UPPSALA
UNIVERSITET

IT 21 123
Examensarbete 30 hp
Januari 2022

Probabilistic Forecast of Time Series with Transformers and Normalizing Flows

Seema Negi

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Probabilistic Forecast of Time Series with Transformers and Normalizing Flows

Seema Negi

Today, machine learning has many applications and is used in different fields and industries. One of its applications is demand forecasting. Future demands play a vital role in any industry. It can help the organization from planning to making the stock levels available to the customer when required.

This thesis aims to understand normalizing flows and do multivariate probabilistic forecasting using normalizing flows conditioned on autoregressive models like GRUs and Transformers. The normalizing flows can be effective in modelling complex distributions. The built models are tested on different available time-series datasets and H&Ms data.The H&Ms dataset consists of the weekly sales of various articles. These articles are supplied to the stores from different warehouses. The time series in fashion retail is often ephemeral; hence shorted time steps are there. Similar is the case with the H&Ms data. The data acquired is first preprocessed and then passed to the model. The results are obtained and compared using different evaluation metrics.

Handledare: Marco Trincavelli
Ämnesgranskare: Hans Rosth
Examinator: Mats Daniels
IT 21 123
Tryckt av: Reprocentralen ITC

Acknowledgements

I would like to express my deepest gratitude to H&M AB for providing me with the opportunity of doing my master's thesis project at the organization. I would like to give a special thanks to my Supervisor Marco Trincavelli and Co-supervisor Jacopo Parvizi. Without their constant support and thoughtful encouragement, the thesis would never have taken such shape.

I would also like to thank my reviewer Hans Rosth who kindly participated in this thesis project by giving his valuable time and feedback. I would like to thank him for being very supportive and guiding me throughout the project.

Finally, I wish to thank my family and friends, especially my parents, for their constant support, love and encouragement throughout my studies.

Contents

List of Figures

List of Tables

Acronyms

ARIMA Autoregressive Integrated Moving Average. 5

ARMA Autoregressive Moving Average. 5

CRPS Continuous Ranked Probability Score. 4, 36

GELU Gaussian Error Linear Units. 35

GPU Graphics Processing Units. 16

GRU Gated Recurrent Unit. 3, 8

MADE Masked Autoencoder for Distribution Estimation. 15

MAF Masked Autoregressive Flow. 15

MSE Mean Square Error. 4, 22, 37

ND Normalized Deviation. 4, 37

NRMSE Normalized Root Mean Square Error. 4, 37

NVP Non Volume Preserving. 14

PDF Probability Density Function. 15

RMSE Root Mean Square Error. 37

RNN Recurrent Neural Network. 3, 5, 7, 8, 10

SGD Stochastic Gradient Descent. 32

SKU Stock Keeping Unit. 44

1 Introduction

Future demands play a very significant role in production, planning and inventory management. Accurate demand forecasting is vital in any industry. It allows the business to set correct stock/inventory levels, price the products correctly, and precisely understand particular or related articles patterns and sales. Demand forecasting has gained a great deal of attention from both researchers and practitioners. Different industries are trying to fill the gap between the demands and supplies. From minimizing inventory costs to optimizing cash flow, accurate demand forecasting can result in one of the most essential factor for any industry, "Customers' satisfaction." Meeting customers expectations for product availability is a primary and prominent aspect for any retailer. Product availability does not mean making products/articles available all the time but certainly having them available when the customers need them. In this kind of situation where retailers want to offer their customers the right product at the right time and the right place, accurate demand forecasting can play a significant role. At the organizational level, forecasts of demands can be an essential inputs to many decision-making activities in various operative areas such as marketing, production/purchasing, sales as well as finance and accounting. Barksdale and Hilliard (1975) [2] examined the relationship between the sales and retail stocks at the aggregate level. They found that successful inventory management depends to a large extent on the accurate forecasting of retail sales or demands. Agrawal and Schorling (1996) [1] also showed that accurate demand forecasting plays a critical role in profitable retail operations and poor forecasts can result in too much or too little stocks that directly affect revenue and the competitive position of the retail business.

The fashion retail industry is one of the fastest growing industries where demand forecasting is crucial but not easy to implement. Following the fashion trends and market responses, fashion products have a highly unpredictable demand. Many factors can affect the demand and shift the demand curve. These factors can involve social factors including cultures, lifestyle, norms, demographics and population changes. Economic factors like economic uncertainty can also contribute to the unpredictability of demands. There can be some unaccounted and unprecedented factors that can alter the forecasts. The COVID-19 pandemic is one of the most prominent and prevailing examples of unprecedented factors that can drastically affect the forecasts. Due to the lockdowns and several restrictions globally, there has

been a radical shift in sales. Song-Yi Youn, Jung Eun Lee and Jung Ha-Brookshire [29] showed that consumer assessments of perceived severity and altruistic fear of COVID-19 and self-efficacy of channel switching extended their beliefs (i.e., attitude, control, perceived behaviour, subjective norm) and intentions to switch shopping channels to online. These demand fluctuations can highly impact the stock levels in warehouses, creating an imbalance. As an outgrowth of high demand variability, inventory batches allotted to different regions will repeatedly either be too small or too large during the selling season, meaning that retailers might end up with either stock outs or unwanted excess inventory for specific products.

Demand forecasting is a time series forecasting problem where the future values are predicted based on historical data. When it comes to the fashion retail industry, accurate demand forecasts are not easy to predict. Today, fashion technology is growing faster than ever, and with this, styling and fashion keep on changing. There are numerous underlying factors accountable for this fast transition over the past years. These factors include globalization, increase in customer requirements and evolving technology. The products in the fashion industry are often ephemeral. The period in which the products are saleable is sometimes very short and seasonal, measured in months or even weeks. The products which might be very popular at a particular time might go out of fashion after some time. The unpredictable nature of the fast-fashion market and the complexity of a large number of products in different sizes being sold at multiple locations make it challenging to implement a straightforward algorithm or strategy. Along with this, the sale of a particular article can be affected by some other articles also. Therefore, it is also essential to consider and take into account the effect of interacting articles, which can lead to cannibalization effects in the case of article competition. Cannibalization can be defined as a reduction in sales revenue, sales volume, or market share of an article due to introducing a new article by the same producer. Many approaches and research have been proposed in the literature over the past few decades to understand these patterns and predict demand accurately. From traditional statistical models to neural networks, research is ongoing.

1.1 Purpose

The purpose of this masters thesis is to forecast the demand and understand the underlying factors for one of the leading names in the apparel industry, Hennes & Mauritz (H&M). H&M is a Sweden based multinational clothing retail company founded in 1947. In 1947, the womenswear store Hennes opened in Västerås, Sweden. Today with a focus on omnichannel sales, H&M Group is a global fashion and design company with 53 online markets and stores in 74 markets. (H&M Group 2021)[13]

To serve the demand in stores correctly, H&M needs to ensure that warehouses stock matches both the store and online demand. Therefore, this thesis project aims to accurately predict the demand in order to level out imbalances. Moreover, it is desirable to predict a measure of uncertainty associated with the prediction in the form of a probabilistic forecast. It can provide additional information to the algorithms that solve downstream tasks. In particular, the most relevant downstream task is lateral transshipments within an inventory system. Lateral transshipments within an inventory system are stock movements between locations of the same echelon. Therefore, to decide whether to move an article from a warehouse to another or not, the results can be useful. In case the move is decided, one can determine how many pieces to move. Also, based on the forecasts, one can determine if the particular article in the warehouse is overstocked or understocked.

1.2 Method

To achieve the stated purpose, a Multivariate Probabilistic Forecasting Model is build using modern Deep Learning Architectures to predict the demand accurately. The multi-variate temporal dynamics of the underlying time series are modelled via an autoregressive deep learning model, where a conditioned normalizing flow represents the data distribution. The performance of different models is compared, and results are analyzed. Along with the H&Ms dataset, the algorithm is applied to some other real-world datasets also.

1.3 Outline of the Thesis

This chapter gives a brief idea about the thesis topic, goal and purpose behind it. Chapter 2, Background section provides an overview of the concepts used in the model. In Chapter 3, the already implemented work in the field and algorithms related to it are discussed. The used datasets, tools and technologies are described in Chapter 4. Chapter 5, Methodologies provides the pipeline from building the models to evaluation of the model. Results on the different datasets using different implemented models are presented in Chapter 6. An overall summary and conclusion of the project is provided in Chapter 7.

2 Background

2.1 Time Series Forecasting

Time series forecasting is the use of a model to predict future values based on previously observed values. Time series forecasting has various applications in fields like economy, weather, stock price, and retail sales. Classical time series forecasting methods rely on the Autoregressive Moving Average (ARMA) method and its variants like Autoregressive Integrated Moving Average (ARIMA). Apart from the fact that these methods require manual feature engineering, they also suffer from the curse of dimensionality. These methods require frequent re-training and are focused on model interpretability rather than test-set accuracy.

Deep learning models over the last years have shown impressive results over classical methods in many fields like computer vision, speech recognition, natural language processing (NLP), and also time series forecasting.

Point forecasts are useful in many applications but it is also often desirable for the outputs to be probability distributions instead of point forecasts. Point Forecast associates the forecasts with a single expected outcome. Example: Forecast to sell 100 units next month. Probabilistic Forecast, on the other hand, provide uncertainty bounds. It identifies a range of outcomes and the probability of each of those outcomes occurring.

2.2 Recurrent Neural Network

Neural networks also called artificial neural networks (ANNs), are a subset of machine learning and core deep learning algorithms. Their name and structure are inspired by the human brain, mimicking how biological neurons signal to one another. Artificial neural networks (ANNs) comprise node layers containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. Figure 1 shows the schematic of ANN. When there are large number of hidden layers, these ANNs are called deep neural networks.

Feedforward Neural Network is one of the ANNs which moves data in only one direction. It does not contain loops which means connections between the nodes do not form a cycle.

While RNNs are a type of Neural Network architecture that are good at modeling sequential data. The RNNs have recurrent hidden units which help

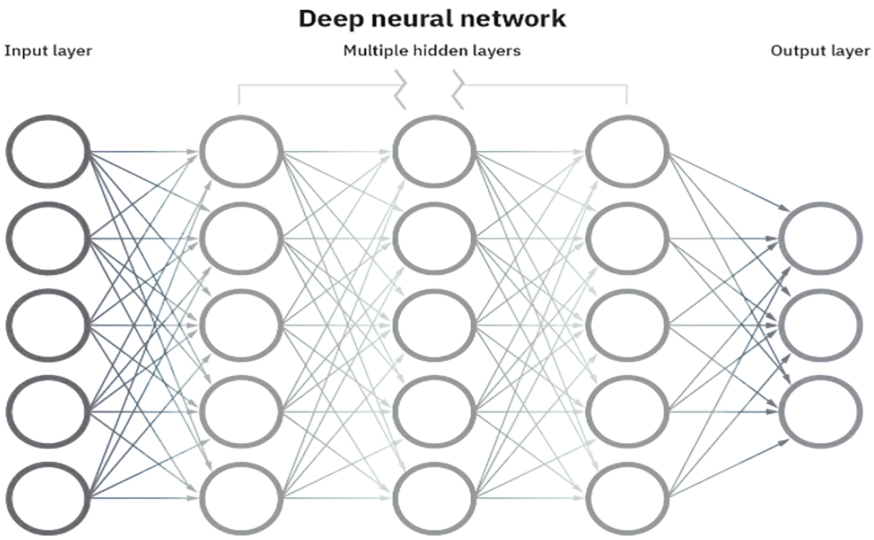


Figure 1: Artificial Neural Network

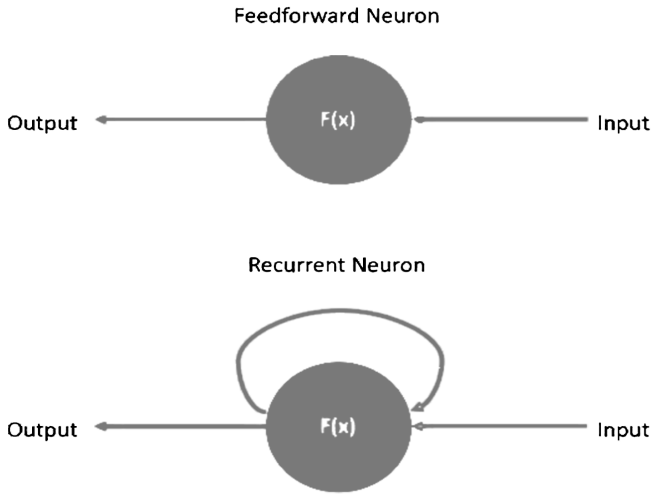


Figure 2: Feedforward Network Neuron and RNN neuron

them model sequential data better. These networks consider the dependency on the prior elements within the sequence. Figure 2 shows the single unit neuron of feedforward and RNN. RNNs use (1) to model the values of their hidden units.

$$h_t = f(h_{t-1}, x_t; \theta), \tag{1}$$

The equation indicates a classic dynamic system driven by an external signal x_t where h_t is the state of the system at step t and θ is a parameter of a transit function f. This means that RNNs are deterministic, non-linear dynamic systems, in contrast to additive exponential smoothing in state space form, which can be represented as linear non-deterministic dynamic systems with a single source of error/innovation (Hyndman et al., 2008). RNNs have the advantage of modelling the sequential nature of time series explicitly, and thus requires a smaller number of parameters that need fitting.

Figure 3 illustrates the RNNs. It can be observed in the figure that at each time point, the network gets input data x_t and the previous state h_t of the RNN and outputs a state for the next timestep h_{t+1}.

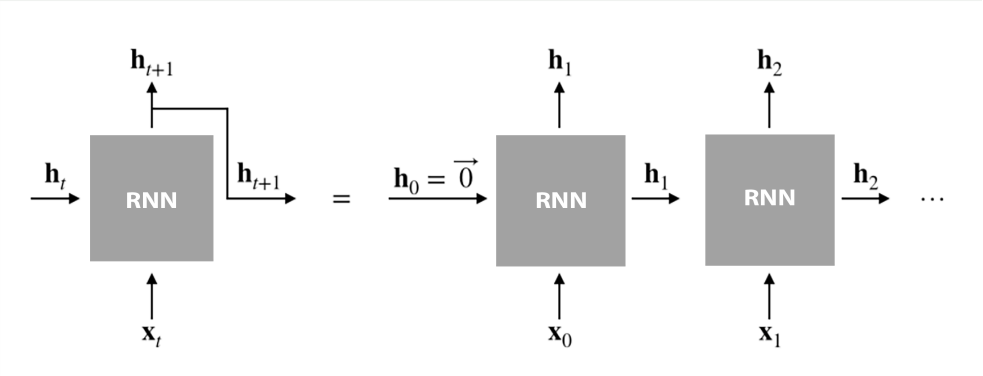


Figure 3: Recurrent Neural Network

RNNs are good for many applications but have a drawback. RNNs generally suffer from exploding and vanishing gradients problems. The size of the gradient is represented as the slope of the loss function along the error curve. When the gradient value is too small, it continues to become smaller, updating the weight parameters until they become insignificant like 0. When this situation occurs, the algorithm is no longer learning. Exploding gradients happen when the gradient is too large, which can result in an unstable model. While in the case of exploding gradients, the model weights grow too

large, ultimately represented as 'NaN'. One solution to these problems is to decrease the number of hidden layers within the neural network, reducing some of the complexity in the RNN model.

2.3 Gated Recurrent Unit

GRUs [7] [6] are considered as the improved version of standard RNNs. As discussed in RNNs subsection, RNNs generally face the problem of vanishing gradients. Compared to RNNs, GRUs can learn long sequences much better. GRUs learn long-term dependencies using mechanisms called "gates". These gates are known as update gate and reset gate, which are different tensor operations that can learn what information to add or remove to the hidden state. The update gate helps the model determine how much of the past information from previous time steps must be passed along to the future. The reset gate is used by the model to decide how much of the past information to forget.

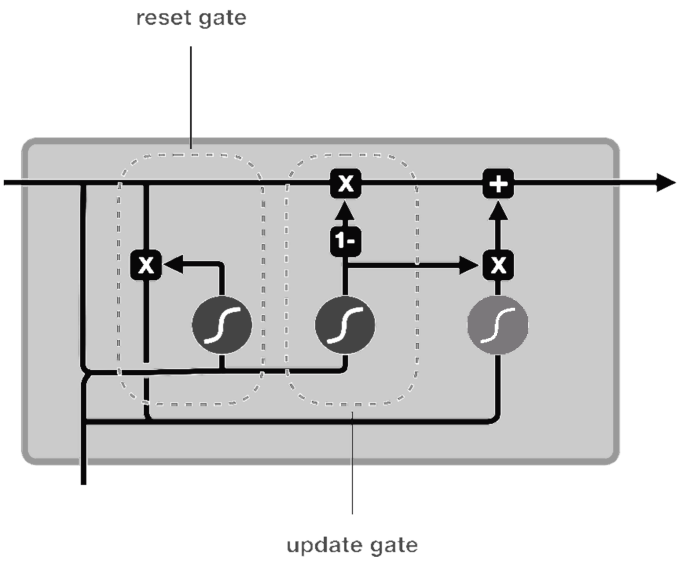


Figure 4: GRU cell and its gates

Figure 4 shows the GRU cell and its gates and Figure 5 shows the meaning of different symbols within the GRU cell.

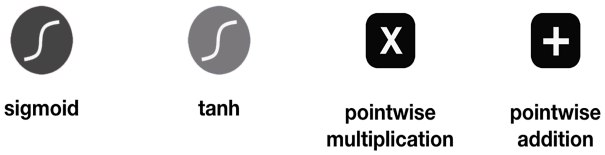


Figure 5: Operations within GRU units

The tanh activation helps in regulating the values flowing through the network. The tanh function takes the real value as input values and output in the range of -1 and 1. When vectors flow through a neural network, it undergoes many transformations because of various math operations. A value multiplied by a big number can cause other values to seem insignificant. Thus, a tanh function ensures that the values stay between -1 and 1, further regulating the output of the neural network.

A sigmoid activation is alike to the tanh activation. Instead of outputting values between -1 and 1, it outputs values in the range of 0 and 1. This helps in updating or forgetting the data. As any number getting multiplied by 0 is 0, causing values to disappear or be "forgotten". Any number multiplied by 1 is the same value therefore that value stays the same or is "kept". This way, the network can learn what data is important or not.

2.4 Encoder Decoder Architecture

As the name suggests, Encoder Decoder Architecture consists of an encoder network which encodes the input sequence and decoder on the other end decodes it. Encoder Decoder Architecture is widely used in sequence to sequence translation.The encoder takes a variable-length sequence as the input and transforms it into a state with a fixed shape. The decoder maps the encoded state of a fixed shape to a variable-length sequence. Figure 6 shows an encoder-decoder architecture.

2.5 Attention

Attention mechanisms have become an essential part of effective sequence modelling and transduction models in various tasks, allowing the modelling of dependencies without consideration to their distance in the input or output sequences [16] [20]. In some cases, such attention mechanisms are used in conjunction with a recurrent network.

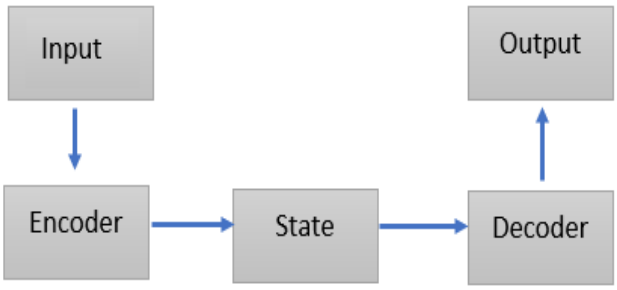


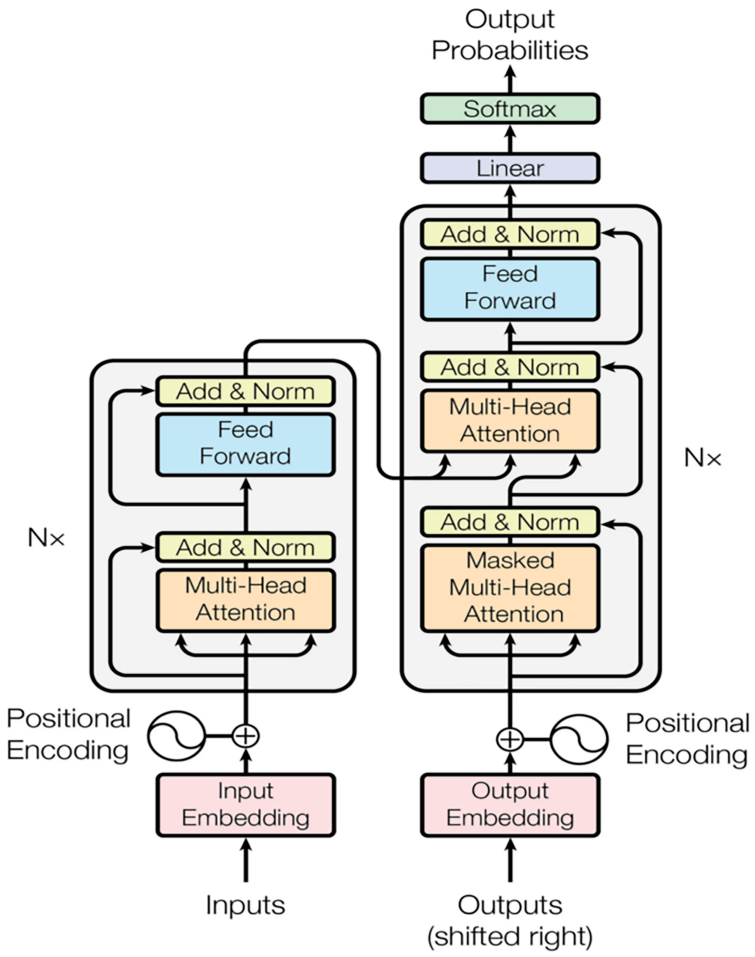
Figure 6: Encoder Decoder Architecture

The attention mechanism looks at an input sequence and decides which other parts of the sequence are essential and of more importance at each step. It is a mechanism targeted to compute the alignment score between two sources through a compatibility function that aims to measure the dependency between these two sources [25]. The softmax function is responsible for transforming these scores into probability distribution by normalising all given iterations of the first source entity. Hence more significant distribution would mean that the given token of the source contributes more important information to the second given entity. An example would be how much contribution a given feature makes for a given time step.

In the case of the encoder-decoder model, instead of paying attention to the last state of the encoder as is generally done with RNNs, in each step of the decoder, all the states of the encoder are looked at, being able to access information about all the elements of the input sequence. Attention extracts information from the whole sequence as a weighted sum of all the past encoder states. This allows the decoder to assign higher weight or importance to a specific input element for each output element. It allows learning in every step to focus on the right element of the input to predict the next output element. However, this approach continues to have a significant limitation. Each sequence must be treated as one element at a time. Both the encoder and the decoder have to wait till the completion of the $t-1$ steps to process the $t-th$ step. So when dealing with colossal sequences, it is very time consuming and computationally inefficient.

2.6 Transformer

Transformers consist of a model architecture abstaining recurrence and instead relying solely on an attention mechanism to model global dependencies among input and output. The Transformer permits for significantly more parallelization. The Transformer is the first transduction model relying totally on self-attention to compute its input and output representations without using sequence-aligned RNNs or convolution [27]. Figure 7 shows the architecture of the transformer model.



Source: Attention Is All You Need [28]

Figure 7: The Transformer Model architecture

In Figure 7, the architecture consists of an encoder and decoder model. Both models contain a core block of "an attention and a feed-forward network" repeated N times that can be stacked on top of each other multiple times (represented by Nx in the figure). In the shown architecture, the attention layer is multi-head attention. As the model is not recurrent, the transformer can not learn inherently temporal dependencies. To oppose that, positional encoding is used, which provides positional information for each token. This information is added to the input.

The multi-head self-attention layer transforms the given input X into H distinct query matrices $Q_h = XW_h^Q$, key matrices $K_h = XW_h^K$ and value matrices $V_h = XW_h^V$, where W_h^V, W_h^Q and W_h^K are the learnable parameters. After these projections, a sequence of vector outputs is calculated by scaled dot-product via :

$$O_h = Attention(Q_h, K_h, V_h) = softmax(\frac{Q_h K_h^T}{\sqrt{d_K}} \cdot M)V_h$$

where mask M is applied to filter the part of the known output sequence / future values by setting its upper triangular elements to -∞.
d_K represents the dimension of W_h^K matrices and softmax denotes a row-wise softmax normalization function.

After that all the H outputs O_h are concatenated and linearly projected again.

To understand transformers better, let us take an example of sequence to sequence translation where we want to translate one sentence from English to Swedish.

English - *I live in Sweden.*
Swedish - *Jag bor i Sverige.*

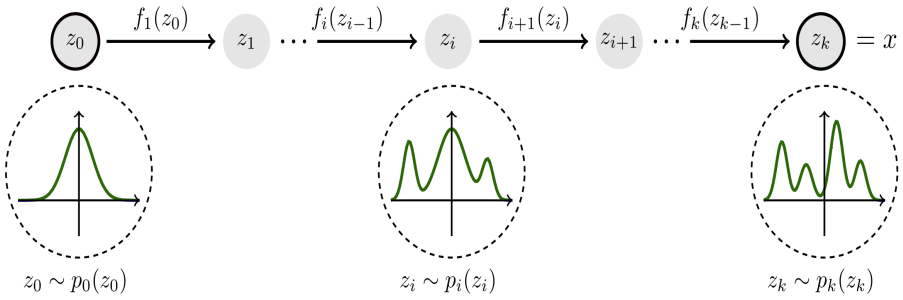
As there is no concept of the time step for the input in the transformer, we pass all the words of the sentence simultaneously and determine the word embeddings. As the model can not understand words, the idea of word embedding is to map every word to a point in space (embedding space) where similar words in meaning are closer to each other. Therefore, embedding space contains the mapping of a word to a vector. However, these words in different sentences can have different meanings. Therefore, positional encoders are used, which contains a vector with information on the distance between words and sentences. The word vectors with positional information are passed to the encoder block, consisting of a multi-headed attention

layer and a feed-forward layer. Attention involves answering what part of the input should be focussed on when translating from English to Swedish. For every word in the sentence, an attention vector is generated, capturing the contextual relationship between words in the sentence. The next layer is the feed-forward network, which transforms the attention vectors in the form acceptable by the next encoder or decoder block. The output, i.e. Swedish translation, is also fed to the decoder during the training phase of translating English to Swedish. Like the encoder, the decoder also receives input as embedding and uses a positional encoder to get the notion of context of the word in the sentence. This vector is passed to the decoder block, which has three main components, two of which are similar to the encoder block. The self-attention block generates attention vectors for every word in the Swedish sentence. These attention vectors and vectors from the encoder are then passed into another attention block. Now in this block, we have one vector from every word in the English and Swedish sentences. This attention block determines how related each word vector is with respect to each other. This is where the main English to Swedish word mapping happens. The output of this block is attention vectors for every word in English and Swedish sentences, each vector representing the relationship with other words in both languages. Next, each attention vector is passed to a feed-forward unit. Then, a linear layer expands the dimensions into the number of words in the Swedish language. Further, the softmax layer transforms it into a probability distribution. The final word is the word corresponding to the highest probability overall. The decoder predicts the next word, and this step is executed over multiple time steps until the end of the sentence token is generated. As seen in the Figure, the attention block in the decoder is called the Masked attention block. While generating the next Swedish word, we can use all English words but only the previous words of the Swedish sentence. If all the Swedish words are used, there would be no learning. So while performing parallelization with matrix operation, the words appearing later in Swedish sentences are masked by transforming them into zeros. So the attention network can not use them in the next block.

2.7 Normalizing Flows

Good density estimation is important for various machine learning problems, but it is hard as well. As for the backward propagation in deep learning models, the embedded probability distribution $p(z \mid x)$ is expected to be simple to calculate the derivative efficiently and easily. This is the reason why Gaussian distribution is often used in latent variable generative models, but most real-world distributions are much more complex than Gaussian distributions.

Normalizing Flow models can be used for modelling complex distributions. A normalizing flow transforms a simple distribution into a complex one by applying a sequence of invertible/bijective transformation functions.



Source: Figure inspired by Lilian Weng

Figure 8: Illustration of a normalizing flow model

Figure 8 shows an illustration of a normalizing flow model, transforming a simple distribution $p_0(z_0)$ to a complex distribution $p_k(z_k)$. Normalizing flows are mappings from R^D to R^D such that densities pX on the input space $X = R^D$ are transformed into some simple distribution p_z on the space $Z = R^D$. This mapping $f : X \mapsto Z$, is composed of a sequence of bijections or invertible functions. Due to the change of variables formula $pX(x)$ can be expressed by

$$pX(x) = pZ(z)|\det(\partial f(x)/\partial x) \tag{2}$$

where $(\partial f(x)/\partial x)$ is the Jacobian of f at x. Normalizing flows have the property that the inverse $x = f^{-1}(z)$ is easy to evaluate and computing the Jacobian determinant takes O(D) time.

There are different type of normalizing flows model. One of the model is Real Non Volume Preserving (NVP). Real NVP introduced a bijection known as coupling layers. A flexible and tractable bijective function is built

by stacking a sequence of simple bijections. In every simple bijection, a part of the input vector is updated using a simple function that is easy and simple to invert and depends on the remainder of the input vector in some complex way. These simple bijections are known as an affine coupling layer. Given a D dimensional input x and $d < D$,the transformation copies the first d elements and then scales and shifts the remaining $D - d$ elements. The output y of an affine coupling layer follows the equations

$$y_{1:d} = x_{1:d},$$
$$y_{d+1:D} = x_{d+1:D} \odot exp(s(x_{1:d})) + t(x_{1:d}),$$

where $s()$ and $t()$ stand for scale and translation functions from $R^d \rightarrow R^{D-d}$ given by neural networks , and $\odot$ is the Hadamard product or element-wise product.

To model a non linear or complex density map, a number of coupling layers are stacked all together. By the change of variables formula discussed in 8.6, given a data point, the Probability Density Function (PDF) of the flow can be written as

$$logpX(x) = logpZ(z) + log|det(\partial z/\partial x)|$$
$$logpX(x) = logpZ(z) + \sum_{i=1}^{K} log|det(\partial y_i/\partial y_{i-1})|$$

The Jacobian for the Real NVP is a block-triangular matrix and thus the log-determinant simply becomes

$$log|det(\partial y_i/\partial y_{i-1})| = sum(log|diag(exp(s(y_{i-1}))|),$$

where sum() is the sum over all the vector elements, log() is the element-wise logarithm and diag() is the diagonal of the Jacobian.

The another flow model is Masked Autoregressive Flow (MAF). MAF uses the Masked Autoencoder for Distribution Estimation (MADE) [11] with Gaussian conditionals as the building block. MADE takes x as input and outputs μ_i and α_i for all i with a single forward pass. The weight matrices of MADE are multiplied with suitably constructed binary masks to enforce the autoregressive property. The use of masking allows transforming from data x to random numbers u and hence calculating the $p(x)$ in one forward pass through the flow. It eliminates the need for sequential recursion.The

transformation layer in MAF is built as an autoregressive neural network. It takes some input $x \in R^D$ and outputs $y = y^1, y^2..., y^D$ with the requirement that the transformation is bijective, and a given output y^i can not depend on input with dimensions less than or equals to i. The Jacobian determinant of this transformation is tractable as the Jacobian is triangular. The use of MADE allows density evaluations without the sequential loop and thus makes MAF fast to evaluate and train on a parallel computing architectures like Graphics Processing Units (GPU)s.

Inference

$$x \sim p\hat{X}$$
$$z = f(x)$$

Generation

$$z \sim pZ$$
$$x = f^{-1}(z)$$

Figure 9 shows the density estimation on moon dataset [3] using RealNVP. It is a toy dataset provided by sklearn which draw two interleaving half circles.

In the Figure 9, in the upper left, the function $f(x)$ maps samples x from the data distribution into approximate samples z from the latent distribution in the upper right. This corresponds to an exact inference of the latent state given the data. The inverse function, $f^{-1}(z)$, maps the samples z from the latent distribution in the lower-left into approximate samples x from the data distribution in the lower-right. It can be seen that it corresponds to the exact generation of samples from the model.

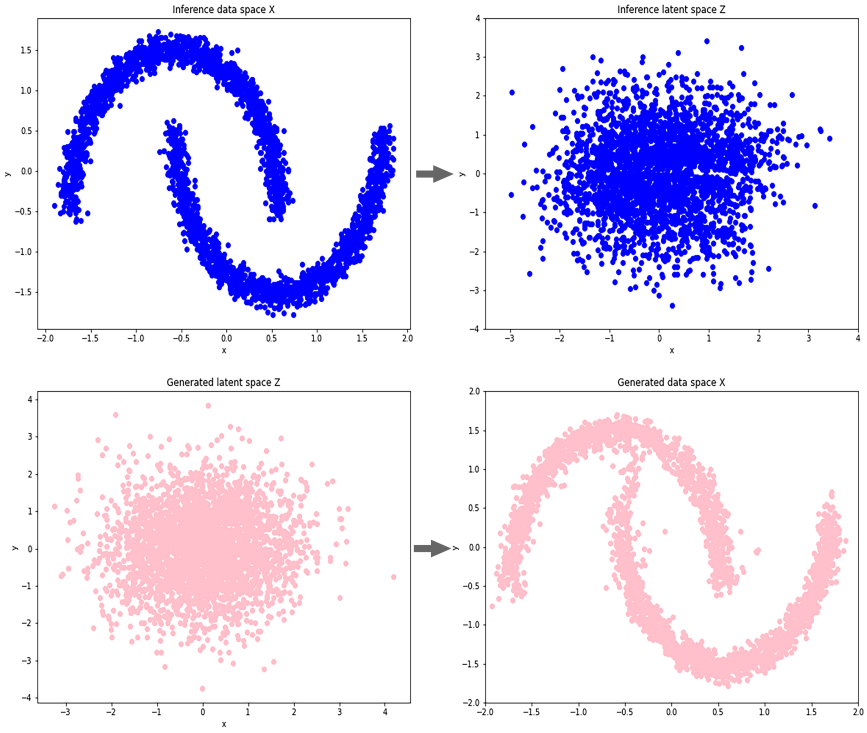


Figure 9: Density estimation using Real NVP on Moon Dataset

3 Related Work

In real-world forecasting problems, particularly within the demand forecasting domain, one is regularly confronted with highly lumpy, discontinuous or intermittent data. For example, Intermittent or sporadic demand can result in situations where demands can be highly variable in size. These situations violate the core assumptions of many classical techniques, such as stationarity, Gaussianity or homoscedasticity of the time series.

Intermittent demand can be treated by using techniques ranging from classical methods [8] to neural networks [14]. The authors of *"Lumpy demand forecasting using neural networks"* [14] compares the performance of forecasts using a neural network to those using three traditional/classical time series methods such as single exponential smoothing, Croston's method, and the Syntetos–Boylan approximation. It was observed that neural networks generally perform better than traditional methods in forecasting lumpy demands. There are also other approaches which incorporate more suitable likelihood functions, such as the zero-inflated Poisson distribution, the negative binomial distribution [26] (Snyder, Ord, Beaumont, 2012), a combination of both [4] (Chapados, 2014), or a tailored multi-stage likelihood [24] (Seeger, Salinas, Flunkert, 2016).

DeepAR [9] provides a forecasting method based on autoregressive recurrent neural networks, which makes use of historical data of all time series in the dataset to learn a global model. It provides an approach for the demand forecasting problem by incorporating appropriate and a wide range of likelihoods and combining them with non-linear data transformation techniques, as learned by a deep neural network. The authors of DeepAR also provide the full probability distribution of the forecasts, which can help in decision making in downstream tasks.

To incorporate the dependency between multiple time series, there is one model 'DeepVar.'[23] It combines an RNN-based time series model with a Gaussian copula process output model with a low-rank covariance structure. This reduces the computational complexity and handles non-Gaussian marginal distributions very well, allowing the modelling of time-varying correlations of thousands of time series.

Sharing information across time series is key to improving forecast accuracy. However, this can be not easy to accomplish in practice due to the non-homogeneous nature of the data. A well-known approach to sharing information across time series is to use clustering techniques such as

k-means clustering.[19] It is used to compute seasonality indices, which are then combined with classic forecasting models. There are many instances where unsupervised learning techniques can be used as a pre-processing step. These effects need to be handled in practical applications, leading to complex pipelines that are difficult to tune and maintain. The complexity of such pipelines is likely to increase when one needs to address specific subproblems, such as forecasts for new products/articles. On the other hand, deep neural network models require a limited number of standardized data preprocessing, after which the forecasting problem is solved by learning an end-to-end model. In specific, data-processing is included in the model and optimized jointly towards the goal of producing the best possible forecast. The deep-learning forecasting pipelines rely mostly on what the model can learn from the data while the traditional pipelines rely heavily on heuristics such as manual covariate design, preprocessing and expert-designed components.

The idea of thesis work is to model the multi-variate temporal dynamics of time series using an autoregressive deep learning model where distribution of data is represented by a conditioned normalizing flow. The combination benefits from the power of both autoregressive models and normalizing flows. The autoregressive models allow good performance in extrapolation into the future, with the flows allowing to model the high-dimensional data distribution while remaining computationally tractable.

There are models where work is done related to normalizing flows. The PixelSNAIL [5] method introduces a new generative model architecture that models the joint probability as a product of conditional distributions and combines causal convolutions with self attention. (Vaswani et al., 2017). Self attention is used to capture long-term temporal dependencies.

There are models like Videoflow [17] which provides a conditional flow-based model for stochastic video generation.

4 Preliminaries

4.1 Dataset

To train and test the models, following datasets are used:

- **Electricity Dataset** - Electricity dataset contains hourly time series of the electricity consumptions of 370 customers. The frequency of the dataset is hourly. The prediction length is 24 which means the data is predicted for next 24 hours on the basis of given past data.
- **Solar Dataset** - The frequency of the dataset is hourly and prediction length is 24.
- **Traffic Dataset** - Traffic dataset contains the hourly occupancy rates, between 0 and 1, of 963 car lanes of San Francisco bay area freeways. The frequency of the dataset is Hourly and prediction length is 24.
- **H&M Dataset** - The dataset is hosted in H&M data lakes.The dataset consists of many articles from several warehouses and countries. The frequency of data is weekly and prediction length is 6.

4.2 Data Preprocessing

Data preprocessing plays an important role in the machine learning. The quality of data directly influences the model's ability to learn; therefore, data must be preprocessed before feeding it into the model. Apart from the H&M dataset, other datasets were already clean and preprocessed. Therefore, no further preprocessing has been done on those datasets. The following steps are performed on H&M dataset:

- The data acquired is from 28th April 2019 to 27th December 2020. Some of the articles in data are seasonal, which means the sale is not available for the entire time period. The time stamps for which sale is not available are imputed with 0 which indicates no sale.
- The data is missing from 5th January 2020 to 19th April 2020 for all warehouses and articles. These values are imputed with 0.

- Some of the values for the Sales are negative. The negative number represents the case when the number of articles returned in the stores are greater in number of articles sold. These entries are dropped from the data.
- Some of the articles contain values for one or two months only. These articles are dropped. The articles for which 80 percentage or above is 'NaN' are dropped.

4.3 Tools and Technologies

- **Python** - Python is a high-level programming language developed by Guido van Rossum, and released in 1991. It is used for scripting, web development, game development and machine learning. It has a very large array of libraries which makes it suitable for performing big data, complex mathematical operations.
- **PyTorch** - PyTorch is a machine learning library for python developed by Facebook's AI Research (FAIR) lab for machine learning, computer vision and natural language processing. PyTorch allows the tensor operators to be performed on the CUDA capable Nvidia graphic cards.
- **GluonTS** - Gluonts is a toolkit for modeling the probabilistic time series. It provides utilities for loading, training and modeling over the time series data set.
- **Pytorch-ts** - PyTorchTS is a PyTorch Probabilistic Time Series forecasting framework which provides state of the art PyTorch time series models by utilizing GluonTS as its back-end API and for loading, transforming and back-testing time series data sets.
- **Matplotlib** - Matplotlib is plotting library for developing static/dynamic interactive, animated plots. It provides an array of object oriented API's for building several plots such as line, bar, scatter and so on.
- **Azure Databricks** - Azure Databricks is a Apache Spark based environment hosted on Microsoft's Azure cloud platform. It is a data analytical platform for experimenting, modeling, training and feature serving the AI workloads.

5 Methodologies

5.1 Building the Model

The following models are built using the concepts discussed in the Background section and used for analysis, training and prediction.

- Deep Auto Regressive Model
- RNN Conditioned Real NVP model
- RNN Conditioned Real MAF model
- Transformer Conditioned Real NVP model
- Transformer Conditioned MAF model

The general idea is to build a model which can make use of normalizing flows and plot a complex density distribution.

In case of the point forecasting, the method takes the hidden state and outputs a single point estimated value of the next time step. In time series regression problems, the model is often trained by minimizing the Mean Square Error (MSE) of the predicted with respect to the observed or ground truth value.

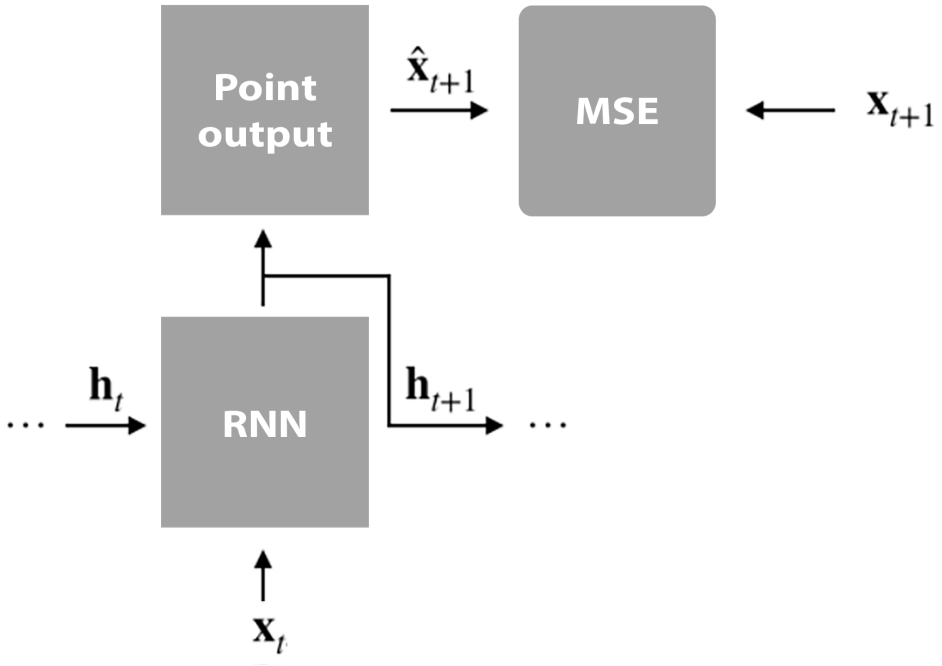


Figure 10: Point Forecast with RNN

As it can be seen in the Figure 10, when the model is used for inference/prediction, RNNs are first to run over the historical data available to warm it up. This state of the RNN is then continued into prediction times by feeding the outputs of the model at each time step as input to the model for the next time step, in addition to the hidden state.

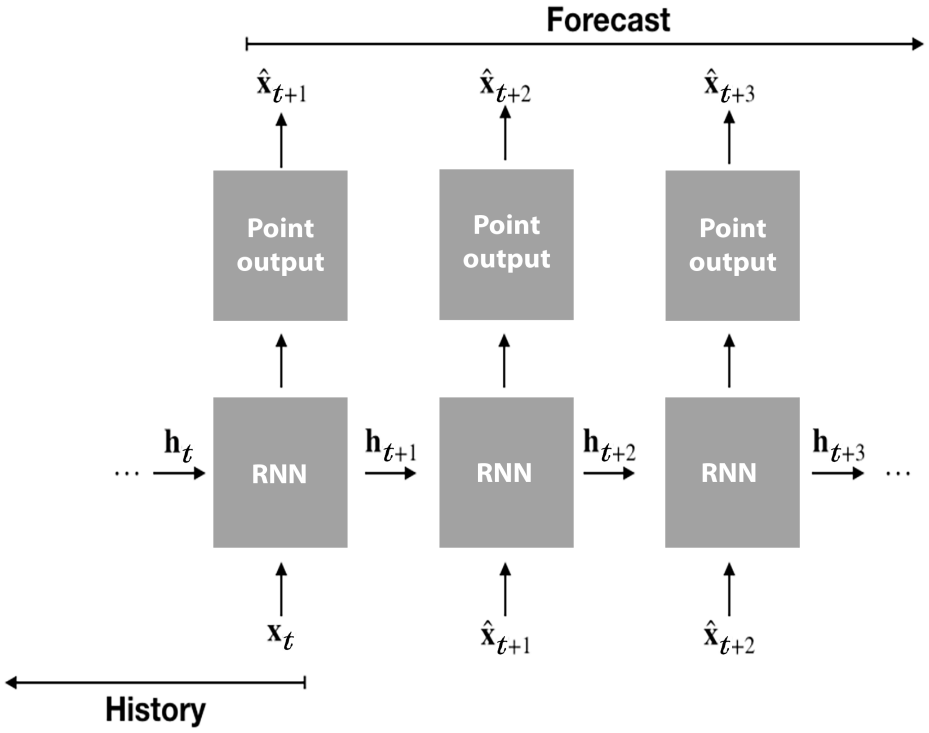


Figure 11: Point Forecast Training with RNN

When we want to predict the probabilistic forecast instead of point forecasts, Figure 10 can be transformed to Figure 12. The output of the model is a distribution output. The model is trained with negative log probability. Instead of predicting the point forecasts, parameters of a distribution can be predicted like mean and variance for the Gaussian distribution.

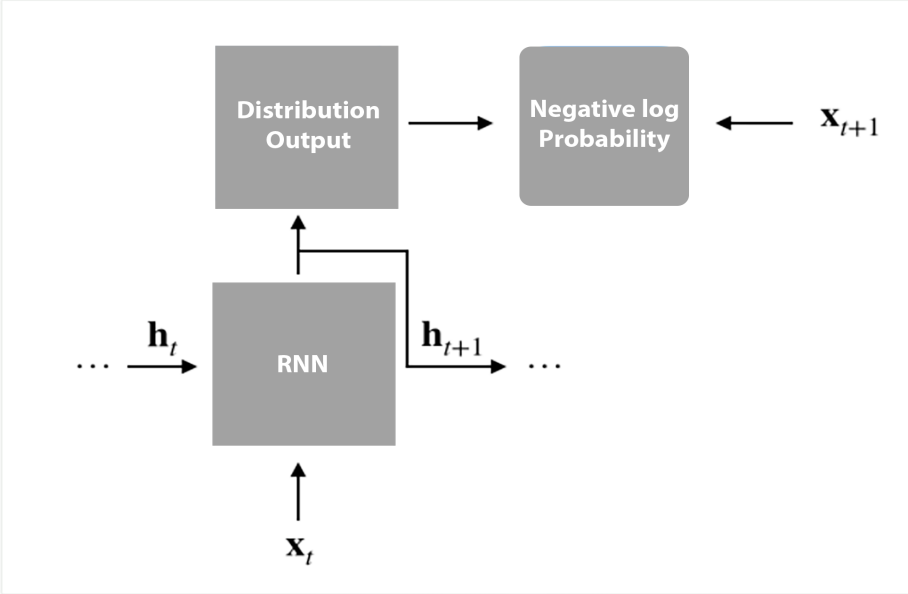


Figure 12: Converting point forecast to probability distribution

Compared to the deterministic point forecast at prediction time, as can be seen in Figure 13 values are sampled from the output distribution for the next step in probabilistic forecasting. These samples are then fed into the RNN to get the new samples for the next time step. This procedure can be repeated to get enough sampled forecasts for the whole prediction interval. These samples can then be used to calculate statistical estimates; for example, any quantiles of interest can be calculated.

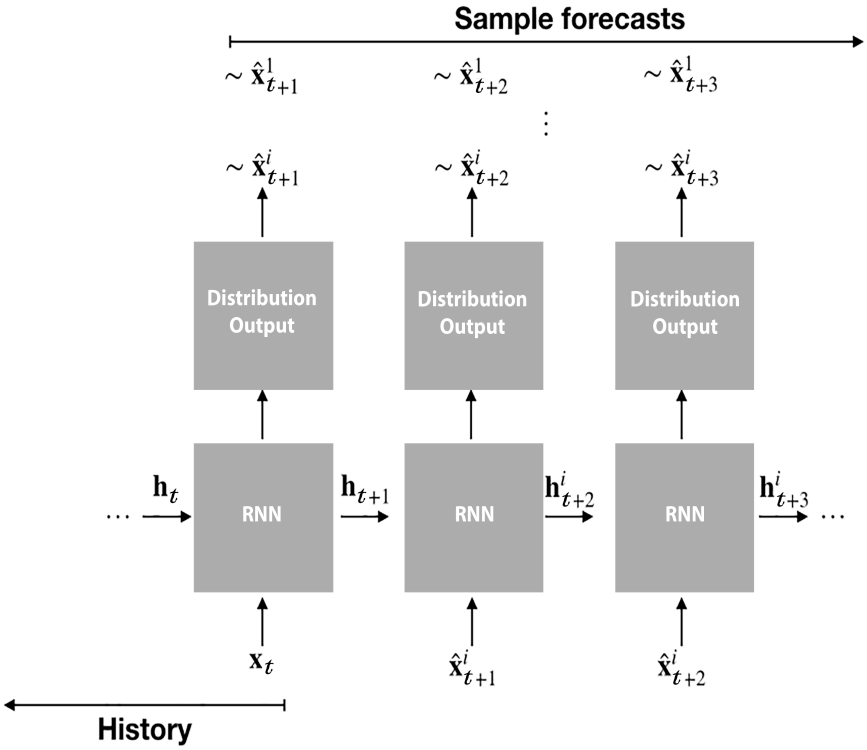


Figure 13: Probabilistic Forecasting with RNNs

5.2 Deep Auto Regressive Model

The DeepAR forecasting method is based on autoregressive recurrent neural networks, which learns a global model from historical data of all time series in the dataset. This model provides probabilistic forecast. The model predicts the parameters of the distribution, for example; mean and variance in case of Gaussian Distribution.

Figure 14 shows the model schematic during training. The network takes the input x_{i_t} and its covariates c_{i_t}. The network further model the conditional probability distribution on the basis of the predicted parameters θ of the distribution.

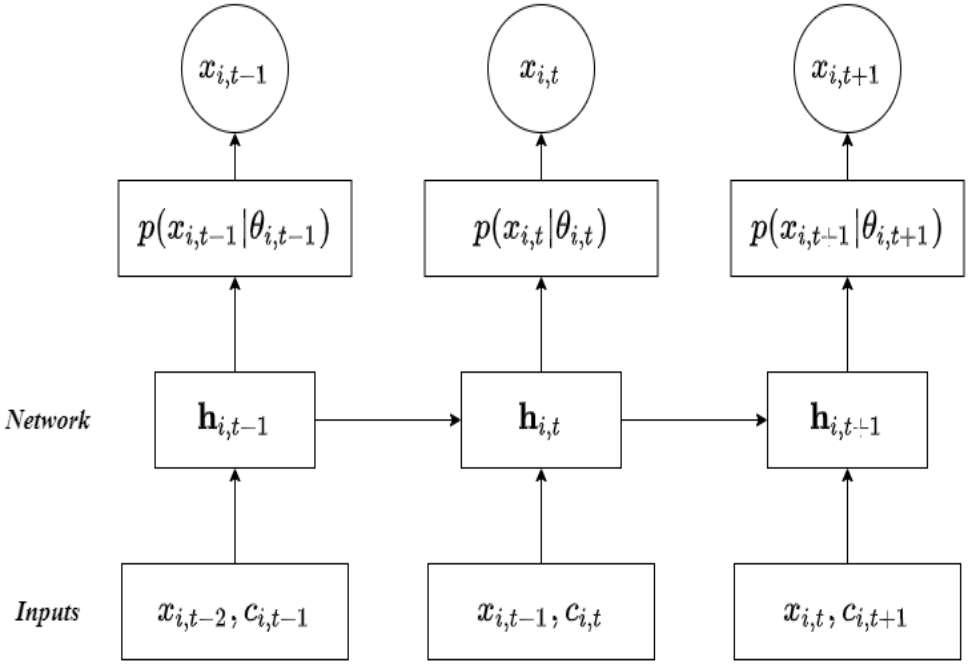


Figure 14: Training with DeepAR

Figure 15 shows the DeepAR network during inference. The model takes the input, which in this case is the sample predicted by the network in previous time step.

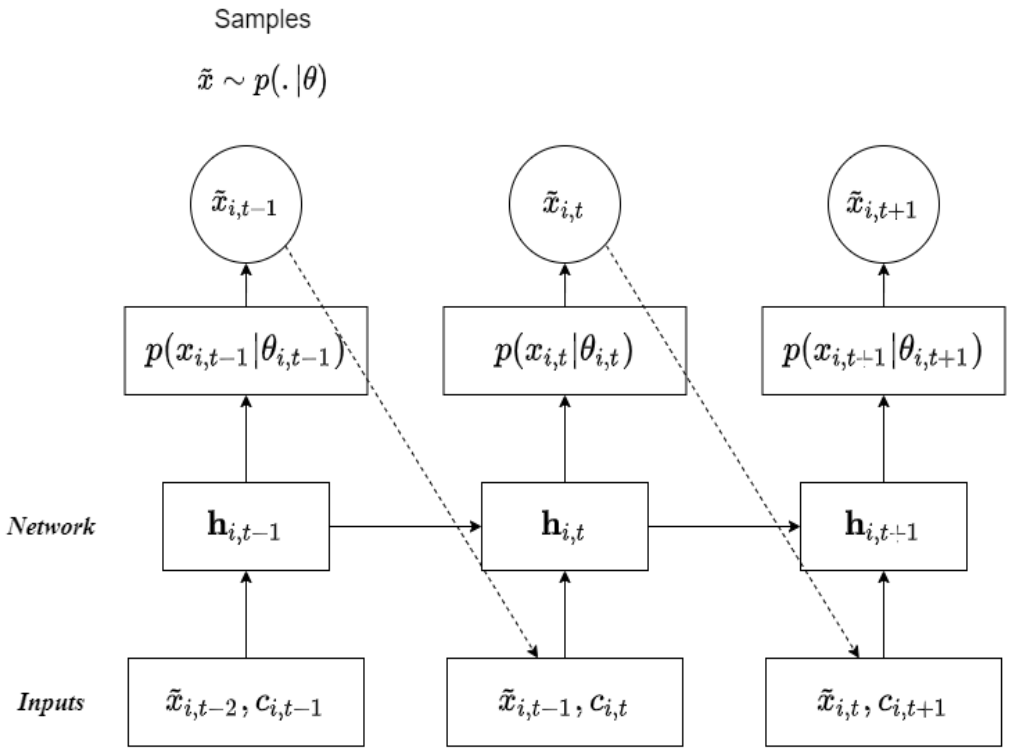
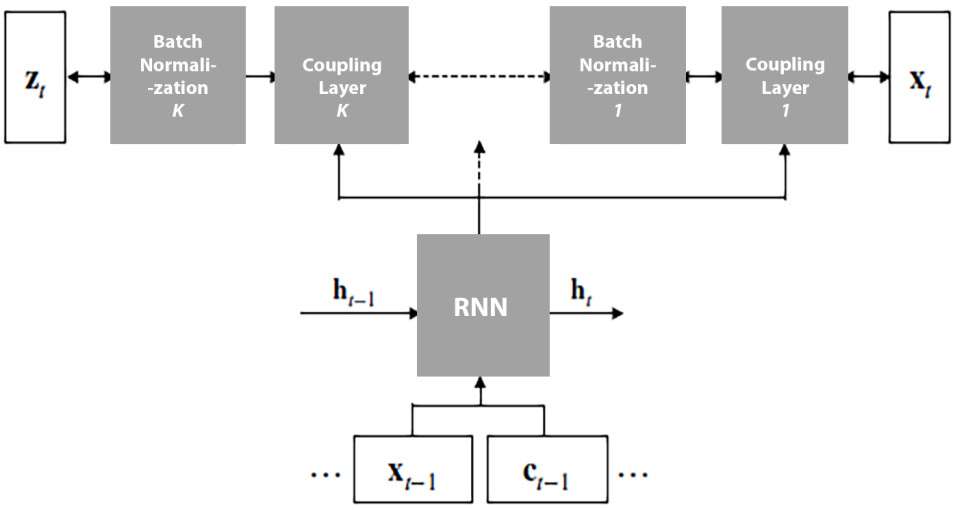


Figure 15: Prediction with DeepAR

5.2.1 RNN Conditioned Real NVP model

The built model is shown in the Figure 16.



Source: Multivariate Probabilistic Time Series Forecasting [21]

Figure 16: RNN Conditioned Real NVP model

This architecture aims to achieve a scalable model for D interacting time series by modelling the conditional joint distribution of all-time series at time t with a normalizing flow, conditioned on the output states of an RNN (LSTM or GRU); i.e

$$p_\chi(x_t|h_t;\theta) \tag{3}$$

The model consists of an RNN followed by Real NVP flow layers. The Real NVP flow consists of the coupling layer followed by batch normalizing layer.

The model takes the time series x_{t-1} and covariates c_{t-1} as input, which are passed to the RNN. In each coupling layer x_t and its transformation is conditioned on the state of the shared RNN from the previous time step x_{t-1} and its covariates c_{t-1}.The Real NVP flow outputs the distribution conditioned on the output from RNN. The samples can be obtained from the output distribution. K in Figure 16 denotes the number of coupling and batch normalization layer.

The model is autoregressive in the sense that it employs the observation of the last time step x_{t-1} as well as the recurrent state h_{t-1} to produce the state h_t on which the current observation is conditioned. In an autoregressive model, the variable of interest is forecasted using a linear combination of the the variable's historical/past values. As the model is autoregressive, it can be written in the form of product of factors

$$p_\chi(x_{t_0:T}|x_{1:t_0-1}, c_{1:T}) = \prod_{t=t_0}^{T} p_\chi(x_t|h_t; \theta), \tag{4}$$

where θ denotes the trainable parameters of both the RNN and flow.

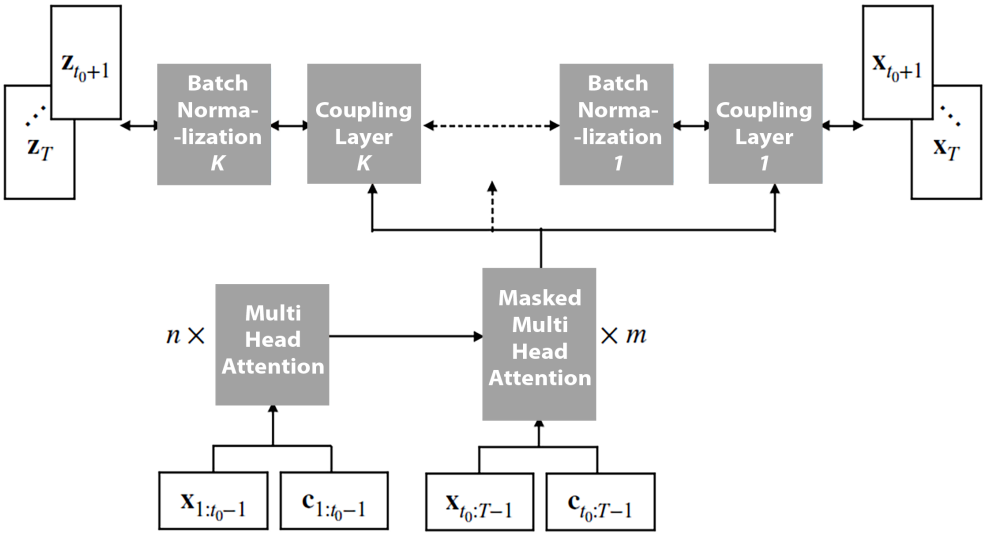
5.2.2 RNN Conditioned MAF model

RNN Conditioned MAF model is also similar to the RNN Conditioned Real NVP model. It differs in the type of flow used. Instead of Real NVP, MAF is used. Similar to the above model, a new architecture has been built where instead of using Real NVP, MAF flow is used.

5.2.3 Transformer Conditioned Real NVP model

Figure 17 shows Transformer Conditioned Real NVP model.

It can be seen that the training time series is split into a encoding/warm up portion $x_{1:t_0-1}$ and the decoding/output portion $x_{t_0:T-1}$. The input embeddings and it's covariates are passed to the multi head attention layer of the transformer. The input passed to the decoder layer goes to the masked multi head attention layer of the transformer to prevent using information from future time points as well as to preserve the autoregressive property by utilizing a mask that reflects the causal direction of progressing time, i.e. to mask out future time points.



Source: Multivariate Probabilistic Time Series Forecasting [21]

Figure 17: Transformer Conditioned Real NVP model

5.2.4 Transformer Conditioned MAF model

Transformer Conditioned MAF model is also similar to the Transformer Conditioned Real NVP model. It differs in the type of flow used. Instead of Real NVP, MAF is used. Similar to the above Transformer Conditioned MAF Model, a new architecture has been built where instead of using Real NVP, MAF flow is used.

5.3 Training the model

As the model is probabilistic, during the training the log-likelihood is maximized using the optimizers; for example training with Adam using Stochastic Gradient Descent (SGD). The log likelihood is given as

$$L = \frac{1}{|D|} \sum_{n=1} logp_{\chi}(x; \theta) \tag{5}$$

and as the RNN is autoregressive; it can be written as

$$h_t = RNN(concat(x_{t-1}; c_{t-1}); h_{t-1}) \tag{6}$$

Using (5) and (6), the log likelihood for the model can be written as

$$L = \frac{1}{|D|T} \sum_{x_{1:T} \in D} \sum_{t=1}^{D} logp_{\chi}(x_t | h_t; \theta) \tag{7}$$

where D = a batch of time series and $|D|$ = batch size

T = time window

x_t = given input time series at time step t

h_t = hidden state at time step t

θ = parameters of RNN and flow

5.4 Tuning the hyperparameters

Hyperparameters are special parameters that express the properties of the model, such as the learning process and the time. These parameters are often provided manually to the model as model can not learn them directly. The hyperparameters depend on the model and data which means different values of hyperparameters can behave differently on different datasets and models. Some of the hyperparameters are discussed below.

5.4.1 Learning Rate

The learning rate is a vital hyperparameter. It controls the size of each step towards the minimum of the objective function which indicates how much to change the model in response to the estimated error each time the model weights are updated. Choosing the perfect learning rate is challenging as well. If we set the learning rate as too low, it may result in a long training process as there will be very tiny updates to the weights in the network, whereas if the value of learning rate is set too high, it may result in converging of the model too quickly to a sub-optimal solution by overshooting a reasonable minimum (global minima) and getting trapped in a bad local minimum. Figure 18 shows the case when the learning rate is too large. It can be seen that too large value of the learning rate causes drastic updates which lead to divergent behaviour.

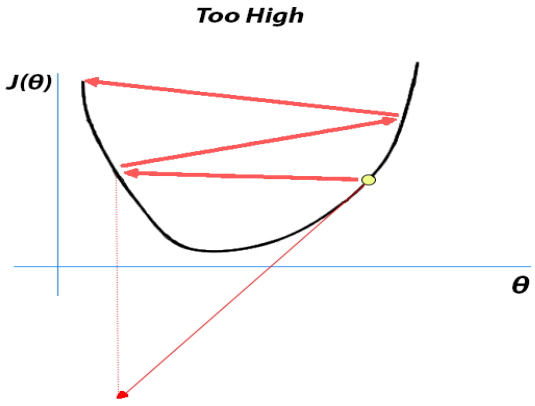


Figure 18: Too large learning Rate

Figure 19 shows the case when learning rate is too small.It can be seen that slow learning rate requires more updates before reaching the minima.

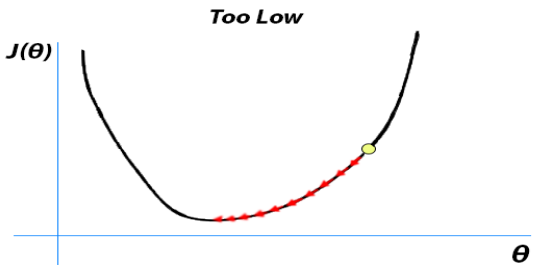


Figure 19: Too small learning Rate

- **OnceCycleLR :** OnecycleLR works on the 1cycle learning rate policy. It sets the learning rate of each parameter group according to this policy. The 1cycle policy takes the learning rate from an initial learning rate to a maximum learning rate and then from that maximum learning rate to a minimum learning rate that is significantly lower than the initial learning rate.

  The learning rate is changed after every batch. In our implementation, the total number of steps is determined by providing the number of epochs and the number of steps per epoch. In this case, the number of total steps is calculated by
  $total\_steps = epochs * steps\_per\_epoch$
  The value of $'epochs'$ and $'steps\_per\_epoch'$ can be defined explicitly.

5.4.2 Epochs

An epoch stands for one complete pass of the training dataset through the machine learning algorithm. Epochs number is one of the essential hyperparameters for the algorithm. It defines the number of epochs or complete

passes of the whole training dataset passing through the training or learning process of the algorithm. With each epoch, the dataset's internal model parameters are updated.

5.4.3 Batch Size

The batch size represents the number of samples/training examples passed to the network at once. A smaller batch size can increase the noise in the gradient descent which results in low accuracy. The result may either diverge or converge to an non optimal minima as it may cause over-fitting of data distribution due to small batch size. It also make the convergence process very slow. A larger batch size can perform and produce better result with the training dataset, it can suffer from poor performance with data outside of training dataset. A larger dataset can take longer time to converge and also consumes lot of computing resources as data from total batch has to be computed.

5.4.4 Normalizing Flows Layers Blocks

It represents the stacks / number of normalizing flow bijections layers.

5.4.5 Transformer Hyperparameters

- d_model – It indicates the number of expected features in the encoder/decoder inputs.
- nhead – It indicates the number of heads in the multihead attention models.
- Activation Function- Gaussian Error Linear Units (GELU) is used as an activation function in the Transformer. GELU [15] is a high-performing neural network activation function. The GELU activation function is defined as x$\phi(x)$, where $\phi(x)$ is the standard Gaussian cumulative distribution function.

5.5 Evaluation

Evaluation metrics help in understanding how well the model has performed and how good the forecasts are. Following are the evaluation metrics which are used.

5.5.1 Continuous Ranked Probability Score (CRPS)

CRPS [18] determines the compatibility of a probability distribution F represented by its quantile function F^{-1} with an observation z. The CRPS has an inherent definition as the pinball loss integrated across all quantile levels $\alpha \in [0,1]$. The pinball loss function also known as quantile loss is a metric that is used to evaluate the accuracy of a quantile forecast.

The pinball loss at a quantile level $\alpha \in [0,1]$ and with the predicted $\alpha{-}th$ quantile q is defined as

$$\Lambda_\alpha(q, z) = (\alpha - \Gamma_{[z<q]})(z - q)$$

where $\Gamma_{[z<q]}$ denotes the indicator function, taking value 1 if $z < q$ and 0 otherwise. Further the CRPS can be defined as [10]

$$CRPS(F^{-1}, z) = \int_0^1 2\Lambda_\alpha(F^{-1}(\alpha), z)\, d\alpha.$$

CRPS has an important property that it is a proper scoring rule [12] which means for any distributions F and G (g being the density of G)

$$\int g(z)CRPS(G^{-1}, z)dz \leq \int g(z)CRPS(F^{-1}, z)dz$$

which implies that the value of CRPS is minimized when the predictive distribution F is equal to the distribution from which the data is drawn i.e G. Lower value of CRPS indicates better forecasts. For the evaluation, CRPS is also computed on the sum of all time series which is indicated as CRPS_{sum}. It can be computed by first summing across the D time series and then averaging the results over the prediction range. When compared, lower value of CRPS are better.

5.5.2 Mean Square Error (MSE)

MSE is defined as the mean squared error over all the time series and whole prediction range

$$MSE = \frac{1}{N(T-t_0)} \sum_{i,t} (z_{i,t} - \hat{z}_{i,t})^2;$$

where $i = 1, ...N$ represents the time series,
$t = T - t_0 + 1, ..., T$ represents the prediction range;
z is the target and $\hat{z}$ the predicted distribution mean.
A MSE of zero value indicates perfect forecast, or no error.

5.5.3 Normalized Deviation (ND)

ND metrics is defined as

$$ND = \frac{\sum_{i,t} |z_{i,t} - \hat{z}_{i,t}|}{\sum_{i,t} |z_{i,t}|}$$

where $\hat{z}_{i,t}$ denotes the predicted mean value for item i at the time t

5.5.4 Normalized Root Mean Square Error (NRMSE)

Root Mean Square Error (RMSE) metrics is defined as

$$RMSE = \frac{\sqrt{\frac{1}{N(T-t_0)} \sum_{i,t} |z_{i,t} - \hat{z}_{i,t}|^2}}{\frac{1}{N(T-t_0)} \sum_{i,t} |z_{i,t}|}$$

where $\hat{z}_{i,t}$ denotes the predicted mean value for item i at the time t. When the RMSE is normalized, it is known as NRMSE. Therefore, NRMSE can be defined as

$$NRMSE = \frac{RMSE}{\bar{z}}$$

where $\bar{z}$ denotes the target mean value.

6 Results

This section includes the results obtained using different models on different datasets. The sum in the evaluation metrics denote the metrics aggregated over all the time series of the dataset. For example, CRPS_{sum} indicates the CRPS computed on the sum of all time series.

6.1 Evaluation Metrics for different Datasets

Table 1 shows the results of different evaluation metrics on Electricity Dataset.

Electricity Dataset				
Model	CRPS_{sum}	NRMSE_{sum}	ND_{sum}	MSE_{sum}
RNN Conditioned NVP	0.020	0.038	0.028	1.14×10^8
RNN Conditioned MAF	0.021	0.039	0.028	1.24×10^8
Transformer Conditioned NVP	0.030	0.050	0.041	1.96×10^8
Transformer Conditioned MAF	0.027	0.047	0.036	1.80×10^8

Table 1: Evaluation Metrics for Electricity Dataset using different models

Table 2 shows the results of different evaluation metrics on Solar Dataset.

Solar Dataset				
Model	CRPS_{sum}	NRMSE_{sum}	ND_{sum}	MSE_{sum}
RNN Conditioned NVP	0.273	0.654	0.353	7.63×10^6
RNN Conditioned MAF	0.386	0.882	0.471	13.86×10^6
Transformer Conditioned NVP	0.406	0.802	0.481	15.72×10^6
Transformer Conditioned MAF	0.321	0.748	0.406	9.96×10^6

Table 2: Evaluation Metrics for Solar Dataset using different models

Table 3 shows the results of different evaluation metrics on Traffic Dataset.

Traffic Dataset				
Model	$CRPS_{sum}$	$NRMSE_{sum}$	ND_{sum}	MSE_{sum}
RNN Conditioned NVP	0.359	0.493	0.557	789.27
RNN Conditioned MAF	0.213	0.372	0.277	353.37
Transformer Conditioned NVP	0.112	0.209	0.143	111.63
Transformer Conditioned MAF	0.107	0.140	0.227	131.09

Table 3: Evaluation Metrics for Traffic Dataset using different models

6.2 Prediction Intervals and Test Set Ground Truth

The figures in this section show the below information

- Test set ground truth observation
- Median of the forecast samples
- 50% and 90% prediction intervals - Prediction interval indicates the confidence that the actual observation will fall in the predicated interval. 90% interval indicates that the model is 90% confident that actual observation will fall in 90% prediction range. With the increase in number, the prediction range also increases which means range for prediction interval 90% will be wider than the prediction interval 50%.

6.2.1 Electricity Dataset

Figure 20 shows the Prediction intervals and test set ground-truth from the different models for Electricity data of the 4 different time series. It can be seen that the prediction intervals cover the actual observations range most of the time.

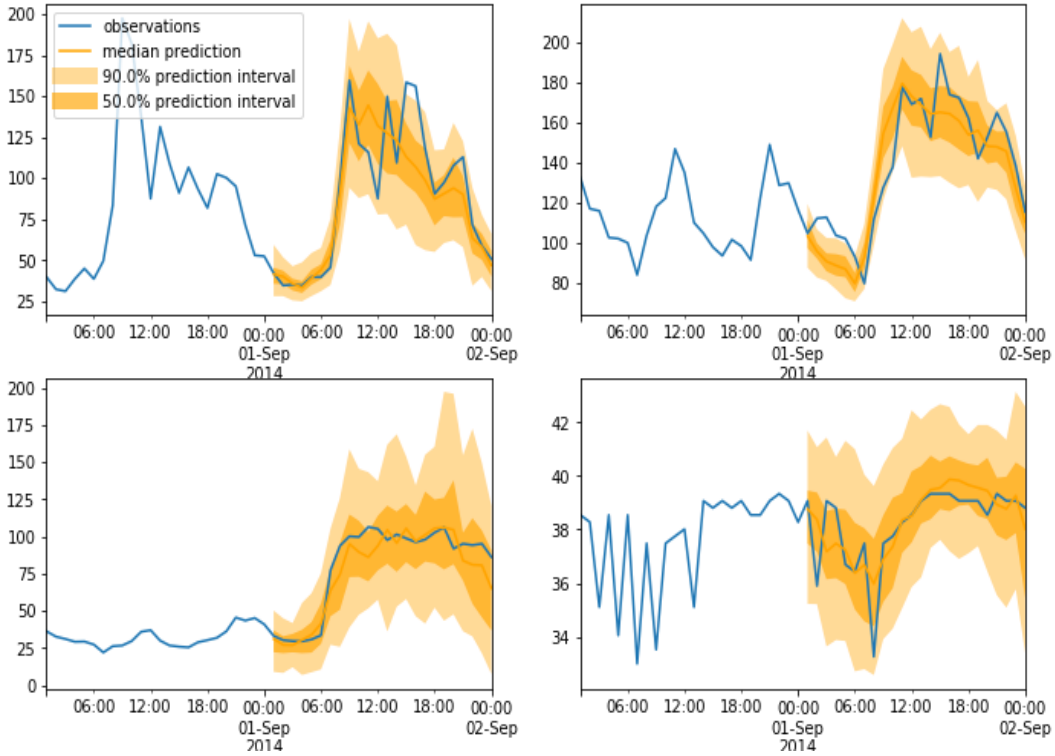


Figure 20: Prediction intervals and test set ground-truth from RNN conditioned MAF model for Electricity data of the 4 different time series.

6.2.2 Traffic Dataset

Figure 21 shows the Prediction intervals and test set ground-truth from Transformer conditioned MAF model for Traffic dataset data of the 4 different time series. It can be seen that the prediction intervals cover the actual observations range most of the time.

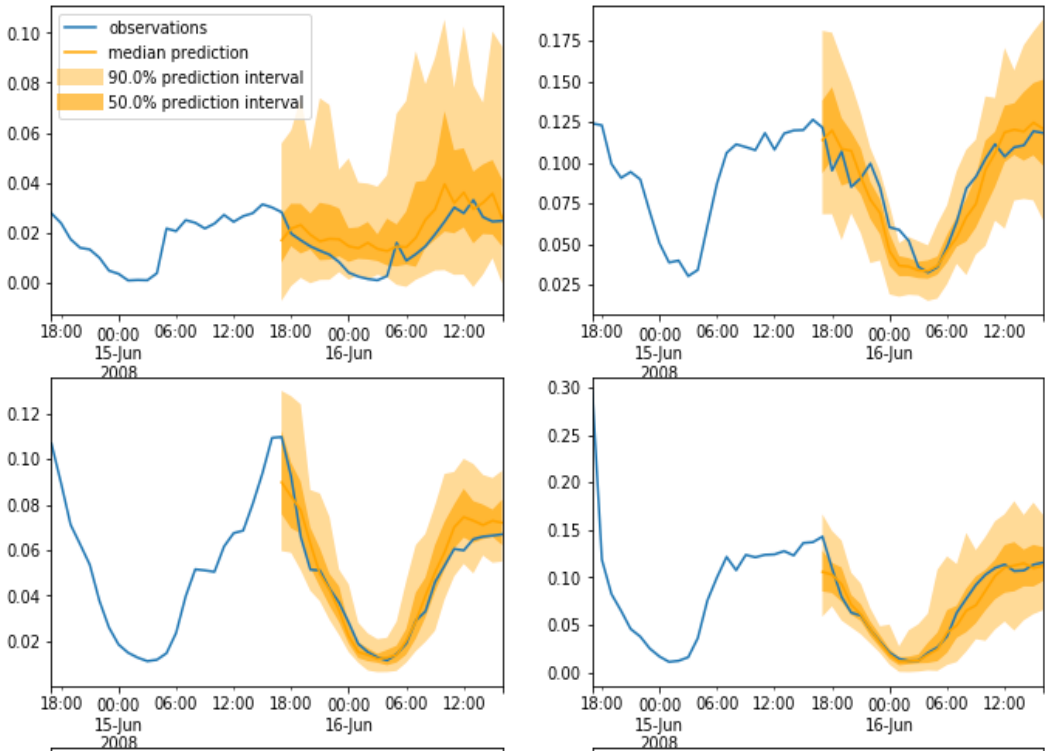


Figure 21: Prediction intervals and test set ground-truth from Transformer MAF model for Traffic data of the 4 different time series.

6.2.3 Solar Dataset

Figure 22 shows the Prediction intervals and test set ground-truth from Transformer MAF model for Solar data of the 4 different time series. It can be seen that the prediction intervals cover the actual observations range most of the time.

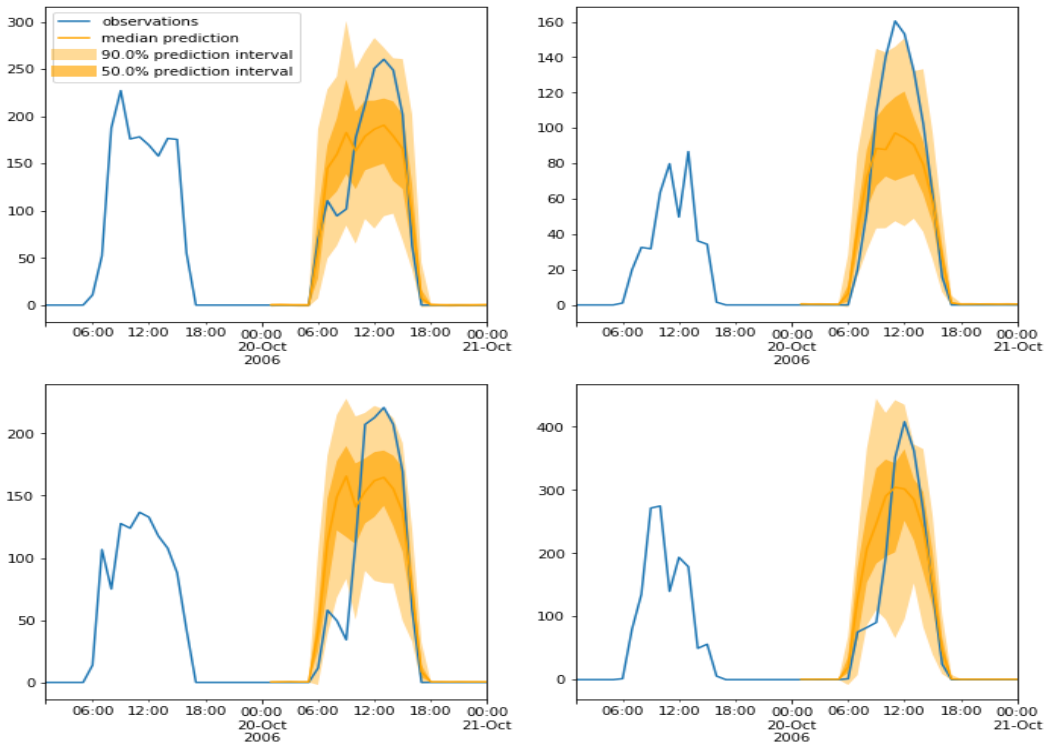


Figure 22: Prediction intervals and test set ground-truth from Transformer MAF model for Solar data of the 4 different time series.

An evaluation metrics comparison table is presented in Appendix section which compares the results between DeepAR and other models.

6.3 Results on H&M Dataset

The models were evaluated on different length of target dimensions and timesteps.

6.3.1 Resuts with 70 unique SKUs

To start with the results were obtained on 70 unique SKUs from 5 unique warehouses. These SKUs belong to the same category. Category here can be like "MensWear", "LadiesWear", "KidsWear" or similar. The target dimension is 310. The data contains the sales value for the SKUs from "2019-07-07" to "2020-10-18". The total timesteps between these dates are 67, out of which data from "2020-01-5" to "2020-04-19" (15 weeks) is missing. Therefore, the total timesteps are 52. The results for two different experiments with the above discussed data are considered below

- **Missing Timesteps are not included:** This is the case where missing time steps from the data were not included. When the missing data is not considered and the continuous time steps are taken into consideration which means data from the period "2019-07-07" to "2020-01-5" (26 weeks). According to this data, there are only 20 time steps to train on and the last 6 time steps are used for prediction. Table 4 shows the values of different evaluation metrics. The columns with '-' indicate 'NaN'. For the models where MAF is used, the output for MSE_{sum} and NRMSE_{sum} was 'NaN'. For NVP based model also, the MSE_{sum} is very high. It can be observed, the results are not good. The reason behind this could be the model not getting enough time steps to train on to learn the pattern.

H&M Dataset				
Model	CRPS_{sum}	NRMSE_{sum}	ND_{sum}	MSE_{sum}
RNN Conditioned NVP	0.565	0.706	0.578	1.71×10^7
Transformer Conditioned MAF	0.445	-	0.581	-
Transformer Conditioned NVP	0.560	0.705	0.580	1.70×10^7
RNN Conditioned MAF	0.570	-	0.587	-

Table 4: Evaluation Metrics on H&M Dataset which do not include missing time steps

- **Missing time steps are imputed with 0 :** The missing values are imputed with 0. This resulted in 68 time steps. Table 5 provides the results of evaluation metrics for this case.

H&M Dataset				
Model	$CRPS_{sum}$	$NRMSE_{sum}$	ND_{sum}	MSE_{sum}
RNN Conditioned NVP	0.570	1.332	0.730	3.70×10^3
Transformer Conditioned MAF	0.683	-	0.581	-
Transformer Conditioned NVP	0.505	1.598	0.832	5.32×10^3

Table 5: H&M Dataset Results with Imputing Missing Values with 0

6.3.2 Results with 63 unique SKUs

In this case, 63 unique SKUs belonging to the similar category are selected.These SKUs are from similar Warehouse. The data starts from 2019-07-28 to 2020-06-14.The missing data is imputed with 0. The target dimension is 63 and prediction length is 6.

Table 6 shows the values of different evaluation metrics calculated for the discussed case. The $CRPS_{sum}$ for DeepAR is blank as DeepAR is not a multivariate model. It can be seen in the table that when compared to the point forecast model (DeepAR), the results are comparable.Transformer Conditioned MAF model has close MSE_{sum} value to DeepAR.

It can also be seen in the Table 6 that evaluation metrics have improved when compared to the above cases on H&M dataset.

H&M Dataset				
Model	CRPS_{sum}	NRMSE_{sum}	ND_{sum}	MSE_{sum}
RNN Conditioned NVP	0.221	0.352	0.230	143.65
RNN Conditioned MAF	0.401	0.598	0.471	414.57
Transformer Conditioned NVP	0.262	0.384	0.384	171.08
Transformer Conditioned MAF	0.208	0.333	0.247	128.85
DeepAR	-	0.319	0.838	117.72

Table 6: Evaluation Metrics on H&M Dataset with SKUs from similar category and single Warehouse

Figure 23 shows the plot for Prediction intervals (0.50 and 0.90) and test set ground-truth from Transformer conditioned MAF model for H&M data of the 4 different time series from single warehouse. It can be seen in the Figure that the forecasts are not very good when compared to the other datasets. The prediction intervals do not cover the actual observation range even for 90% prediction interval in most of the cases.

If these intervals are forecasted properly, it can be useful to get an estimate of the demand for the articles. For example, if the prediction (90% prediction interval) for an article include range from 2 to 10, the model is 90% is confident that the sale of particular article will fall under this range.

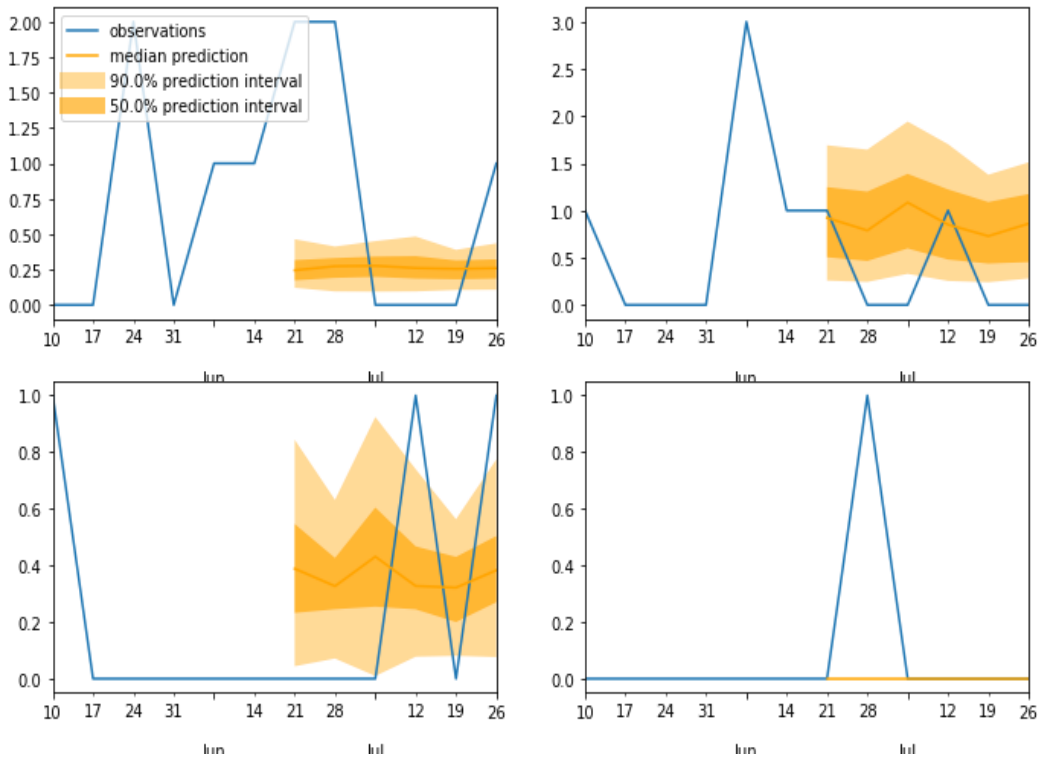


Figure 23: Prediction intervals and test set ground-truth from Transformer conditioned MAF model for H&M data(single warehouse) of the 4 time series.

7 Conclusions

The thesis aims to build a multivariate probabilistic forecasting model using normalizing flows, which can be used for demand forecasting. The various concepts are investigated and analyzed, which in combination, can result in a good probabilistic model. Autoregressive models using recurrent neural networks such as 'GRU', 'LSTM' and attention mechanisms such as 'Transformers' are discussed. Autoregressive models work very well for time series forecasting because of their good performance in extrapolation into the future. The flow models resulted in modelling complex distributions very well. These flows do not assume the distribution as a simple or normal distribution. The model does not only consider the past values but take the other time series also in consideration. The model gave impressive results on open datasets. These datasets are clean, and mostly, all the time series contain a significant length which is not the case with the H&M dataset. The articles in fashion retail often consist of an ephemeral life cycle. H&M has a vast dataset that contains data of many articles and their sales.

As it can be seen in the Results section, different flow models gave different results. MAF performed better than the NVP for open datasets. Similarly, other flow models can be analyzed and implemented to see how the results can be improved. The current model provided very good results on open datasets with minimal hyperparameters tuning. Further tuning can, of course, improve the results more. There can be more experiments with different values of hyperparmeters used. Along with model building and hyperparameters tuning, data preprocessing is a crucial part. A good model with badly unprocessed data can never result in acceptable results. Therefore, the model data needs to be processed, appropriately cleaned. Further, the ways to improve the results on time series datasets like H&Ms can be researched more, as it is still an open problem how to model discrete ordinal data via flow - which would best capture the nature of some data sets (e.g. sales data)[22].

References

[1] Deepak Agrawal and Christopher Schorling. "Market share forecasting: An empirical comparison of artificial neural networks and multinomial logit model". In: *Journal of Retailing* 72.4 (1996), pp. 383–407. ISSN: 0022-4359. DOI: https://doi.org/10.1016/S0022-4359(96)90020-2. URL: https://www.sciencedirect.com/science/article/pii/S0022435996900202.

[2] Hiram C. Barksdale and Jimmy E. Hilliard. "A Cross-Spectral Analysis of Retail Inventories and Sales". In: *The Journal of Business* 48.3 (1975), pp. 365–382. ISSN: 00219398, 15375374. URL: http://www.jstor.org/stable/2352230.

[3] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[4] Nicolas Chapados. "Effective Bayesian Modeling of Groups of Related Count Time Series". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, 22–24 Jun 2014, pp. 1395–1403. URL: http://proceedings.mlr.press/v32/chapados14.html.

[5] XI Chen et al. "PixelSNAIL: An Improved Autoregressive Generative Model". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 864–872. URL: http://proceedings.mlr.press/v80/chen18h.html.

[6] KyungHyun Cho et al. "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *CoRR* abs/1409.1259 (2014). arXiv: 1409.1259. URL: http://arxiv.org/abs/1409.1259.

[7] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[8] J. D. Croston. "Forecasting and Stock Control for Intermittent Demands". In: *Operational Research Quarterly (1970-1977)* 23.3 (1972), pp. 289–303. ISSN: 00303623. URL: http://www.jstor.org/stable/3007885.

[9] Valentin Flunkert, David Salinas, and Jan Gasthaus. "DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks". In: *CoRR* abs/1704.04110 (2017). arXiv: 1704.04110. URL: http://arxiv.org/abs/1704.04110.

[10] Jan Gasthaus et al. "Probabilistic Forecasting with Spline Quantile Function RNNs". In: *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 16–18 Apr 2019, pp. 1901–1910. URL: https://proceedings.mlr.press/v89/gasthaus19a.html.

[11] Mathieu Germain et al. "MADE: Masked Autoencoder for Distribution Estimation". In: *CoRR* abs/1502.03509 (2015). arXiv: 1502.03509. URL: http://arxiv.org/abs/1502.03509.

[12] T. Gneiting and A. Raftery. "Strictly Proper Scoring Rules, Prediction, and Estimation". In: *Journal of the American Statistical Association* 102 (2007), pp. 359–378.

[13] H&M Group. *H&M Group Market overview*. URL: https://hmgroup.com/about-us/markets-and-expansion/market-overview/. (accessed: 24.07.2021).

[14] Rafael S. Gutierrez, Adriano O. Solis, and Somnath Mukhopadhyay. "Lumpy demand forecasting using neural networks". In: *International Journal of Production Economics* 111.2 (2008). Special Section on Sustainable Supply Chain, pp. 409–420. ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2007.01.007. URL: https://www.sciencedirect.com/science/article/pii/S0925527307000540.

[15] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2020. arXiv: 1606.08415 [cs.LG].

[16] Yoon Kim et al. "Structured Attention Networks". In: *CoRR* abs/1702.00887 (2017). arXiv: 1702.00887. URL: http://arxiv.org/abs/1702.00887.

[17] Manoj Kumar et al. "VideoFlow: A Flow-Based Generative Model for Video". In: *CoRR* abs/1903.01434 (2019). arXiv: 1903.01434. URL: http://arxiv.org/abs/1903.01434.

[18] James E. Matheson and Robert L. Winkler. "Scoring Rules for Continuous Probability Distributions". English. In: *Management science* 22.10 (1976), pp. 1087–1096.

[19] Maryam Mohammadipour, John Boylan, and Aris Syntetos. "The Application of Product-Group Seasonal Indexes to Individual Products". In: *Foresight: The International Journal of Applied Forecasting* 26 (Summer 2012), pp. 20–26. URL: https://ideas.repec.org/a/for/ijafaa/y2012i26p20-26.html.

[20] Ankur P. Parikh et al. "A Decomposable Attention Model for Natural Language Inference". In: *CoRR* abs/1606.01933 (2016). arXiv: 1606.01933. URL: http://arxiv.org/abs/1606.01933.

[21] Kashif Rasul et al. "Multi-variate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows". In: *CoRR* abs/2002.06103 (2020). arXiv: 2002.06103. URL: https://arxiv.org/abs/2002.06103.

[22] Kashif Rasul et al. "Multi-variate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows". In: *CoRR* abs/2002.06103 (2020). arXiv: 2002.06103. URL: https://arxiv.org/abs/2002.06103.

[23] David Salinas et al. "High-Dimensional Multivariate Forecasting with Low-Rank Gaussian Copula Processes". In: *CoRR* abs/1910.03002 (2019). arXiv: 1910.03002. URL: http://arxiv.org/abs/1910.03002.

[24] Matthias W Seeger, David Salinas, and Valentin Flunkert. "Bayesian Intermittent Demand Forecasting for Large Inventories". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/03255088ed63354a54e0e5ed957e9008-Paper.pdf.

[25] Tao Shen et al. *DiSAN: Directional Self-Attention Network for RNN/CNN-Free Language Understanding*. 2017. arXiv: 1709.04696 [cs.CL].

[26] Ralph D. Snyder, J. Keith Ord, and Adrian Beaumont. "Forecasting the intermittent demand for slow-moving inventories: A modelling approach". In: *International Journal of Forecasting* 28.2 (2012), pp. 485–496. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2011.03.009. URL: https://www.sciencedirect.com/science/article/pii/S0169207011000781.

[27] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[28] Ashish Vaswani et al. “Attention is All You Need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[29] Song-yi Youn, Jung Eun Lee, and Jung Ha-Brookshire. “Fashion Consumers’ Channel Switching Behavior During the COVID-19: Protection Motivation Theory in the Extended Planned Behavior Framework”. In: *Clothing and Textiles Research Journal* 39.2 (2021), pp. 139–156. DOI: 10.1177/0887302X20986521. eprint: https://doi.org/10.1177/0887302X20986521. URL: https://doi.org/10.1177/0887302X20986521.

8 Appendix

8.1 Covariance Matrix

Covariance matrices represent the covariance values of each pair of variables in multivariate data. The covariance between the identical variables equals variance, so in a covariance matrix, the diagonal shows the variance of each variable. Let us say there are two variables as x and y in the data set. The covariance matrix for these variables looks like

$$\begin{bmatrix} var(x) & cov(x,y) \\ cov(x,y) & var(y) \end{bmatrix}$$

8.2 PDF

Probability density function (PDF) defines a probability distribution or the likelihood of an outcome for a discrete random variable as opposed to a continuous random variable.

8.3 CDF

The cumulative distribution function (CDF) of a random variable is a method to represent the distribution of random variables.CDF can be defined for any kind of random variable, for example, discrete, continuous, and mixed.

8.4 Copula

Copulas are the multivariate cumulative distribution functions used for describing the dependency between the random variables having marginal probability distribution with uniform interval [0 , 1] .

8.5 Chain Rule

The chain rule is an important derivation rule which is essential in understanding the working of the backpropagation algorithm. It help in calculating the error gradient of the loss function with respect to each weight in a neural network. It expresses the derivation of the composition of two differentiable functions f and g in two separate derivatives of f and g.

The application of chain rule states that the derivation of the outer function has to be performed first and then proceed further through inner functions. The chain rule can be applied to both univariate functions and also to multivariate function.

8.5.1 Univariate Function

Univariate functions are the functions with only single variable such as $f(x) = x^3$ which describes the change of function is based on only on one variable. The chain rule for the univariate function can be defined as $y = f(g(x))$ where $f(x)$ and $g(x)$ are univariate functions then $y^{'} = f^{'}(g(x)).g^{'}(x)$.

Let us suppose, $y = sin(10x^3)$, the derivation of y can be explained using chain rule.
Let $f(x) = sin(x)$ and $g(x) = 10x^3$, then $g^{'}(x) = 30x^2$.
$y^{'} = f^{'}(g(x)).g^{'}(x)$, by substituting we get $y^{'} = cos(10x^3).30x^2$.
Therefore, the derivation of $y = sin(10x^3)$ is $y^{'} = 30x^2cos(10x^3)$.

8.5.2 Multivariate Function

Multivariate functions are the functions with input and/or output consisting of more than one variable such as $z = f(x, y) = x^3 + 2y$ which describes that change of function depends on range of multiple variables. The chain rule for multivariate function requires the use of partial derivative. Let $z = f(x, y)$ where $x = g(t)$ and $y = h(t)$, then $z = f(x(t), y(t))$. The chain rule for the multivariate function z can be defined as
$\frac{dz}{dt} = \frac{\partial z}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial z}{y} \cdot \frac{dy}{dt}$, where ordinary derivations are evaluated at t and partial derivations are evaluated at (x, y).

Let $z = f(x, y) = 4x^3 + 2y^2$, where functions $x = x(t) = sin(t)$ and $y = y(t) = cos(t)$, the derivation of z can be explained using chain rule.

To applying the chain rule to the above equation, we need to evaluate following quantities

$\frac{\partial z}{\partial x} = 12x^2$, $\cdot\frac{dx}{dt} = cos(t)$, $\frac{\partial z}{y} = 4y$, $\frac{dy}{dt} = -sin(t)$

by substituting above quantities in equation $\frac{dz}{dt} = \frac{\partial z}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial z}{y} \cdot \frac{dy}{dt}$, we get $\frac{dz}{dt} = (12x^2).(cos(t)) + (4y).(-sin(t)) \implies 12\mathrm{x}^2 cos\, t - 4y sin\, t.$

We know that, $x = x(t) = sin\, t$ and $y = y(t) = cos\, t$, by substituting them in above equation we get.

$\frac{dz}{dt} = 12(sin\, t)^2 cos\, t - 4(cos\, t) sin\, t.$

8.6 Change of Variables Formula

A change of variables is basic technique of solving complex equations by replacing the original variables with the functions of other variable to make equations simpler.

Let us assume we have a 6th degree polynomial say $x^6 - 6x^3 + 1 = 0$. Finding roots for sixth degree polynomial is very difficult in terms of radicals. We can solve the above equation by simplifying the equation by replacing x with new variable u $x = \sqrt[3]{u}$ or $u = x^3$.

By replacing we get a simplified quadratic equation, $u^2 - 6u + 1 = 0$, which has two solutions $u = 3 - 2\sqrt{2}$ and $u = 3 + 2\sqrt{2}$

By substituting the original values $u = x^3$, we get $x^3 = 3 - 2\sqrt{2}$ and $x^3 = 3 + 2\sqrt{2}$. Therefore, the value of x is $\sqrt[3]{3 - 2\sqrt{2}}$ and $\sqrt[3]{3 + 2\sqrt{2}}$.

8.7 Comparison With DeepAR model

Table 7 shows the results of different evaluation metrics on Electricity Dataset.

Electricity Dataset		
Model	CRPS	ND
DeepAR	0.050	0.061
TempFlow NVP	0.052	0.069
TempFlow MAF	0.051	0.069
Transformer NVP	0.070	0.093
Transformer MAF	0.059	0.078

Table 7: Evaluation Metrics for Electricity Dataset using different models

Table 8 shows the results of different evaluation metrics on Solar Dataset.

Solar Dataset		
Model	CRPS	ND
DeepAR	0.398	0.490
TempFlow NVP	0.397	0.503
TempFlow MAF	0.424	0.532
Transformer NVP	0.887	1.224
Transformer MAF	0.374	0.495

Table 8: Evaluation Metrics for Solar Dataset using different models

Table 9 shows the results of different evaluation metrics on Traffic Dataset.

Traffic Dataset		
Model	CRPS	ND
DeepAR	0.126	0.150
TempFlow NVP	0.416	0.545
TempFlow MAF	0.296	0.377
Transformer NVP	0.182	0.229
Transformer MAF	0.183	0.233

Table 9: Evaluation Metrics for Traffic Dataset using different models