maintained for training the model with NASP data set.

Table 2.6 LSTM: optimum hyper-parameter set for Microsoft data set

LSTM	1-hour	2-hour	4-hour	8-hour	16-hour
Learning Rate	5×10^{-4}	5×10^{-4}	5×10^{-4}	5×10^{-4}	5×10^{-4}
Batch Size	256	256	256	256	512
Number of epochs	90	90	50	50	50

2.6 Results and discussion

We train the models with the training data with the ranges of hyperparameters in table 2.1 and report the best hyperparameters in tables 2.2 to 2.5. The size of the training sets in Microsoft and NASP data sets are approximately (for all the forecasting horizons) 10,200 and 3,300 windows, respectively. The loss function while training and validation is mean absolute error (MAE). At the end, we select sMAPE, RMSE and MAE as the metrics to evaluate the performances of the models on the test sets, where MAE is minimum. The size of the

Microsoft and NASP test sets are approximately (for all the forecasting horizons) 2190 and 900 windows respectively (section 2.3.2).

We show the sMAPE (percentage error) metric results of the models from the Microsoft data set in Fig. 2.7 versus horizons of 1-h, 2-h to 16-h. In this figure we can see that the sMAPE of all the models is generally increasing until 4-h horizon, with increasing the time intervals. LSTM shows the maximum sMAPE as compared to all the other models for all the horizons. Its sMAPE hits the maximum of nearly 11% of the prediction for Microsoft data set in 4-h and drops for the next two horizons its increase and decrease is more visible due to the magnitude (Fig. 2.7 and table 2.7).

Naïve shows the least sMAPE among the models and linear growth for all the horizons (table 2.7 and Fig. 2.7). The sMAPE of MLP is very close to that of Naïve and it demonstrates increasing trend until four hours but a decrease in 8-h and a slight rise in 16-h. The trends might not be visible as of LSTM due to its small magnitude in Fig. 2.7 but they could be seen easier in table 2.7.

N-Beats shows in average 0.5% sMAPE error of prediction of SNR in Microsoft data set. It has a considerable rise in the1-h time interval, and from 2-h (minimum error), it shows a slight increase until 16-h (table 2.7 and Fig. 2.7).

In Fig. 2.8, we illustrate the RMSE (dB) metric (has been defined earlier) results of all the algorithms for the Microsoft data set over all the horizons (1-h, 2-h to 16-h). The trend of the performance of the models are the same as our sMAPE metric results (as it was expected). Naïve is showing the minimum RMSE error, and except for a rise at 1-h interval, it shows a linear increase from 0.027 dB in 2-h interval to 0.045 dB in 16-h interval (table 2.8). There is sharp increase in RMSE of 1-hour horizon for the N-Beats model, like its sMAPE (%) of 1-hour horizon for the Microsoft data set (the same as Naïve) (table 2.8). That shows that the chances of having slightly more error in prediction in 1 hour is sometimes higher than the other horizons. Moreover, we see an increasing trend of the error for the 2-hour horizon. The

maximum error value with regards to the RMSE results belongs to the LSTM with the maximum of approximately 2 dB at 4-hour horizon and it has a better performance at 16-hour horizon (Fig. 2.8 and table 2.8).

Fig. 2.9 shows the MAE (dB) metric (has been defined earlier) on the test set of all the algorithms for the Microsoft data set over all horizons (1-h, 2-h to 16-h). The general trend of increasing the error alongside with longer horizons is also observed in this metric. Again, the minimum value is that of Naïve with a linear increase from approximately 0.011 to 0.030 dB (Fig. 2.9 and table 2.9). The MAE of MLP is competing with Naïve (being between 0.027 to 0.032 dB). Another burst move in 1-hour is observed in the MAE value of N-Beats (the same as what we saw in the sMAPE of N-Beats) (Fig. 2.9 and table 2.9). Apart from a spike at 1-h horizon, its error shows between approximately 0.08 to 0.09 dB, with a general increasing trend. LSTM shows the maximum MAE of 1.4 dB at 4-h horizon (Fig. 2.9 and table 2.9) and similar to its sMAPE and RMSE, it shows downward trend with reaching its minimum of 0.43 dB at 16-h horizon.

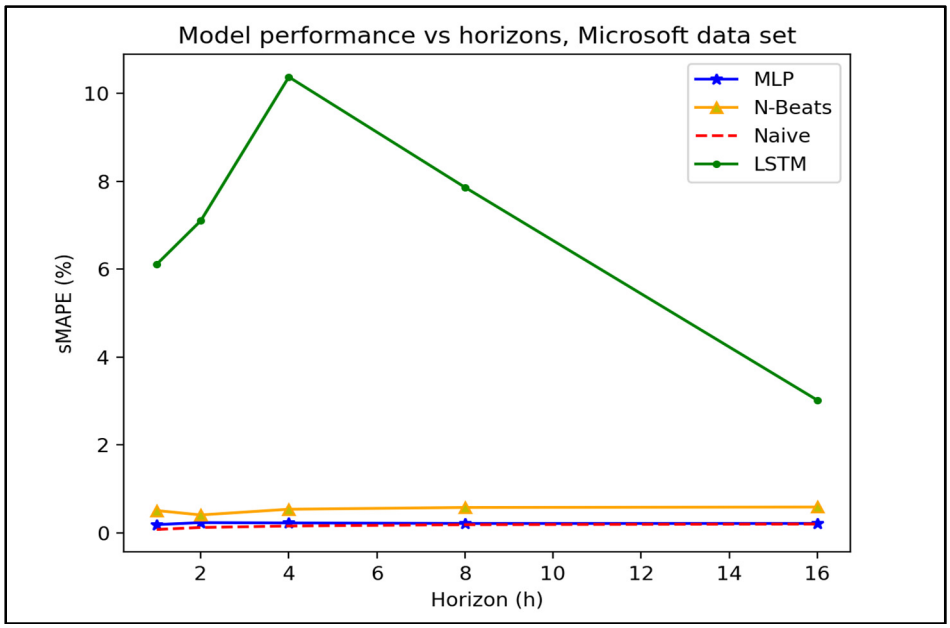

Figure 2.7 sMAPE metric indicating the performance of the models; N-Beats, MLP, LSTM and Naïve, Microsoft data set

Table 2.7 sMAPE (%) of the models, Microsoft data set

sMAPE	1 hour	2 hours	4 hours	8 hours	16 hours
MLP	0.19	0.236	0.237	0.2158	0.2173
LSTM	6.11	7.10	11.07	7.86	3.02
N-BEATS	0.647	0.407	0.549	0.583	0.59
NAIVE	0.080	0.125	0.159	0.189	0.204

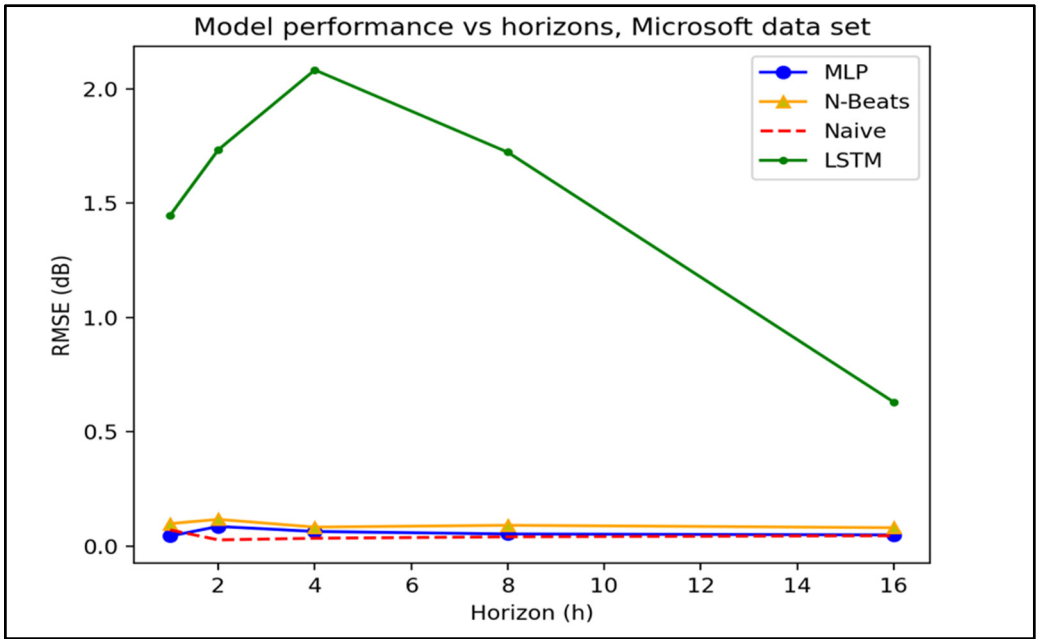

Figure 2.8 RMSE metric indicating the performance of the models; N-Beats, MLP, LSTM and Naïve, Microsoft data set

Table 2.8 RMSE (dB) of models, Microsoft data set

RMSE	1 hour	2 hours	4 hours	8 hours	16 hours
MLP	0.044	0.086	0.064	0.053	0.049
LSTM	1.447	1.734	2.083	1.723	0.630
N-BEATS	0.0983	0.1170	0.0830	0.0911	0.0806
NAIVE	0.0707	0.0273	0.0343	0.0406	0.0455

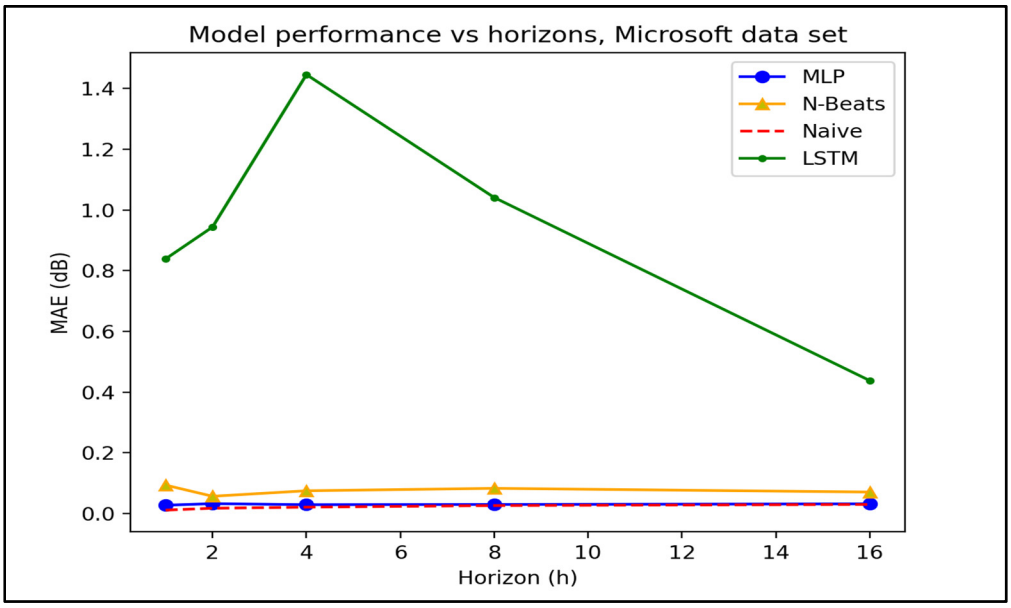

Figure 2.9 MAE metric indicating the performance of the models; N-Beats, MLP, LSTM and Naïve, Microsoft data set

Table 2.9 MAE (dB) of models, Microsoft data set

MAE	1 hour	2 hours	4 hours	8 hours	16 hours
MLP	0.0276	0.0328	0.0293	0.0301	0.0321
LSTM	0.839	0.944	1.446	1.041	0.438
N-BEATS	0.0938	0.0571	0.0752	0.0832	0.071
NAIVE	0.0114	0.0177	0.0212	0.0265	0.0301

We illustrate the sMAPE (percentage error) metric results of the models from NASP data set in Fig. 2.10 versus horizons of 1-h, 2-h to 16-h. In this figure, we can see that sMAPE of all the models are increasing by the horizon time except for LSTM. In general, the sMAPE values of the NASP data set are larger than that of Microsoft data set, although the amount of data in NASP is less than that of Microsoft data. That indicates the more diverse the data, the more increase of the error metrics (comparing the Microsoft data set (Fig. 1.1) to NASP data set (Fig. 1.2) with sMAPE of both data sets (tables 2.7 and 2.10).

As we can see in this figure (Fig. 2.10) Naïve again shows the least sMAPE among the models for all the horizons (being approximately 0.2 to 0.3% error, table 2.10). The sMAPE of MLP

is very close to that of Naïve. Both MLP and Naïve demonstrate linear increase of sMAPE error. N-Beats shows nearly 0.7 to 0.8% error of prediction of SNR in NASP data set. Its error increases gradually from 1-h to 16-h horizon. LSTM is showing the maximum error among all the models for the whole horizons. Its sMAPE hits the maximum of nearly 13.50% at 4-h horizon and then decreases until the 16-h horizon.

In Fig. 2.11, we give the RMSE (dB) metric (has been defined earlier) results of all the algorithms for the NASP data set over all the horizons (1-h, 2-h and 16-h). The trend of the performance of the models are the same as our sMAPE metric results for this data set. The RMSE error of LSTM rises to its maximum of 1.99 dB at 4-h horizon and reaches its minimum of 1.84 dB at 16-h horizon (table 2.11). The same behavior has been observed for LSTM in Microsoft data set. Naïve is showing the minimum RMSE error, linearly increasing from nearly 0.037 dB to 0.088 dB. The closet RMSE values to that of Naïve belongs to the MLP model, linearly increasing from 0.062 dB to 0.101 dB (table 2.11). N-Beats represents a moderate rise in RMSE from 0.134 to 0.137 dB. Still, comparing the RMSE of all the algorithms, the prediction error of LSTM is high for the NASP data set.

Fig. 2.12 provides the MAE (dB) metric (has been defined earlier) on the test set of all the algorithms for the NASP data set over all horizons (1-h, 2-h to 16-h). The general trend of increasing the error with longer horizons could be observed in this metric as well. The MAE of Naïve is the minimum among almost all the horizons.

The general increasing trend with the time of horizons is again observed for N-Beasts (0.079 in 1-h to 0.092 dB in 16-h), which is in agreement with the RMSE and the sMAPE metrics of N-Beats. Therefore, we may conclude that, the oscillations in the behavior of all the metrics are more pronounced for the NASP data set than for the Microsoft data set, which comes from the diversity of data more than the amount of data (in Microsoft data set, the data in each set is more than two times that of NASP data set). Comparing to the sMAPE error values, the MAE metric is representing lower value error (maximum MAE of 1.604 dB as compared to the maximum RMSE of 1.99 dB, tables 2.10 to 2.12). Overall, we can conclude that LSTM is

not showing an acceptable performance for the two target data sets with respect to the sMAPE, RMSE and MAE metrics given its structure. We use four LSTM layers and did not use the encoder-decoder LSTM structure, which could be a good target for future work. It could be one of the reasons why its error is high. On the other hand, we did not use regularization. Regularization could be a good option in case of too similar data (Microsoft data set). Although there was no over-fitting in the learning-curves, using this sort of regularization may help having more accurate results by avoiding predictions too far from the mean. Moreover, NASP data set is more diverse and therefore less predictable compared to the Microsoft data set. Therefore, the error metrics obtained from training with NASP data set are higher. Naïve appears to be a good choice for the prediction among the time series that we use for the experiments because of very short forecast horizons according to the obtained lower metrics. However, Naïve, in terms of all the error metrics, demonstrates linear increase of error, from 1-h to 16-h horizons. The linear increase of error causes higher chances of missing accuracy for longer horizons. This could be reflected in its variance error in table 2.13, which demonstrates the variances of the sMAPE errors of all the models for both data sets. From this table it could be seen that the variances of the MLP model for Microsoft and NASP data sets are minimum among all the models. In case of NASP data set, the sMAPE variance of N-Beats is less than that of Naïve (0.0024 % as compared with 0.0046 %). This implies that although Naïve model had the minimum error metrics, its growth by time was more for the NASP data set.

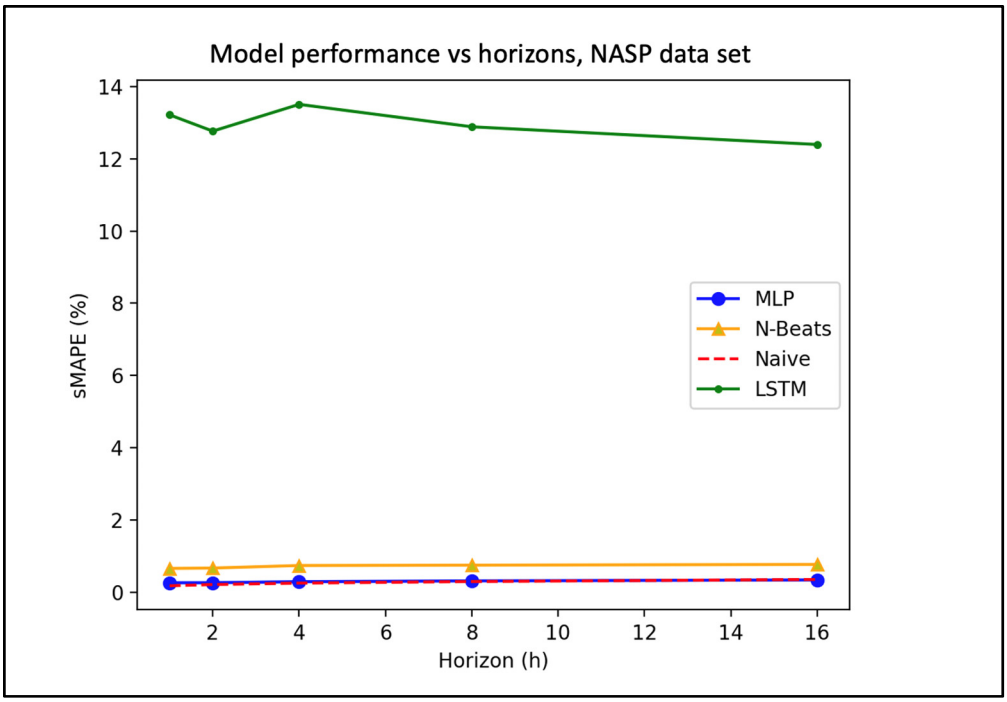

Figure 2.10 sMAPE metric indicating the performance of the models; N-Beats, MLP, LSTM and Naïve, NASP data set

Table 2.10 sMAPE (%) of models, NASP data set

SMAPE	1 hour	2 hours	4 hours	8 hours	16 hours
MLP	0.273	0.274	0.302	0.325	0.352
LSTM	13.21	12.76	13.50	12.88	12.39
N-BEATS	0.67	0.68	0.75	0.76	0.78
NAIVE	0.19	0.224	0.264	0.307	0.364

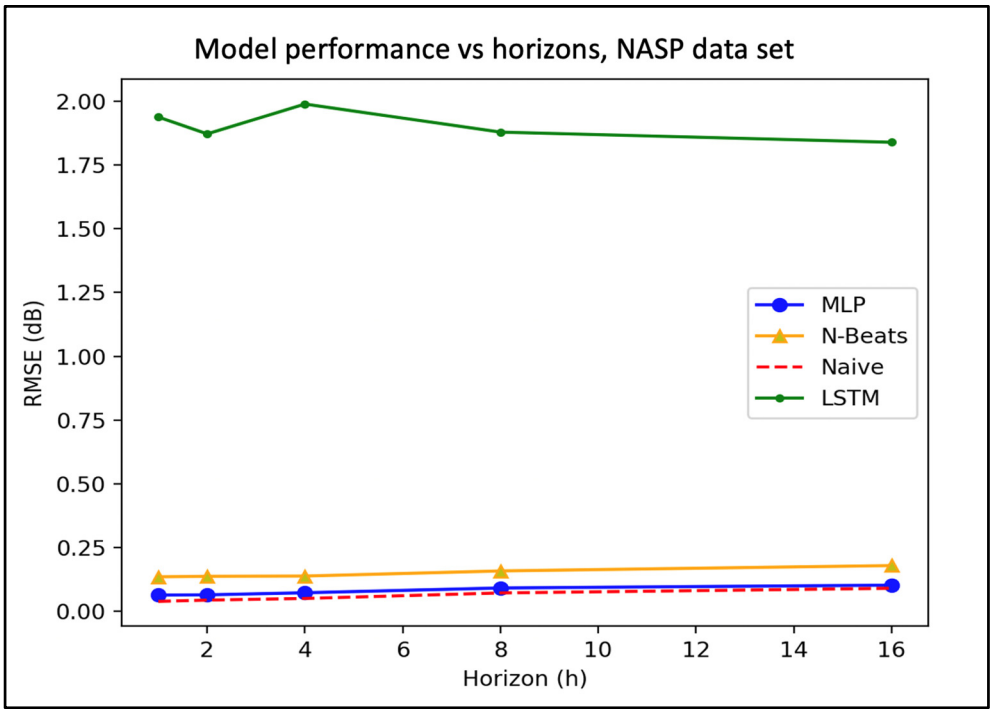

Figure 2.11 RMSE metric indicating the performance of the models; N-Beats, MLP, LSTM and Naïve, NASP data set

Table 2.11 RMSE (dB) of models, NASP data set

RMSE	1 hour	2 hours	4 hours	8 hours	16 hours
MLP	0.0624	0.0632	0.0714	0.09	0.101
LSTM	1.939	1.873	1.99	1.88	1.84
N-BEATS	0.134	0.136	0.137	0.157	0.178
NAIVE	0.0374	0.0424	0.0489	0.0707	0.0888

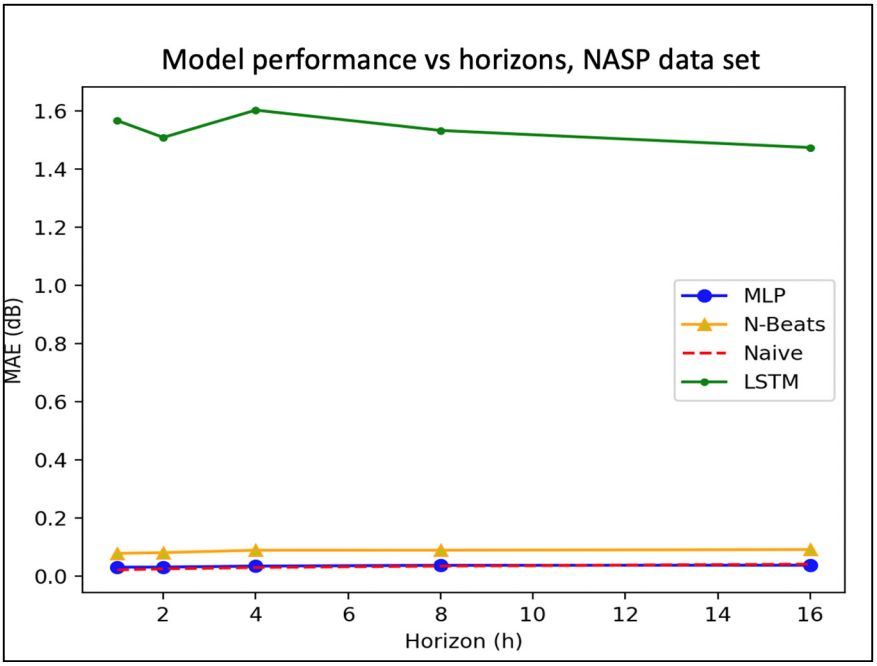

Figure 2.12 MAE metric indicating the performance of the models; N-Beats, MLP, LSTM and Naïve, NASP data set

Table 2.12 MAE (dB) of models, NASP data set

MAE Verizon	1 hour	2 hours	4 hours	8 hours	16 hours
MLP	0.0323	0.0324	0.0359	0.0385	0.03907
LSTM	1.568	1.510	1.604	1.534	1.4751
N-BEATS	0.0795	0.0820	0.0903	0.0904	0.0925
NAIVE	0.0228	0.0262	0.0311	0.0359	0.0425

Comparing our results with those of (Chouman et al., 2021), who worked with the same time series (Microsoft data set and NASP data sets), both results show a higher RMSE metric in case of NASP data set for all the models (regardless of the model and the window settings of the data) compared to the Microsoft data set. Meaning that when the data is almost flat (like the quiet lines in Microsoft data set, Fig.1.1), the more probable prediction is the target being similar to the history (naïve). This implies that naïve in this type of data could always outperform other models. However, if the target data set is more diverse, the predictions with

other models will be more preferable. In the same way, with increasing the forecasting horizon, other models can show better performances while Naïve error is still increasing linearly.

Moreover, our naïve and MLP models yield similar RMSE values for both data sets, in comparison with the results of Chouman et al. (Chouman et al., 2021). For instance, the RMSE of the MLP model, for Microsoft data set in point prediction of 16 h time interval is 0.049 dB and the one obtained by (Chouman et al., 2021) for 24 h time interval is 0.04 dB. For a more precise comparison, having the same train settings is essential, especially for LSTM as the model is very sensitive to the choice of hyperparameters (including the number of layers).

Table 2.13 Variances of the model's sMAPE for both data sets

	MLP	**LSTM**	**N-Beats**	**Naïve**
Microsoft data set	0.00036	8.485	0.0081	0.0025
NASP data set	0.0011	0.181	0.00246	0.0046

2.7 Training time

The other challenge to select a forecaster for the time series could be the time required for training. Since the test time is fast, the training time could be an indication for the time needed for each model (except for naïve). Naïve model takes minimum time since it does not involve any training. Table 2.14 represents the training time for each model in 70 epochs. In this table, LSTM and naïve show the maximum and the minimum prediction times, respectively.

2.8 Assumptions

In these sets of training, we made some simplifying assumptions: (a) The model is trained and tested on the same data set. Although the partitions of data were specified before training, we used the same time series for test and train. Meaning that we trained the model with Microsoft data set and we test the model on the same data set, (b) We assume that our LSTM architecture

is the best for the experiments, however, other architectures might be considered with better results.

Table 2.14 Training time for each of the models

	Time (s)	
	Microsoft data set	**NASP data set**
N-Beats	26	16
MLP	10	3
LSTM	4200	1440
Naive	0	0

2.9 Conclusion

We conducted a set of experiments with two time series data sets from production networks (Microsoft and NASP) to evaluate our implemented models. In comparison, Naïve showed the best performance over the other models in terms of minimum metrics for the experimented horizons. However, its variance of sMAPE error metric was not minimum among other models for each of the data sets. For NASP data set its error variance was worse than MLP and N-Beats. Moreover, its error is increasing linearly by the time. With this trend, its performance can deteriorate increasingly in forecasting longer time intervals. At the same time, N-Beats and MLP are representing more stable error metrics. Meaning that they are more reliable models in predicting lightpath Q-factor.

In addition, Naïve is not suitable for implementing uncertainty and we cannot derive confidence levels from its predictions. Although it is possible to derive the uncertainty of each prediction model using statistical analytics, the author recommends the implementation of uncertainty estimation on more reliable methods.

CHAPTER 3

UNCERTAINTY AND CONFIDENCE LEVELS

According to the literature survey, one possible approach to apply the uncertainty regarding the cost of calculations and the accuracy is approximating Bayesian inference using Monte Carlo dropout. We implement dropout, as an approximation to the Gaussian process (Gal & Ghahramani, 2016b), for the selected algorithm among our models. We select MLP due to the accuracy of the prediction of the experimented time intervals as well as its simplicity of implementation.

Using Bayesian probabilistic approach, for implementing uncertainty provides distributional inference instead of point prediction. Gaussian distribution could provide the prior distribution required by this process. For better understanding of Gaussian process (GP), and how the dropout is an approximation of the Bayesian probabilistic, the interested reader is invited to read the Appendix II of this thesis.

In this chapter, one set of MLP results in which the error is minimum is selected. The forecasting time horizon of 1 hour is the set that we select to apply the uncertainty method on due to its minimum error among examined horizons. Moreover, we select NASP data set, as a better choice of data set in terms of lightpath SNR dynamicity.

3.1 Monte Carlo dropout

Monte Carlo dropout uses the same approach as the NN dropout for regularization.

3.1.1 Dropout

Deep neural networks have the tendency to overfit when training data is limited or the model is complex (Al Osman & Shirmohammadi, 2021). Ensemble learning techniques like bagging are one strategy to prevent overfitting (Breiman, 1996). However, the resulting ensemble

increases computational complexity (Al Osman & Shirmohammadi, 2021). Dropout (Szegedy et al., 2015) is a method similar to bagging but using a different strategy that does not lead to increase in the training time or model complexity (Al Osman & Shirmohammadi, 2021). With dropout, the output of the network at each run time could be different because it generates different networks as a result of dropping some units randomly each time. Therefore, the output of each network is multiplied by a Bernoulli distribution of random variable (Al Osman & Shirmohammadi, 2021). Gal et al. (Gal & Ghahramani, 2016b) showed that the NN Monte Carlo dropout is a good approximation for the Gaussian process (GP) to define the uncertainty for the models, as well as defining the confidence intervals. It should be noted that, for regularization purpose, the dropout is used during training.

3.1.2 Applying dropout during inference time

As we mention in the previous section, using dropout during training reduces overfitting. However, to estimate the uncertainty using Monte Carlo dropout, the dropout should also be applied at test time. With this method, each point prediction will be replaced by distribution prediction.

According to the GP (Appendix II), we need the mean of distribution and the covariance function. GP could be a computationally costly process with more complicated data set. To understand the dropout strategy and how it is related to the Gaussian process, we need to explain the dropout. We consider the input data set $\{x_1, \dots, x_N\}$, the outputs $\{y_1, \dots, y_N\}$ and the prediction vector $\{\hat{y}_1, \dots, \hat{y}_N\}$ and the goal is to estimate the function $y = f(x)$. Following the Bayesian approach, our task is to find the distribution of the posterior having our data set over the space of function $p(f)$ (Rasmussen, 2004).

$$p(f|X, Y) \propto p(Y|X, f)p(f) \tag{3.1}$$

Let us consider a NN with the simplest possible case, a network with only one hidden layer (Gal & Ghahramani, 2016a). We consider the weight matrix between the first layer and the

hidden layer W_1, and the one connecting the hidden layer to the output layer W_2 (Gal & Ghahramani, 2016a). These are the linear transforms before applying the nonlinearity, which is the activation function $\sigma(.)$ such as ReLU or hyperbolic tangent (TanH). The bias is added to shift the non-linearity and denoted by b. We assume that the output has the dimension of D while its inputs are the vectors with Q dimension, and we have K hidden layers. Therefore, we have two matrices: $W_1$with the dimension $Q \times K$ and W_2 with the dimension $K \times D$ and b which is a D dimensional vector. The output of a standard NN is formulated as (Gal & Ghahramani, 2016a):

$$\hat{y} = \sigma(x\, W_1 + b)W_2 \tag{3.2}$$

During the optimization process, a regularization term could be added. L_2 regularization could be applied by a weight decay λ while minimizing the loss function (Eq. 3.3) (Gal & Ghahramani, 2016b):

$$\mathcal{L}_{\text{dropout}} = \frac{1}{N}\sum_{i=1}^{N} E(y_i, \hat{y}_i) + \lambda \sum_{i=1}^{L} (\|W_i\|_2^2 + \|b_i\|_2^2) \tag{3.3}$$

In this equation, $E(y_i, \hat{y}_i)$ is the loss function of the output value and the prediction, N is the number of data points, L is indicating the number of layers, and b_i is the bias term in each layer of the network. When the dropout is applied, two binary vectors of the dimensions Q and K should be taken into considerations, z_1 and z_2 . The elements have the Bernoulli distribution with some parameters $p_i \in [0, 1]$, for $i = 1, 2$. The value of each binary variable is 1 with the probability p_i for layer i. If the value of the binary value is 0, the unit will be dropped (Gal & Ghahramani, 2016a). Therefore, we have two Bernoulli distributions: $z_{1,q} \sim \text{Bernoulli}(p_1)$ for q = 1,…, Q and $z_{2,k} \sim \text{Bernoulli}(p_2)$ for k = 1,…, K.

With dropout, if we consider an input x, $1 - p_1$ portion of this input is set to zero and the Hadamard product of the two vectors are $x \circ z_1$. Considering the two Bernoulli distributions, we can rewrite the output Eq. (3.4) as:

$$\hat{y} = \sigma(x\,(z_1 W_1) + b)(z_2 W_2) \tag{3.4}$$

The same procedure is repeated for a network with more layers. Here by z_1 we mean the $\mathrm{diag}(z_1)$. As for any regression model, we write the Euclidean loss as:

$$E = \frac{1}{2N}\sum_{n=1}^{N} \|y_n - \hat{y}_n\|_2^2 \tag{3.5}$$

In this equation, N observed outputs are $\{y_1, \dots, y_N\}$ and $\{\hat{y}_1, \dots, \hat{y}_N\}$ are the N outputs of the model corresponding to the inputs $\{x_1, \dots, x_N\}$.

During dropout regularization, some weight decay terms are added to the equation. Usually, L_2 regularization is used with the weight decay λ and the minimization is called the cost function:

$$L_{dropout} = E + \lambda_1 \|W_1\|_2^2 + \lambda_2 \|W_2\|_2^2 + \lambda_3 \|b\|_2^2 \tag{3.6}$$

Equation (3.6) is the extended format of the equation (3.3).

One should note that after regularization, the output of the dropped weights $z_1 W_1$ and $z_2 W_2$ are scaled by the factor of $\frac{1}{p_i}$ to have the constant output. The sampling process (dropout) does not take place in the testing time. Meaning that, if we use dropout while training, the weights would be initialized by the factor of $\frac{1}{p_i}$, and during inference the weights are scaled by p_i (Gal & Ghahramani, 2016a) (if the output of a neuron is x and the probability to keep that neuron is p_i, then the output of the neuron is $p_i x + (1 - p_i)0 = p_i x$).

3.2 Uncertainty implementation

As mentioned in the beginning of this chapter, we select the MLP model predicting 1-h horizon of NASP data set due to its performance (Fig. 2.10 to 2.12, chapter 2) and simplicity of implementation for uncertainty estimation. To implement the model uncertainty using dropout, we needed to have dropout only in inference.

Our strategy is to train a MLP model, with the optimized sets of hyperparameters that we found in the previous section and to save the weights of the model. This way, we have a pre-trained model. When we run the model at the test time N number of times ($N \to \infty$), according to the central limit theorem, the accumulation of a set of random variables is also random and increasingly becomes Gaussian. In this chapter, we assume that the variational distributions as the result of dropout have the Gaussian form.

At the test time, we implement a second network with the same structure as the first one, and use the weights of the first network for the second network. In our MLP model, we have three dense layers with 16, 16 and 1 units, respectively. The first one is the input layer. Therefore, we applied the dropout before the second and the third layers. As we discussed earlier, using the dropout for the regularization, input elements are randomly set to zero and other elements should be rescaled (Gal & Ghahramani, 2016a) (Fig. 3.1). Fig. 3.1. A is showing the structure of our MLP model and the Fig. 3.1. B represents the model which is using the weights of the first MLP model with dropout and rescaled dropout rate. For instance, if with a probability rate elements of input x are dropped (set to zero), then $(1 - \mathrm{rate})$ will remain. The remaining elements will be scaled by $1/(1 - \mathrm{rate})$ (*Tf.Nn.Dropout | TensorFlow Core v2.5.0*, n.d.)(Lambrugh, n.d.)). This way, the output values will remain intact. Since we do not apply the dropout during training, so that TensorFlow (or any other framework used for the implementation) can rescale the output, we need to do the rescaling explicitly. Hence, we added layers to multiply the output of each dropout later by $(1 - \mathrm{rate})$ so that we could have the dropout for the uncertainty approximation and, at the same time, rescale outputs to the expected values. It is worthwhile to mention that failing to do so will cause a shift to appear in the

Probability Distribution Function (PDF) diagrams (will be discussed in the results). The more the dropout rate, the greater the shift is. Therefore, performing this rescaling is essential.

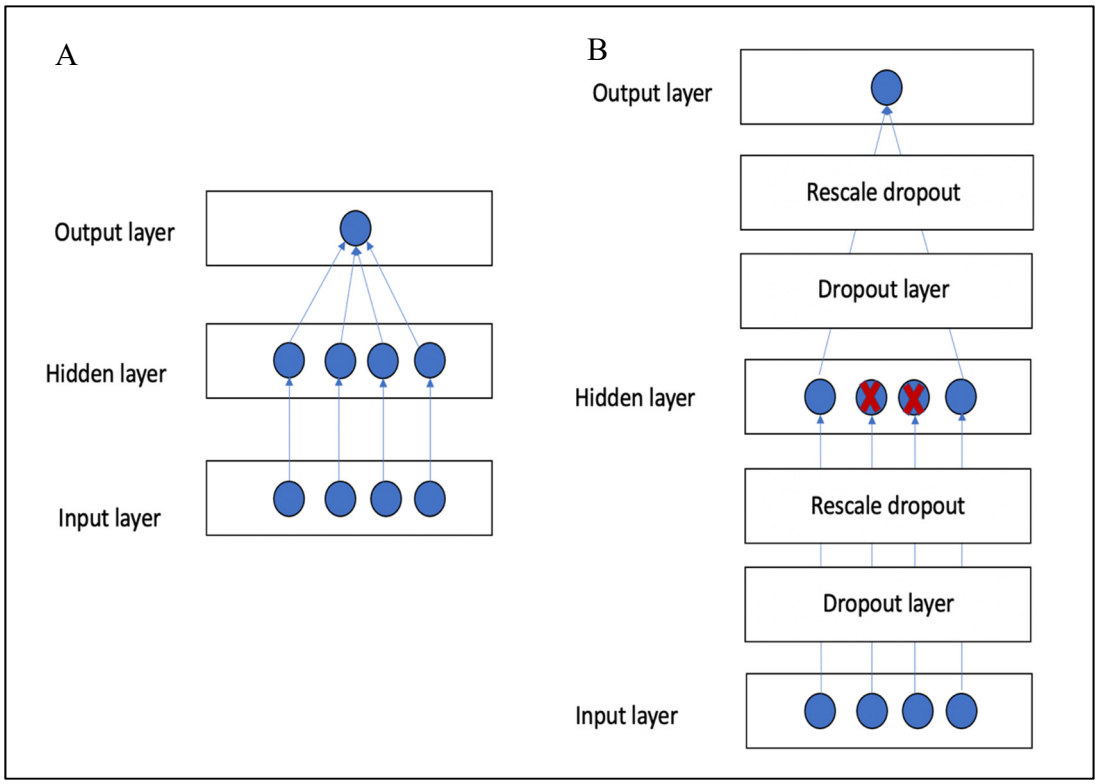

Figure 3.1 The MLP models with and without dropout. A the MLP model without dropout, B the MLP model with dropout and with rescaling the dropout

To evaluate the uncertainty implemented with dropout, we need a comparison baseline. We select quantile regression as another method to estimate the confidence margins. Quantile Regression was introduced by Koenker and Bassett (Koenker & Bassett, 1978) and it estimates the conditional median of the target. A special case of quantile regression is the Least Absolute Deviation (LAD) Koenker and Bassett (Koenker & Bassett, 1978), that fits the medians to the linear function of the covariates. Part of the attraction of the LAD estimation is because median can be a better measure for location than mean. Some useful features of quantile regression are: (a) It has a linear representation which makes its estimation easier, given a set of regressors, (b) The model can be used for characterizing the entire conditional distribution, quantile regression function is the weighted sum of absolute deviations, which is a robust measure of location. Therefore, the estimated vector of coefficients is not sensitive to the

outliers of the dependent variable. (c) Finally, in case of non-normal error term, quantile regression could be more efficient than the least square error.

3.3 Quantile Regression

Quantiles are providing us with the information about the mass of the data. They summarize the univariate probability distributions and could be depicted with the box plots (Koenker, 2017). They are sensitive to the fluctuations of the mass of the data. With quantile, one can move the mass around the median, either above or below the median, without distributing it, provided that the mass does not move from above to below the median or another way round. This locality characteristic has made the quantile very robust compared to the conventional statistical calculations. We can use quantile to estimate the median or any other quantile (e.g., 25% or 75%). Moreover, we may do so in the presence of outliers, when the residuals are not normal, or there is an increase in the error variance for the expected output variable (*Efron, B. (1991). Regression Percentiles Using Asymmetric Squared Error Loss. Vol.1, No.1.*, n.d.). Univariate quantiles could be the solutions for piecewise loss problems (Koenker, 2017).

To understand quantile regression, we may recall some information about the linear regression. In a simple linear regression model, we have a function relating the independent variables x to the dependent variable y:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{np} \tag{3.7}$$

In which, p is the number of regressors and $i \in \{1, \dots, n\}$ is defining the number of data points. One of the best error estimators for linear regression is MSE:

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}))^2 \tag{3.8}$$

We can write the quantile regression model equation for the τ-th quantile:

$$Q_\tau(y_i) = \beta_0(\tau) + \beta_0(\tau)x_{i1} + \cdots + \beta_0(\tau)x_{ip} \tag{3.9}$$

here again, p is the number of regressor variables and n stands for the number of data points. the β coefficients are dependent on the quantile. Finding the β coefficeints at the specific quantile is a similar process as for linear regression with the difference of calculating the median absolute deviation:

$$\text{MAD} = \frac{1}{n}\sum_{i=1}^{n} \rho_\tau(y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})) \tag{3.10}$$

In this equation ρ is the check function (*Efron, B. (1991). Regression Percentiles Using Asymmetric Squared Error Loss. Vol.1, No.1.*, n.d.). which gives asymmetry to the error respecting the defined quantile. We can define ρ as:

$$\rho_\tau(u) = \tau \max(u, 0) + (1 - \tau)\max(-u, 0) \tag{3.11}$$

In this equation, u is the error and it implies that if the error is positive, the check function will return the error multiplied by the quantile (τ), and if the error is negative, it will return the error multiplied by the quantile (1-τ).

To implement quantile regression, we use our MLP model, and compile the model using loss function defined in Eq. 3.11. In this way, for the defined quantile value of interest, we can find the region where the mass of the data is located. For instance, we can consider 5% and 95% quantiles, define the distributional mass and compare the results with the upper bound, lower bounds of what we earlier calculated using the dropout method as the approximation to the GP.

3.4 Results and discussion

We trained the MLP model without using dropout and, in inference time, used a second model with the same structure with the dropout layers and the rescale dropout layers, while its weights are loaded from the pre-trained model. We choose to work with NASP with 1-hour time

interval time series as the data set for the experiments for implementing uncertainty. We use the same window format setting (192 SNR data points as the history to predict a data point in the future) and performed random sampling.

We run our model a given number of times and, as a result, have a vector of the size of the number of runs that contains variational instances for each prediction. This vector has a Gaussian distribution due to the central limit theorem. To select the number of runs we calculate the mean of standard deviation for each prediction and the mean of RMSE of the test set for different dropout rates in the array of [0.01, 0.02, 0.05, 0.1, 0.2] (table 3.1). From this table, it can be seen that the mean of standard deviation and mean of RMSE (dB) of the distribution of the whole test set for 50 runs at the test time, and 0.01 dropout rate, is the minimum. Moreover, this minimum does not change considerably when increasing the number of runs to 100 or more. However, the mean of standard deviation of each distribution is closer to the mean RMSE of the whole test set in case of 0.01 dropout and 50 runs. Therefore, we selected 50 runs for inference. The dropout rates less than 0.01 are also considered (e. g. 0.005), however, due to the topology of our MLP model and number of neurons, being 16 in each layer, 0.005 dropout rate, considering the randomness of the dropout, could end up with no dropped unit. Therefore, the minimum dropout rate is considered 0.01.

Fig. 3.2 represents the PDF of the residuals. We calculate the residuals as the subtraction of the distributions of each prediction from their corresponding actual value (Eq. 3.12).

$$\text{residulas} = \mathrm{Y_m} - \widehat{\mathrm{Y}}_{\mathrm{m\times n}} \tag{3.12}$$

where $\mathrm{Y_m}$ is the vector of the actual values and $\widehat{\mathrm{Y}}_{\mathrm{m\times n}}$ is the matrix of variational inference. Therefore, we produce a matrix of residuals with the dimension $\mathrm{m} \times \mathrm{n}$ (m being the number predictions (the numbers of test instances) and n being the number of runs (50)). We can observe positive tails for all PDFs, which implies that most variational inference values are less than the actual values (Fig. 3.2). Therefore, since almost all the variational inference values are less than the actual values, we only have positive residuals (Fig. 3.2).

Table 3.1 Selecting number of runs based on the mean of standard deviation per rate.

Number of runs during inference time	**Dropout rate**	**Means of standard deviation**	**Mean of RMSE (dB)**
50	0.01	1.12	1.23
	0.02	1.62	1.80
	0.05	2.43	2.93
	0.1	3.14	4.34
	0.2	3.78	6.44
100	0.01	1.15	1.24
	0.02	1.61	1.77
	0.05	2.45	2.93
	0.1	3.16	4.34
	0.2	3.80	6.46

To calculate the upper and lower bounds of the data with the dropout method, we use the quantile of the variational inference data. We select 5 and 95% quantile for lower and upper bounds, respectively. Due to the very low residual (being RMSE = 0.062 dB), the difference between the true value and the prediction is minimal. Moreover, as we expect, the upper bound is very close to the actual value because as it can be seen from Fig. 3.2, the negative tail is very short (Eq. 3.12). On the other hand, since the tails of the distributions were all in the positive

side (Fig. 3.3), the difference between the lower bound (the green line in Fig. 3.3) and the true value is reaching to 4 dB. The lower bound is higher than it is expected.

Using the dropout, we generate the distribution of the values during inference in the credible region $(-\infty, \mathrm{y}]$. Most of the data are centered around the most probable values (Fig. 3.2), and the frequency of lower values is very small. However, to capture 90% (or 95%) quantile, we capture all the infrequent data. This is why, in Fig. 3.3 obtained from the approximation to the GP, the lower bound (green line) has a considerable difference from the prediction.

In regression, intuitively, the model is called confident if, for instance, its prediction falls in 90% confidence intervals 90% of the times (Kuleshov et al., 2018). The confidence estimates are called well-calibrated when the model's prediction could be trustable and the reported confidence is high (Lakshminarayanan et al., 2017). In practice however, estimations to Bayesian uncertainty mostly fails to capture the true distribution of the data (Lakshminarayanan et al., 2017). This problem might happen due to the bias of the model. For instance, the predictor might not be able to assign the right probability to every interval, the same process that might occur in classification (Lakshminarayanan et al., 2017). According to Kuleshov et al. (Kuleshov et al., 2018), the dropout (Gal & Ghahramani, 2016b) and the ensemble techniques (Lakshminarayanan et al., 2017), may not result in calibrated predictions. Meaning that, using dropout method, with 90% confidence margin, the model tries to capture all the possible points in the variational distribution, which results in appearing the green line in Fig. 3.3, that contains the least frequent variational distributions. In this case, the model is called mis-calibrated (Kuleshov et al., 2018). One strategy proposed by Kuleshov et al. (Kuleshov et al., 2018) to solve this problem is that once 95% confidence level is considered but only 80% of the observed y_t (variational points in the distribution) fall in the considered margins, then the margins simply adjust the 80%.

Moreover, earlier, we assumed that using the Gaussian process (approximation to the Bayesian probabilistic) will result in Gaussian distribution in the inference. However, if the true distributions are not Gaussian, the uncertainty yielded from the Bayesian model could not be

calibrated (Kuleshov et al., 2018). This should be considered as one limitation of the dropout for uncertainty estimation. Therefore, calibration as one more step for implementing precise uncertainty estimation seems to be essential.

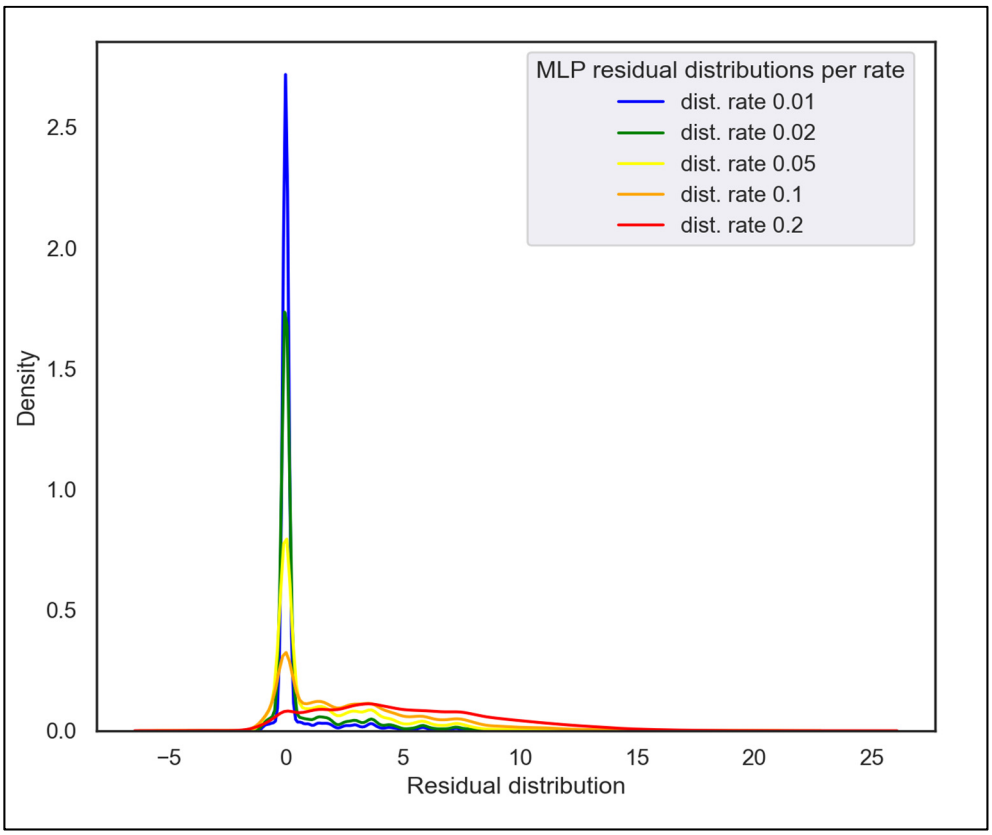

Figure 3.2 PDF of the residuals per rate

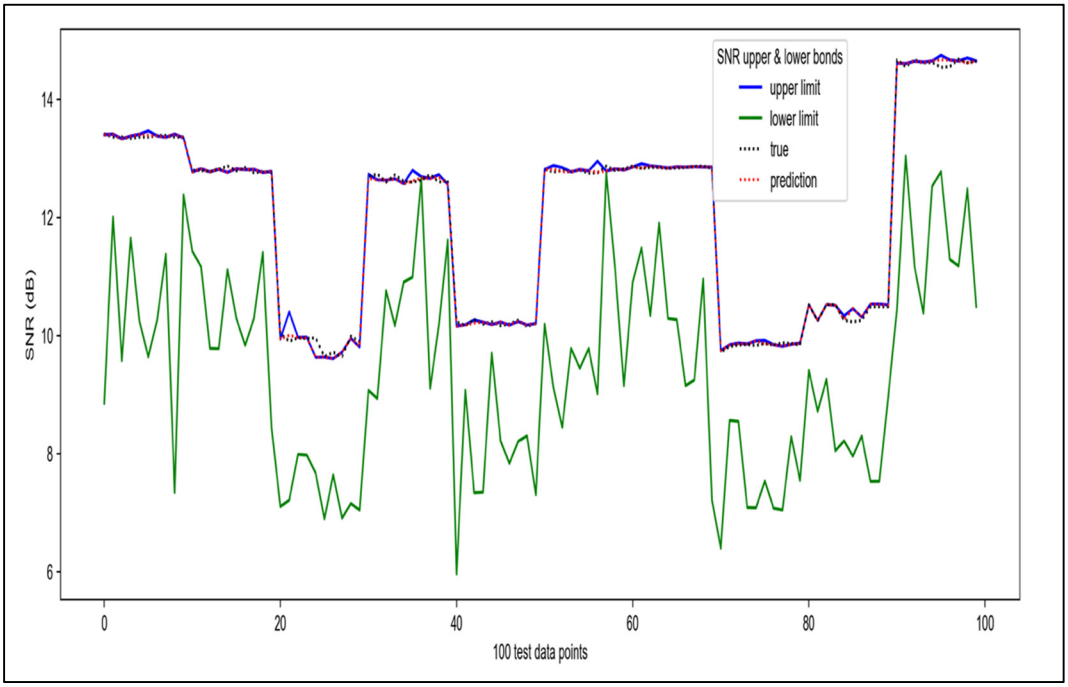

Figure 3.3 The upper and lower bonds, the true and predicted values obtained from the dropout method of uncertainty

As another measure to evaluate the upper and lower bounds, we use quantile regression. Quantile regression provides the median and the upper and lower bounds are defined to be 95% and 5% quantiles, respectively. Having the confidence margins from the two methods, now we can evaluate our uncertainty method. With this method, we obtain the median value, the upper limit, and the lower limit of the 95% quantile (Fig. 3.4). In Fig. 3.4, the upper and lower bounds are in the same distance from the median (nearly 0.1 dB). As it can be observed from Fig. 3.3 and 3.4, the two lower bounds are different. Given the output distribution of the target (obtained from method one, the dropout method) and the upper and lower bounds obtained from the quantile regression method, the main question is how to evaluate the two approaches. One way to assess which method is giving a better estimation of the confidence levels is by applying re-calibration method proposed by Kuleshov et al. (Kuleshov et al., 2018). With implementing this algorithm, 90% confidence interval contains 90% of their predictions. This process could be investigated in the future work.

Choosing a method of calibration and implementing is beyond the scope of this project but exploring and conducting a means of calibration for the Bayesian models would help evaluate the strategy to estimate uncertainty.

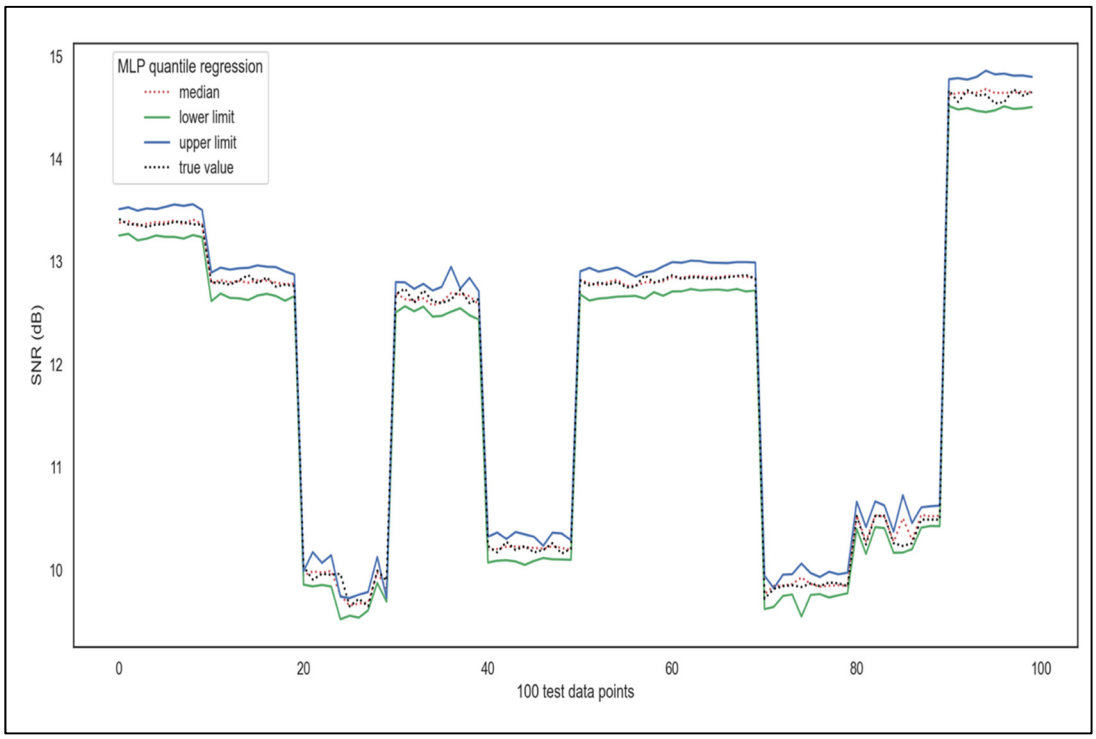

Figure 3.4 The upper and lower bonds, the true value, and the median, calculated by the quantile regression

CONCLUSION

This thesis successfully explored the application of deep learning forecasters for time series analysis in the field of optical communications. Implementing N-Beats, whose code is not yet offered as a library in Python, and have it outperformed other forecasters was one of the very interesting achievements of this thesis (Appendix I). For the target time series of this project, the performance of Naïve model was better for the point prediction of the time intervals of choice, according to the evaluation metrics. Naïve shows optimal performance when the signal is stable over time. However, according to its error variance, this method is not recommended.

The uncertainty was implemented using the NN dropout to the MLP model. The margins of data as the indicators of having variational inference instead of point prediction, were also evaluated by the margins found with the quantile regression method. The results from the two approaches were different. Although the results obtained from quantile regression seem better, the results obtained from dropout method are acceptable from Gaussian process approximation (Levi et al., 2020). The reason is that using Bayesian uncertainty most of the times do not result in capturing the true distribution of the data (Lakshminarayanan et al., 2017). Because using this method, the distribution is somewhere in the range $(-\infty, \mathrm{y}]$. As the PDF diagrams demonstrate (Fig. 3.2), the distributions of each of the examined dropout rates are not Gaussian while we assumed them to be Gaussian. As the result of these approximations, the lower margin is lower than it is expected. Therefore, a calibration method is needed to adjust the lower margin so that, for instance, 80% of predictions can be in the 80% confidence margin. This is the goal that could be accomplished by deploying a calibration method as the further steps in implementing uncertainty with Bayesian approximations.

RECOMMENDATIONS

According to our results, we give the following recommendations for future work on the topic: (a) Selecting more diverse time series, i.e., lightpaths which exhibit some performance dynamicity over time. In Microsoft data set, similarity of the data caused serious problems of forecasting with almost all the DL forecasters. (b) Using some regularization methods such as dropout or batch normalization while training is highly recommended to avoid over-fitting problems. (c) Incorporating a technique for uncertainty estimation to the other DL models, such as N-Beats and LSTM, and investigating novel ways of doing so would be an interesting topic for another project. (d) Calibrating the uncertainty models using strategies for the regression problems is recommended to validate the implemented method for estimating uncertainty, and to increase the confidence of the predictions.

ANNEX I

N-BEATS MODEL WITH TWO STACKS AND THREE BLOCKS IN EACH STACK

We hard coded the N-Beats model:

```
def make_nbeats_2():
  layer_width=128
  input_widith =10
  blocks_forecast_1 =[]
  inputs = tf.keras.Input(shape=(input_widith,),dtype=None,name= 'input')

  x1 = inputs
  for i in range (3):
     x1 = tf.keras.layers.Dense(layer_width, activation='relu')(x1)
  theta_b1 = tf.keras.layers.Dense(layer_width, name = 'theta_b1')(x1)
  theta_f1 = tf.keras.layers.Dense(layer_width, name = 'theta_f1')(x1)
  forecast1 = tf.keras.layers.Dense(layer_width,name = 'forecast1')(theta_f1)
  backcast1 = tf.keras.layers.Dense(input_widith,name = 'backcst1')(theta_b1)

  blocks_forecast_1.append(forecast1)

  residual1 = inputs
  residual1 = tf.keras.layers.Subtract()([residual1,backcast1])

  x2 = residual1
  for i in range (3):
     x2 = tf.keras.layers.Dense(layer_width, activation='relu')(x2)

  theta_b2 = tf.keras.layers.Dense(layer_width, name = 'theta_b2')(x2)
```

```
    theta_f2 = tf.keras.layers.Dense(layer_width, name = 'theta_f2')(x2)
    forecast2 = tf.keras.layers.Dense(layer_width,name = 'forecast2')(theta_f2)
    backcast2 = tf.keras.layers.Dense(input_widith,name = 'backcst2')(theta_b2)

    blocks_forecast_1.append(forecast2)

    residual2 = residual1
    residual2 = tf.keras.layers.Subtract()([residual2,backcast2])

    x3 = residual2
    for i in range (3):
       x3 = tf.keras.layers.Dense(layer_width, activation='relu')(x3)

    theta_b3 = tf.keras.layers.Dense(layer_width, name = 'theta_b3')(x3)
    theta_f3 = tf.keras.layers.Dense(layer_width, name = 'theta_f3')(x3)
    forecast3 = tf.keras.layers.Dense(layer_width,name = 'forecast3')(theta_f3)
    backcast3 = tf.keras.layers.Dense(input_widith,name = 'backcst3')(theta_b3)

    blocks_forecast_1.append(forecast3)

#     output_1 = tf.keras.layers.Add()(blocks_forecast_1)
#     model_1 = tf.keras.Model(inputs=inputs, outputs= output_1)

#    stack_forecast_1 = blocks_forecast_1

############################### Next stack ##################

    #blocks_forecast_2 =[]

    residual3 = residual2
```

```
residual3 = tf.keras.layers.Subtract()([residual3,backcast3])

x4 = residual3
for i in range (3):
   x4 = tf.keras.layers.Dense(layer_width, activation='relu')(x4)

theta_b4 = tf.keras.layers.Dense(layer_width, name = 'theta_b4')(x4)
theta_f4 = tf.keras.layers.Dense(layer_width, name = 'theta_f4')(x4)
forecast4 = tf.keras.layers.Dense(layer_width,name = 'forecast4')(theta_f4)
backcast4 = tf.keras.layers.Dense(input_widith,name = 'backcst4')(theta_b4)

blocks_forecast_1.append(forecast4)

residual4 = residual3
residual4 = tf.keras.layers.Subtract()([residual4,backcast4])

x5= residual4
for i in range (3):
   x5 = tf.keras.layers.Dense(layer_width, activation='relu')(x5)

theta_b5 = tf.keras.layers.Dense(layer_width, name = 'theta_b5')(x5)
theta_f5 = tf.keras.layers.Dense(layer_width, name = 'theta_f5')(x5)
forecast5 = tf.keras.layers.Dense(layer_width,name = 'forecast5')(theta_f5)
backcast5 = tf.keras.layers.Dense(input_widith,name = 'backcst5')(theta_b5)

blocks_forecast_1.append(forecast5)

residual5 = residual4
residual5 = tf.keras.layers.Subtract()([residual5,backcast5])
```

```
    x6 = residual5
    for i in range (3):
        x6 = tf.keras.layers.Dense(layer_width, activation='relu')(x6)

    theta_b6 = tf.keras.layers.Dense(layer_width, name = 'theta_b6')(x6)
    theta_f6 = tf.keras.layers.Dense(layer_width, name = 'theta_f6')(x6)
    forecast6 = tf.keras.layers.Dense(layer_width,name = 'forecast6')(theta_f6)
    backcast6 = tf.keras.layers.Dense(input_widith,name = 'backcst6')(theta_b6)

    blocks_forecast_1.append(forecast6)

    stack_forecast = blocks_forecast_1

    output_2 = tf.keras.layers.Add()(stack_forecast)
    model_2 = tf.keras.Model(inputs=inputs, outputs= output_2)

    return model_2
```

ANNEX II

GAUSSIAN PROCESS

Deep Gaussian Process

Deep Gaussian process gives us a tool to model distributions over functions (Gal & Ghahramani, 2016b) and is a collection of any finite number of random variables which have joint Gaussian distribution (Rasmussen, 2004).

A Gaussian Process is completely determined with its mean $m(x)$ and the covariance function $k(x, \acute{x})$. The Gaussian process is defined over the function versus the Gaussian distribution which is defined over the vector (Eq. A II-1).

$$f \sim GP(m, k) \tag{A II-1}$$

which means that “the function of f is distributed as a GP with mean function m and the covariance function k" (Rasmussen, 2004) In this process for each input x there is a random function $f(x)$, "which is the value of stochastic function at that location.” (Rasmussen, 2004). Using the mean m and the covariance k , and after calculating the Cholesky decomposition of the matrix, we will be able to a move from a Gaussian process to the Gaussian distribution for a random function (Rasmussen, 2004). For instance, for the following mean function and covariance functions, and through a process derive the Gaussian distribution (Rasmussen, 2004) (Eq. A II-2):

$$f \sim GP(m, k), \text{where } m(x) = \frac{1}{4}x^2, \text{and } k(x, \acute{x}) = \exp(\frac{1}{2}(x - \acute{x})^2) \tag{A II-2}$$

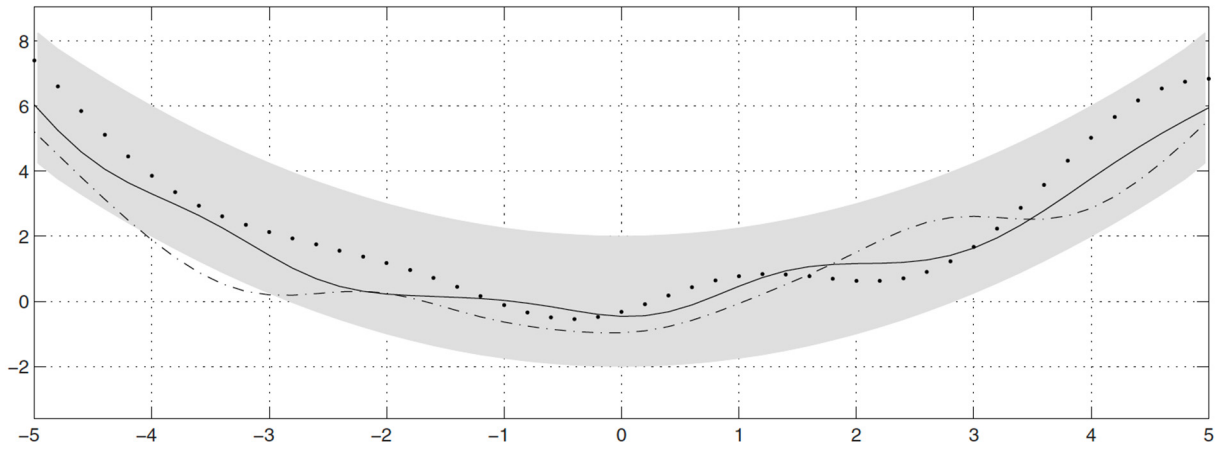

Figure A II-1 The three lines are derived from the function values at random inputs from the GP (Eq. A II-1). The gray area represents the 95% confidence interval
Taken from Rasmussen (2004)

In the next section, we will show how to have inference about functions.

Gaussian Process for posterior

In the previous section we defined a distribution over a function using the GP process. Now, we will use the GP as a prior for Bayesian inference. It is important to note that the prior is not dependent on the training data but contains some information about the functions (Rasmussen, 2004) . In this section we first explain the prior can be updated given the training data.

The aim to calculate the posterior is to make prediction about the unseen test data. Here, we let f be the function values of the training data set, X, and f_* be the ones of the test data set, X_*. We can write the joint distribution as:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim N\left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma_* \\ \Sigma_*^T & \Sigma_{**} \end{bmatrix}\right) \tag{A II-3}$$

where, $\mu = m(x_i), i = 1, \dots, n$ for the means of the training set and μ_* is the means of the test set. With the same analogy, Σ stands for the covariance of the training set, Σ_* is the covariance of the train-test set and Σ_{**} is representing the test set covariances. Suppose that we know the

values of the training set with the f function. Now, we are interested in finding the conditional distribution of f_* given f (Rasmussen, 2004) as:

$$f_*|f \sim N(\mu_* + \Sigma_*^T\Sigma^{-1}(f-\mu), \Sigma_{**} - \Sigma_*^T\Sigma^{-1}\Sigma_*) \quad \text{(A II-4)}$$

Which defines the posterior distribution for the specific test set. Now, the corresponding posterior process could be defined as:

$$f|D \sim GP(m_D, k_D), \quad \text{(A II-5)}$$

$$m_D(x) = m(x) + \Sigma^{-1}(f-m) \quad \text{(A II-6)}$$

$$k_D(x, \acute{x}) = k(x, \acute{x}) - \Sigma(X, x)^T\Sigma^{-1}\Sigma(X, \acute{x}) \quad \text{(A II-7)}$$

"where $\Sigma(X, x)$ is the covariance vector between any training case and x (Rasmussen, 2004).

These are the fundamental equations of the GP predictions. The same procedure applies for the covariance and mean of the posterior. The variance of the posterior $k_D(x, x)$ is the same as that of prior, $k(x, x)$ minus one positive term; which means that the variance of the prior is always bigger than that of posterior.

In the GP models, a common term is mostly added which is the noise. Since the distribution of the noise is considered independent, we only add the variance of the noise to f(x):

$$y(x) = f(x) + \varepsilon, \quad \varepsilon \sim N(0, \sigma_n^2), \quad \text{(A II-8)}$$

$$f \sim GP(m, k), \quad y \sim GP(m, k + \sigma_n^2\delta_{i\acute{i}}) \quad \text{(A II-9)}$$

where $\delta_{i\acute{i}} = 1$ only if $i = \acute{i}$ and is the delta function.

Now, we can repeat the same numerical example for 20 training points for the posterior distribution, as we did for the prior (Fig. A-II-2) ((Rasmussen, 2004).

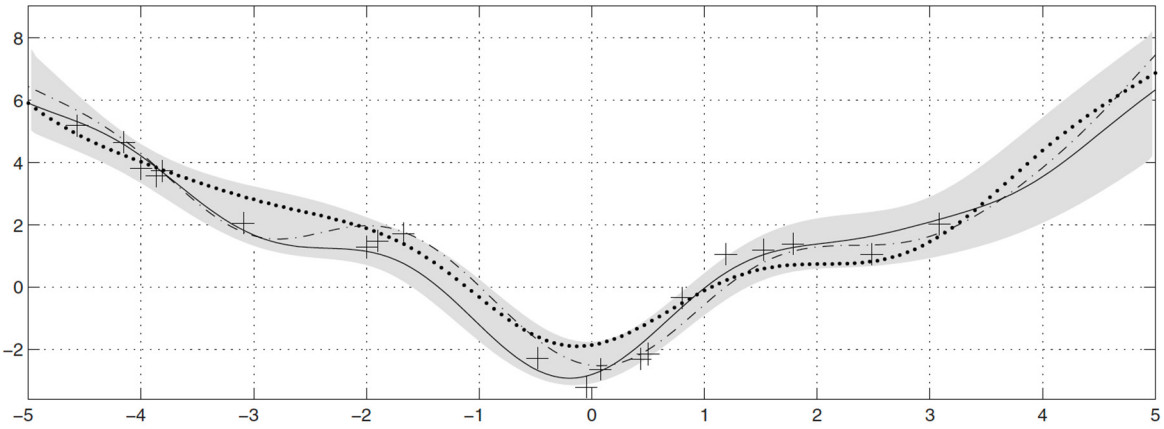

Figure A II-2 The three lines are derived randomly from the posterior, having 20 training points. The gray area represents the 95% confidence interval. The noise is also considered Taken from Rasmussen (2004)

The Gaussian process has a good advantage of being easily interpreted for simple data sets, however, a drawback of using this method may lack in the computational power for complex data sets. Therefore, in practice, the method is not pragmatic (Rasmussen, 2004).

Training process for GP

In the previous section we saw the process to update the prior given the training data. This process is applicable when the data is simple or in other words when we have enough prior information about the data. However, this is not the case for most of the applications. Therefore, we need a process that makes us able to choose between different mean and covariance functions (Rasmussen, 2004). This process is the GP training process.

We want to assess our model by making inference bout all the hyperparameters. To do so, we need to calculate the probability of the data having the hyperparameters. By assumption, this distribution is Gaussian (Rasmussen, 2004).

$$L = \log p(y|x,\theta) = -\frac{1}{2}\log|\Sigma| - \frac{1}{2}(y-\mu)^T\Sigma^{-1}(y-\mu) - \frac{n}{2}\log(2\pi) \qquad \text{(A II-10)}$$

This equation is called marginal likelihood. Now, we can find the set of hyperparameters that can optimize this marginal likelihood:

$$\frac{\partial \mathrm{L}}{\partial \theta_{\mathrm{m}}} = -(\mathrm{y}-\mu)^{\mathrm{T}}\Sigma^{-1}\frac{\partial \mathrm{m}}{\partial \theta_{\mathrm{m}}} \tag{A II-11}$$

$$\frac{\partial \mathrm{L}}{\partial \theta_{\mathrm{k}}} = \frac{1}{2}\mathrm{trace}\left(\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_{\mathrm{k}}}\right) + \frac{1}{2}(\mathrm{y}-\mu)^{\mathrm{T}}\frac{\partial \Sigma}{\partial \theta_{\mathrm{k}}}\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_{\mathrm{k}}}(\mathrm{y}-\mu) \tag{A II-12}$$

in which θ_{m} and θ_{k} are indicating the hyperparameters of the mean and covariance functions, respectively (Rasmussen, 2004).

Having reviewed the Gaussian process, we have seen how it is possible to define a non-linear regression with this process. Many of these mean and covariance functions are known and finding the ones with specific properties is an important task. It should be noted the GP is well suited the simple data sets. Otherwise, for more complex data, computing the inverse of the covariance matrix Σ, is computationally costly and limiting. Therefore, these methods are relying on the approximations (Rasmussen, 2004).

Dropout

As we discussed in the previous section, GP could be a computationally costly process with more complicated data set. Gal et al. (Gal & Ghahramani, 2016b) showed that the NN Monte Carlo dropout is a good approximation for the Gaussian process to define the uncertainty for the models as well as defining the confidence intervals. They showed that using the dropout method before each weight layer in an arbitrary network with non-linearities, is mathematically equivalent to the probabilistic deep Gaussian process (Damianou & Lawrence, 2013). The significant importance of this method is that the dropout will minimize the Kullback-Leibler (KL)divergence [68]. The reader is encouraged to read the appendix of Gal et al. (Gal & Ghahramani, 2016b).

To understand the dropout strategy and how it is related to the Gaussian process, we need to explain the dropout. Dropout method is used in NNs for regularization and to avoid overfitting. In our prediction process with the NN models that we implemented, we have a point prediction and the prediction vector ($\hat{y}_n$) corresponds to each data in the data set (x_n), as the output of a network with L layers and a loss function, Euclidean (squared loss), or softmax loss. We use notations of Gal et al. (Gal & Ghahramani, 2016b) here. We consider the input data set $\{x_1, \dots, x_N\}$, the outputs $\{y_1, \dots, y_N\}$ and the prediction vector $\{\hat{y}_1, \dots, \hat{y}_N\}$ and the goal is to estimate the function $y = f(x)$. Following the Bayesian approach, our task is to find the **distribution of the posterior** having our data set over the space of function $p(f)$ ((Rasmussen, 2004).

$$p(f|X, Y) \propto p(Y|X, f)p(f) \qquad \text{(A II-13)}$$

This distribution is containing the most likely functions respecting the seen data (Gal & Ghahramani, 2016a). For the regression tasks, we implement the joint Gaussian distribution over all the values of the function:

$$F|X \sim N(0, K(X,X))$$
$$Y|F \sim N(F, \tau^{-1} I_N)$$

in which τ is the set of hyperparameters and I_N is a $N \times N$ identity matrix.

Let us consider a NN with the simplest possible case, a network with only one hidden layer (Gal & Ghahramani, 2016a). We consider the weight matrix between the first layer and the hidden layer W_1 , and the one connecting the hidden layer to the output layer W_2 (Gal & Ghahramani, 2016a). These are the linear transforms before applying the nonlinearity, which is the activation function $\sigma(.)$ such as ReLU or hyperbolic tangent (TanH). The bias is added to shift the non-linearity and denoted by b . We assume that the output has the dimension of D while its inputs are the vectors with Q dimension, and we have K hidden layers. Therefore, we have two matrices: $W_1$with the dimension $Q \times K$ and W_2 with the dimension $K \times D$ and b which is a D dimensional vector. The output of a standard NN is formulated as (Gal & Ghahramani, 2016a):

$$\hat{y} = \sigma(x\, W_1 + b) W_2 \qquad \text{(A II-14)}$$

During the optimization process, a regularization term could be added. L_2 regularization could be applied by a weight decay λ while minimizing the loss function (Eq. 15) (Gal & Ghahramani, 2016b):

$$\mathcal{L}_{dropout} = \frac{1}{N} \sum_{i=1}^{N} E(y_i, \hat{y}_i) + \lambda \sum_{i=1}^{L} (\|W_i\|_2^2 + \|b_i\|_2^2) \qquad \text{(A II-15)}$$

In this equation $E(y_i, \hat{y}_i)$ is the loss function of the output value and the prediction and b_i is the bias term in each layer of the network. Dropout is randomly turning off some of the units of the network. For better understanding of the process, we will extend this loss function for the dropout. When the dropout is applied, two binary vectors of the dimensions Q and K should be taken into considerations, z_1 and z_2 . The elements have the Bernoulli distribution with some parameters $p_i \in [0,1]$, for $i = 1, 2$. The value of each binary variable is 1 with the

probability p_i for layer i. If the value of the binary value is 0, the unit will be dropped (Gal & Ghahramani, 2016a).

Therefore, we have two Bernoulli distributions:

$z_{1,q} \sim \text{Bernoulli}(p_1)$ for q = 1,…, Q and

$z_{2,k} \sim \text{Bernoulli}(p_2)$ for k = 1,…, K

With the dropout, if we consider an input x, $1 - p_1$ portion of this input is set to zero and the Hadamard product of the two vectors are $x \circ z_1$. Considering the two Bernoulli distributions, we can rewrite the output Eq. A II-14 as:

$$\hat{y} = \sigma(x\,(z_1 W_1) + b)(z_2 W_2) \tag{A II-16}$$

The same procedure is repeated for a network with more layers. Here by z_1 we mean the $\text{diag}(z_1)$.

As for any regression model, we write the Euclidean loss as:

$$E = \frac{1}{2N} \sum_{n=1}^{N} \|y_n - \hat{y}_n\|_2^2 \tag{A II-17}$$

In this equation, N observed outputs are $\{y_1, \dots, y_N\}$ and $\{\hat{y}_1, \dots, \hat{y}_N\}$ are the N outputs of the model corresponding to the inputs $\{x_1, \dots, x_N\}$.

During dropout regularization, some weight decay terms are added to the equation. Usually, L_2 regularization is used with the weight decay λ and the minimization is called the cost function:

$$L_{\text{dropout}} = E + \lambda_1 \|W_1\|_2^2 + \lambda_2 \|W_2\|_2^2 + \lambda_3 \|b\|_2^2 \tag{A II-18}$$

Equation 18 is the extended format of the Eq. A II-15.

One should note that after regularization, the output of the dropped weights z_1W_1 and z_2W_2 are scaled by the factor of $\frac{1}{p_i}$ to have the constant output. The sampling process (dropout) does not take place in the testing time. Meaning that, if we use dropout while training, the weights would be initialized by the factor of $\frac{1}{p_i}$, and during inference the weights are scaled by p_i (Gal & Ghahramani, 2016a) (if the output of a neuron is x and the probability to keep that neuron is p_i, then the output of the neuron is $p_i x + (1 - p_i)0 = p_i x$).

This is a critical part of working with the dropout because failing to do so will cause a shift in the PDF diagrams (will be discussed in the results section). This is the procedure that is considered with any platform like TensorFlow to apply the dropout.

Dropout as the GP approximation

As we mentioned in the previous section, dropout is an approximation to the GP (Gal & Ghahramani, 2016b). In the Gaussian process section, we stated that for the GP, we need the mean of distribution and the covariance function. To show how the dropout could be an approximation for the GP, we start from the covariance function and assume that we have the bellow function:

$$K(x, y) = \int p(w)p(b)\sigma(w^T x + b)\sigma(w^T y + b)\, dwdb \qquad \text{(A II-19)}$$

In this equation $\sigma(.)$ is the non-linearity such as ReLU or hyperbolic tangent (TanH), and $p(w)$ and $p(b)$ are the distributions over the weights and the bias, respectively (Gal & Ghahramani, 2016a, 2016b), where $p(w)$ has the normal distribution and $p(b)$ has some distribution.

To approximate Eq. 19, Gal et al. [65], used the Monte Carlo integration for a network of K hidden units. They re-parametrized the GP model and maginalized it using $W_1, W_2,$ and b as the random variables (Eq. 20). Then, they approximated the posterior over the mentioned variables using the approximations to the "variational distributions" (Gal & Ghahramani, 2016a) (to have a better understanding of the procedure of the approximation to the variational

inference, the reader is invited to read the appendix of the paper by Gal et al. (Gal & Ghahramani, 2016b)) (Eq. A II-20).

$$\widehat{K}(x,y) = \frac{1}{K}\sum_{k=1}^{K}\sigma(w_k^T x + b_k)\sigma(w_k^T y + b_k) \qquad \text{(A II-20)}$$

$$F|X,\ W_1, b \sim N\left(0, \widehat{K}(x,y)\right)$$

$$Y|F \sim N(F, \tau^{1-} I_N)$$

In Eq. 20, $\widehat{K}(x,y)$ is the GP of $K(x,y)$, the covariance function, $\omega_k \sim p(\omega)$, $b_k \sim p(b)$ and K is the number of hidden units.

$$p(Y|X) = \int p(Y|F)p(W_1, b, X)p(W_1)p(b) \qquad \text{(A II-21)}$$

where the integration is over F, W_1 and b , and W_1 is the $Q \times K$ matrix. Denoting W_2 as the $K \times D$ matrix, after specific mathematical procedure (Barber & Bishop, 1998) Eq. 22 turns out to the following equation:

$$p(Y|X) = \int p(Y|X, W_1, W_2, b)p(W_1)p(W_2)p(b) \qquad \text{(A II-22)}$$

where the integration is over W_1 , W_2 and b. This equation is the GP which is re-parametrized and marginalized (Gal & Ghahramani, 2016a) respecting W_1 , W_2 and b as random variables. Then, similar to the previous section, where we explained the GP, there is the process of approximation of the posterior distribution. The interested reader is invited to read the appendix of the paper Gal et al. (Gal & Ghahramani, 2016b).

Posterior distribution $p(\omega|X,Y)d\omega$ is hard to control (Gal & Ghahramani, 2016b). Let us consider $q(\omega)$, the distribution over the matrices which their columns are randomly set to zero by applying the dropout. This distribution could serve as an approximation to the posterior. $q(\omega)$ could be written as (Gal & Ghahramani, 2016b):

$$W_i = M_i . \mathrm{diag}([z_{i,j}]) \tag{A II-23}$$

$$z_{i,j} \sim \mathrm{Bernoulli}(p_i) \quad \text{for } i = 1, \dots, L,$$
$$j = 1, \dots, K_{i-1}$$

with respect to the probability p_i and $M_i$as the variational parameters. The parameter $z_{i,j}$ is set to zero when we drop the jth unit from the layer $i-1$. There is a strong correlation between the weights in the weight metrices (Gal & Ghahramani, 2016b).

The last step is minimizing the KL divergence:

$$-\int q(\omega) \log p(Y|X,\omega) d\omega + \mathrm{KL}(q(\omega)\|p(\omega)) \tag{A II-24}$$

In this minimization process, $q(\omega)$ is the approximation of the posterior. The first and the second terms could be approximated by $-\sum_{n=1}^{N} q(\omega) \log p(y_n|x_n,\omega)\, d\omega$, $\widehat{\omega}_n \sim q(\omega)$ and $\sum_{i=1}^{L}(\frac{p_i l^2}{2}\|M_i\|_2^2 + \frac{l^2}{2}\|m_i\|_2^2)$. After rescaling the model by $\frac{1}{\tau N}$, the loss of GP-Monte Carlo becomes:

$$L_{GP-MC} \propto \frac{1}{N}\sum_{n=1}^{N} \frac{-\log p(y_n|x_n,\widehat{\omega}_n)}{\tau} + \sum_{i=1}^{L}(\frac{p_i l^2}{2\tau N}\|M_i\|_2^2 + \frac{l^2}{2\tau N}\|m_i\|_2^2) \tag{A II-25}$$

The loss function, first term in Eq. (A II-24), now could be written as $E(y_n, \hat{y}(x_n, \widehat{\omega}_n))$ and could be approximated as (Gal & Ghahramani, 2016b):

$$E(y_n, \hat{y}(x_n, \widehat{\omega}_n)) = -\log p(y_n|x_n,\widehat{\omega}_n)/\tau \tag{A II-26}$$

in which τ is a set of hyperparameters and $\widehat{\omega}_n$ are the sampled weights.

So far, we implemented three forecasting models and a baseline (Naïve). We evaluated their prediction performances in chapter three. According to our results, Naïve showed the best performance for the two time-series we performed experiments on. However, the problem with Naïve is that we can't define its confidence about the results (sMAPE, MAE and MAE metrics, Fig. 3.9). Based on the results, after Naïve, MLP is showing the most robust results. In addition to its simple implementation, makes this model a good candidate for implementing uncertainty with the selected method that we have mentioned earlier in this chapter.

LIST OF BIBLIOGRAPHICAL REFERENCES

A- 1.6 The basic steps in a forecasting task | Forecasting: Principles and Practice (2nd ed). (n.d.). Retrieved December 6, 2021, from https://Otexts.com/fpp2/

A- 2.3 Time series patterns | Forecasting: Principles and Practice (2nd ed). (n.d.). Retrieved December 6, 2021, from https://Otexts.com/fpp2/

A- 3.1 Some simple forecasting methods | Forecasting: Principles and Practice (2nd ed). (n.d.). Retrieved August 3, 2021, from https://Otexts.com/fpp2/

A- 6.1 Time series components | Forecasting: Principles and Practice (2nd ed). (n.d.). Retrieved December 6, 2021, from https://Otexts.com/fpp2/

A- 6.2 Moving averages | Forecasting: Principles and Practice (2nd ed). (n.d.). Retrieved December 6, 2021, from https://Otexts.com/fpp2/

A- 8.1 Stationarity and differencing | Forecasting: Principles and Practice (2nd ed). (n.d.). Retrieved December 6, 2021, from https://Otexts.com/fpp2/

A- 8.3 Autoregressive models | Forecasting: Principles and Practice (2nd ed). (n.d.). Retrieved December 6, 2021, from https://Otexts.com/fpp2/

A- 11.3 Neural network models | Forecasting: Principles and Practice. (n.d.). Retrieved July 12, 2020, from https://Otexts.com/fpp2/

A- A Survey on Application of Machine Learning Techniques in Optical Networks. (n.d.). GroundAI. Retrieved August 29, 2020, from https://www.groundai.com/project/a-survey-on-application-of-machine-learning-techniques-in-optical-networks/1

Al Osman, H., & Shirmohammadi, S. (2021). Machine Learning in Measurement Part 2: Uncertainty Quantification. IEEE Instrumentation & Measurement Magazine, 24, 23–27. https://doi.org/10.1109/MIM.2021.9436102

Aladin, S., Tran, A. V. S., Allogba, S., & Tremblay, C. (2020). Quality of Transmission Estimation and Short-Term Performance Forecast of Lightpaths. Journal of Lightwave Technology, 38(10), 2807–2814. https://doi.org/10.1109/JLT.2020.2975179

Aladin, S., & Tremblay, C. (2018). Cognitive Tool for Estimating the QoT of New Lightpaths. Optical Fiber Communication Conference (2018), Paper M3A.3, M3A.3. https://doi.org/10.1364/OFC.2018.M3A.3

Assaad, M., Boné, R., & Cardot, H. (2008). A New Boosting Algorithm for Improved Time-Series Forecasting with Recurrent Neural Networks. Information Fusion, 9, 41–55. https://doi.org/10.1016/j.inffus.2006.10.009

Barber, D., & Bishop, C. M. (1998). Ensemble learning in Bayesian neural networks. Undefined. https://www.semanticscholar.org/paper/Ensemble-learning-in-Bayesian-neural-networks-Barber-Bishop/78d19e190dc5ece1e4e42a76662c5e98c2bb0842

Bishop, C. M. (1994). Mixture density networks [Monograph]. Aston University. http://publications.aston.ac.uk/id/eprint/373/

Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015). Weight Uncertainty in Neural Networks. ArXiv:1505.05424 [Cs, Stat]. http://arxiv.org/abs/1505.05424

Brando, A., Rodríguez-Serrano, J. A., Ciprian, M., Maestre, R., & Vitrià, J. (2019). Uncertainty Modelling in Deep Networks: Forecasting Short and Noisy Series. In U. Brefeld, E. Curry, E. Daly, B. MacNamee, A. Marascu, F. Pinelli, M. Berlingerio, & N. Hurley

(Eds.), Machine Learning and Knowledge Discovery in Databases (pp. 325–340). Springer International Publishing. https://doi.org/10.1007/978-3-030-10997-4_20

Breiman, L. (1996). Bagging predictors. Machine Learning, 24(2), 123–140. https://doi.org/10.1007/BF00058655

Brownlee, J. (2016, December 1). What Is Time Series Forecasting? Machine Learning Mastery. https://machinelearningmastery.com/time-series-forecasting/

Brownlee, J. (2018, July 19). Difference Between a Batch and an Epoch in a Neural Network. Machine Learning Mastery. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

Brownlee, J. (2019a, January 20). How to Control the Stability of Training Neural Networks With the Batch Size. Machine Learning Mastery. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

Brownlee, J. (2019b, January 22). How to Configure the Learning Rate When Training Deep Learning Neural Networks. Machine Learning Mastery. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

Budhiraja, A. (2018, March 6). Learning Less to Learn Better—Dropout in (Deep) Machine learning. Medium. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

Chapter 1 Getting started | Forecasting: Principles and Practice. (n.d.). Retrieved July 13, 2020, from https://Otexts.com/fpp2/

Chapter 5 Time series regression models | Forecasting: Principles and Practice (2nd ed). (n.d.). Retrieved December 6, 2021, from https://Otexts.com/fpp2/

Chapter 8 ARIMA models | Forecasting: Principles and Practice (2nd ed). (n.d.). Retrieved December 6, 2021, from https://Otexts.com/fpp2/

Choi, H., & Yang, S. S. (2008). Network Survivability in Optical Networks with IP Prospective [Chapter]. Encyclopedia of Internet Technologies and Applications; IGI Global. https://doi.org/10.4018/978-1-59140-993-9.ch049

Choudhury, G., Lynch, D., Thakur, G., & Tse, S. (2018). Two Use Cases of Machine Learning for SDN-Enabled IP/Optical Networks: Traffic Matrix Prediction and Optical Path Performance Prediction. ArXiv:1804.07433 [Cs, Stat]. http://arxiv.org/abs/1804.07433

Chouman, H., Djukic, P., Tremblay, C., & Desrosiers, C. (2021). Forecasting Lightpath QoT with Deep Neural Networks. 2021 Optical Fiber Communications Conference and Exhibition (OFC), 1–3.

Collins, M. (2007). Ensembles and probabilities: A new era in the prediction of climate change. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 365(1857), 1957–1970. https://doi.org/10.1098/rsta.2007.2068

Damianou, A. C., & Lawrence, N. D. (2013). Deep Gaussian Processes. ArXiv:1211.0358 [Cs, Math, Stat]. http://arxiv.org/abs/1211.0358

Deep Learning. (n.d.). Retrieved July 13, 2020, from http://www.deeplearningbook.org/

DeVries, T., & Taylor, G. W. (2018). Learning Confidence for Out-of-Distribution Detection in Neural Networks. ArXiv:1802.04865 [Cs, Stat]. http://arxiv.org/abs/1802.04865

Doucet, A., de Freitas, N., & Gordon, N. (2001). An Introduction to Sequential Monte Carlo Methods. In A. Doucet, N. de Freitas, & N. Gordon (Eds.), Sequential Monte Carlo

Methods in Practice (pp. 3–14). Springer. https://doi.org/10.1007/978-1-4757-3437-9_1

Du, D., Jia, X., & Hao, C. (2016, February 8). A New Least Squares Support Vector Machines Ensemble Model for Aero Engine Performance Parameter Chaotic Prediction [Research Article]. Mathematical Problems in Engineering; Hindawi. https://doi.org/10.1155/2016/4615903

Efron, B. (1991). Regression percentiles using asymmetric squared error loss. Vol.1, No.1. (n.d.). Retrieved February 13, 2021, from http://www3.stat.sinica.edu.tw/statistica/j1n1/j1n16/j1n16.htm

Elements of Statistical Learning: Data mining, inference, and prediction. 2nd Edition. (n.d.). Retrieved July 13, 2020, from https://web.stanford.edu/~hastie/ElemStatLearn/

Gal, Y., & Ghahramani, Z. (2016a). Dropout as a Bayesian Approximation: Appendix. ArXiv:1506.02157 [Stat]. http://arxiv.org/abs/1506.02157

Gal, Y., & Ghahramani, Z. (2016b). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. ArXiv:1506.02142 [Cs, Stat]. http://arxiv.org/abs/1506.02142

Girard, A., Rasmussen, C. E., Quinonero-Candela, J., & Murray-Smith, R. (2003). Gaussian Process priors with uncertain inputs? Application to multiple-step ahead time series forecasting [Conference Proceedings]. Neural Information Processing Systems. MIT Press. https://eprints.gla.ac.uk/3117/

Gneiting, T., & Katzfuss, M. (2014). Probabilistic Forecasting. Annual Review of Statistics and Its Application, 1(1), 125–151. https://doi.org/10.1146/annurev-statistics-062713-085831

Gneiting, T., & Raftery, A. E. (2005). Weather Forecasting with Ensemble Methods. Science, 310(5746), 248–249. https://doi.org/10.1126/science.1115255

Hastie, T., & Tibshirani, R. (1987). Generalized Additive Models: Some Applications. Journal of the American Statistical Association, 82(398), 371–386. JSTOR. https://doi.org/10.2307/2289439

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778. https://doi.org/10.1109/CVPR.2016.90

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2016). Densely Connected Convolutional Networks. https://arxiv.org/abs/1608.06993v5

Hui, R. (2019). Introduction to Fiber-Optic Communications. Elsevier.

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ArXiv:1502.03167 [Cs]. http://arxiv.org/abs/1502.03167

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1999). An Introduction to Variational Methods for Graphical Models. Machine Learning, 37(2), 183–233. https://doi.org/10.1023/A:1007665907178

Koenker, R. (2017). Quantile Regression: 40 Years On. Annual Review of Economics, 9(1), 155–176. https://doi.org/10.1146/annurev-economics-063016-103651

Koenker, R., & Bassett, G. (1978). Regression Quantiles. Econometrica, 46(1), 33–50. https://doi.org/10.2307/1913643

Koochali, A., Schichtel, P., Dengel, A., & Ahmed, S. (2019). Probabilistic Forecasting of Sensory Data With Generative Adversarial Networks – ForGAN. IEEE Access, 7, 63868–63880. https://doi.org/10.1109/ACCESS.2019.2915544

Kuleshov, V., Fenner, N., & Ermon, S. (2018). Accurate Uncertainties for Deep Learning Using Calibrated Regression. ArXiv:1807.00263 [Cs, Stat]. http://arxiv.org/abs/1807.00263

Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2017). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. ArXiv:1612.01474 [Cs, Stat]. http://arxiv.org/abs/1612.01474

Lambrugh, R. (n.d.). Deep Learning With Python by Francois Chollet. In Deep Learning With Python by Francois Chollet. Retrieved July 13, 2020, from https://www.academia.edu/40318927/Deep_Learning_With_Python_by_Francois_Chollet

Levi, D., Gispan, L., Giladi, N., & Fetaya, E. (2020). Evaluating and Calibrating Uncertainty Prediction in Regression Tasks. ArXiv:1905.11659 [Cs, Stat]. http://arxiv.org/abs/1905.11659

Lewinson, E. (2020, November 1). Choosing the correct error metric: MAPE vs. sMAPE. Medium. https://towardsdatascience.com/choosing-the-correct-error-metric-mape-vs-smape-5328dec53fac

Linda, O., Vollmer, T., & Manic, M. (2009). Neural Network based Intrusion Detection System for critical infrastructures. 1827–1834. https://doi.org/10.1109/IJCNN.2009.5178592

LSTM network: A deep learning approach for short-term traffic forecast—IET Journals & Magazine. (n.d.). Retrieved June 12, 2020, from https://ieeexplore.ieee.org/abstract/document/7874313?casa_token=r_ThTSezVY0AAAAA:djMNLhZZ62OoZiGO-VXAYr4Q06MB4buCvYWWKFsBTOn013Z8matv6jk3PEW9FYJvvZx1nOel-w

Makridakis, S., & Hibon, M. (2000). The M3-Competition: Results, Conclusions and Implications. International Journal of Forecasting, 16, 451–476. https://doi.org/10.1016/S0169-2070(00)00057-1

Mcompetitions/M4-methods. (n.d.). GitHub. Retrieved July 13, 2020, from https://github.com/Mcompetitions/M4-methods

Mo, W., Gutterman, C. L., Li, Y., Zussman, G., & Kilper, D. C. (2018). Deep Neural Network Based Dynamic Resource Reallocation of BBU Pools in 5G C-RAN ROADM Networks. Optical Fiber Communication Conference (2018), Paper Th1B.4, Th1B.4. https://doi.org/10.1364/OFC.2018.Th1B.4

Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press. http://ebookcentral.proquest.com/lib/mcgill/detail.action?docID=3339490

Myslín, M. (2020, March 16). Machine learning has uncertainty. Design for it. Medium. https://towardsdatascience.com/machine-learning-has-uncertainty-design-for-it-f015a249a444

Naeini, M. P., Cooper, G. F., & Hauskrecht, M. (2015). Obtaining Well Calibrated Probabilities Using Bayesian Binning. Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence, 2015, 2901–2907.

NeurIPS 2020: (Track2) Practical Uncertainty Estimation and Out-of-Distribution Robustness in Deep Learning. (n.d.). Retrieved February 14, 2021, from https://nips.cc/virtual/2020/public/tutorial_0f190e6e164eafe66f011073b4486975.html

Ogunmolu, O., Gu, X., Jiang, S., & Gans, N. (2016). Nonlinear Systems Identification Using Deep Dynamic Neural Networks. ArXiv:1610.01439 [Cs]. http://arxiv.org/abs/1610.01439

Optical Data. (n.d.). Microsoft Download Center. Retrieved December 26, 2020, from https://www.microsoft.com/en-us/download/details.aspx?id=54267

Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2020a). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. ArXiv:1905.10437 [Cs, Stat]. http://arxiv.org/abs/1905.10437

Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2020b). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. ArXiv:1905.10437 [Cs, Stat]. http://arxiv.org/abs/1905.10437

Palmer, T. N. (2002). The economic value of ensemble forecasts as a tool for risk assessment: From days to decades. Quarterly Journal of the Royal Meteorological Society, 128(581), 747–774. https://doi.org/10.1256/0035900021643593

Quick Start. (n.d.). Prophet. Retrieved September 10, 2020, from http://facebook.github.io/prophet/docs/quick_start.html

Rasmussen, C. E. (2004). Gaussian Processes in Machine Learning. In O. Bousquet, U. von Luxburg, & G. Rätsch (Eds.), Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2—14, 2003, Tübingen, Germany, August 4—16, 2003, Revised Lectures (pp. 63–71). Springer. https://doi.org/10.1007/978-3-540-28650-9_4

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting, 36(3), 1181–1191. https://doi.org/10.1016/j.ijforecast.2019.07.001

Snyder, R., Ord, K., & Beaumont, A. (2012). Forecasting the intermittent demand for slow-moving inventories: A modelling approach. International Journal of Forecasting - INT J FORECASTING, 28. https://doi.org/10.1016/j.ijforecast.2011.03.009

Stigler, S. M. (1975). The transition from point to distribution estimation. Bulletin of the International Statistical Institute, 46(2).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going Deeper With Convolutions. 1–9. https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html

Teye, M., Azizpour, H., & Smith, K. (2018). Bayesian Uncertainty Estimation for Batch Normalized Deep Networks. In J. Dy & A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning (Vol. 80, pp. 4907–4916). PMLR. http://proceedings.mlr.press/v80/teye18a.html

Tf.nn.dropout | TensorFlow Core v2.5.0. (n.d.). TensorFlow. Retrieved July 9, 2021, from https://www.tensorflow.org/api_docs/python/tf/nn/dropout

Understanding LSTM Networks—Colah's blog. (n.d.). Retrieved September 8, 2020, from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Wang, Z., Zhang, M., Wang, D., Song, C., Liu, M., Li, J., Lou, L., & Liu, Z. (2017). Failure prediction using machine learning and time series in optical network. Optics Express, 25(16), 18553–18565. https://doi.org/10.1364/OE.25.018553

Yadav, A., Jha, C. K., & Sharan, A. (2020). Optimizing LSTM for time series prediction in Indian stock market. Procedia Computer Science, 167, 2091–2100. https://doi.org/10.1016/j.procs.2020.03.257

Yan, X., & Chowdhury, N. A. (2015, January 29). Midterm Electricity Market Clearing Price Forecasting Using Two-Stage Multiple Support Vector Machine [Research Article]. Journal of Energy; Hindawi. https://doi.org/10.1155/2015/384528

Zaremba, W., Sutskever, I., & Vinyals, O. (2015). Recurrent Neural Network Regularization. ArXiv:1409.2329 [Cs]. http://arxiv.org/abs/1409.2329

Zhang, C., & Patras, P. (2018). Long-Term Mobile Traffic Forecasting Using Deep Spatio-Temporal Neural Networks. Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, 231–240. https://doi.org/10.1145/3209582.3209606

Zhao, Z., Chen, W., Wu, X., Chen, P. C. Y., & Liu, J. (2017). LSTM network: A deep learning approach for short-term traffic forecast. IET Intelligent Transport Systems, 11(2), 68–75. https://doi.org/10.1049/iet-its.2016.0208