FACULTÉ DES SCIENCES APPLIQUÉES
Ecole d'ingénieurs et d'informaticiens

LIÈGE université

FIRST VERSION

Master Thesis

Stassen Théo (s150804)

Promoter : D. Ernst

2019-2020

1 Introduction and problem definition

1.1 Introduction

Blacklight Analytics is a enterprise that develop solutions regarding the management of electrical distribution networks. Distribution systems need to integrate more and more renewable generation in their network. Since networks cannot be quickly upgraded and at low cost, new generators are connected to the network under non-firm access contract. The assets used in networks have been designed to transmit power in one direction, from the global network to the local network. This configuration implies situations where the high production creates congestion problems in the assets, while energy generated must be injected into the global network . There is a need for a practical method able to compute the production limits of the generators such that the system can be considered safe, i.e has a very low probability that the electrical power flowing through one asset is higher than the maximum tolerated power is this asset.

Blacklight Analytics develops a solution to this problem. The method cast the problem into a stochastic decision process. This process is divided in three phases : (i) generation of a network model, (ii) forecasting of the power produced or consumed and (iii) computation of the generator limits. The phase 2 takes as input historical measurements of the networks and output future production probability distribution for each sources. This output will be used to define generator limits. In the current implementation, the forecasting method uses Gaussian process regression to forecast individually each source and combine information using covariance estimation. This master thesis subject addresses the question of what is the best forecasting method to implement in this described context, and what are the elements of comparison that allows us to make these affirmation. The growing scientific field of Deep Learning has a great potential to be exploited to achieve this goal.

Therefore, the concrete goal of this Master Thesis is the comparison of different probabilistic forecasting models and techniques, mostly from the deep learning scientific field, in the context of the prediction of renewable energy production.

1.2 Related Works

Time series forecasting is a well-defined and extensively explored scientific field. It is considered as part of the probabilities scientific field for many years, from very simple or naive forecasting models (*Naive*, *Seasonal Naive*, *Moving Average*, etc) **forecasting_principle1** to more complex models as the *Autoregressive integrated moving average* (*ARIMA*) **forecasting_principle2** or *Exponential smoothing*, introduced by statistician Robert Goodell Brown in 1956 **smooth_for**. *Exponential smoothing* and *ARIMA* models are the two most widely used approaches to time series forecasting, and provide complementary approaches to the problem. While *exponential smoothing* models are based on a description of the trend and seasonality in the data, *ARIMA* models aim to describe the autocorrelations in the data.

Probabilistic time series forecasting stems from point time series forecasting. For instance, *ETS* forecasting is an *exponential smoothing* method that can generate point but also intervals prediction **exp_smooth**.

In recent years, advances in deep learning has led to interesting results in probabilistic forecasting field. Recent publications (**deep_factors_gp**, **multi_horizon_quantile**) and forecasting competitions results (**extreme_event_for_nn**, **m4_competition**) have shown the relevance of using deep learning based models to obtain better results than with ETS or ARIMA methods.

Deep Learning probabilistic forecasting of renewable energy production is a subject that has already been covered by various scientific teams, with different contexts, goals and techniques. These various results has been compiled and analysed in different scientific review papers. Probabilistic forecasting in the context of wind turbines generation has been reviewed in **review_prob_for_wind**. Deep learning techniques for renewable energy forecasting has been reviewed in **review_deep_renewable_for**. These reviews exposes notably a great number of different models.

Regarding the implementation of these techniques, deep learning frameworks, such as *Tensorflow* **tensorflow**, *MXNet* **mxnet** and *Pytorch* **pytorch** are popular solutions. Considering the time series modeling, a number of commercial and open-source solutions exist but does not provides toolkits focused on modern deep learning. For example the *R-forecast* package **r_article** provide a plethora of models and tooling for classical forecasting methods and contains neural forecasting models, however these pre-date modern deep learning methods and only contain stand-alone implementations of simple local models, they lack state-of-the-art architectures. This lack of specific toolkit has been recently filled with *GluonTS* (`https://gluon-ts.mxnet.io`) **gluontsbible**, a deep learning library that bundles components, models and tools for time series applications such as point and probabilistic forecasting or anomaly detection. It relies on the deep-learning framework MXNet, developed by *Amazon Web Services.* As it provides all the services needed to produce experiments and comparison of deep learning probabilistic forecasting models, it is the main implementation tool of this master thesis.

1.3 Problem Statement

Before entering into solution description, we need to define mathematically in general the problem of time series deep-learning forecasting, whereabouts *GluonTS* has been conceived.

As different types of models has fundamental differences in the way they express the problem, the mathematical formulation differs. Two mathematical formulation are presented, corresponding to two types of models found among the implemented models. The first mathematical formulation corresponds to the feed-forward models. The second mathematical formulation corresponds to the RNN models.

Models data is composed of certain number of time series in training data and in testing data. In general the training and testing data is composed of the same time series (this affirmation is discussed in section 2.3). This situation is the one considered in this section.

Let the following variables :

- I the number of time series considered
- T the number of time steps in time series
- $x_{i,t}$ a scalar variable representing the value of the time series i at time step t.
- a, b, c, d, e specific values of t, with $e = T$
- $t = 0 < t = a < t = b < t = c < t = d < t = e$
- $predict\_length = b - a$, fixed hyper parameter of the problem. Indicates the number of time steps about which the models must make a prediction simultaneously.
- $context\_length = c - b$, fixed hyper parameter of the problem. Indicates the number of time steps that the model consider as input simultaneously.

We also define $L : \mathbb{R} \times (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R}$ the loss function taking a value and a probability distribution as input and giving a loss value as output, As example of function L, we can mention the negative log-likelihood :

$$L(y, \phi(x)) = -ln(\phi(x = y)) \tag{1}$$

The details of loss implementation are discussed in section 2.5.

Each time series can be conceptually decomposed as two pieces. The first piece, $[x_{i,0}, ..., x_{i,d-1}]$ correspond to the part of the time series whereupon we want to train. The second piece $[x_{i,d}, ..., x_{i,e-1}]$ correspond to the part of the time series whereupon we want to test. The training set is composed only of the first pieces of each time series when the training set is composed of the two pieces (i.e. the totality of the time series).

At the end of the training, the model neural network has been trained on the whole training data, and can be tested on the testing data.

1.3.1 First mathematical formulation : Feed Forward models

This formulation corresponds to the way the models that can be described as Feed Forward (*Simple* and *SimpleFeedForward*) compute a solution of the problem.

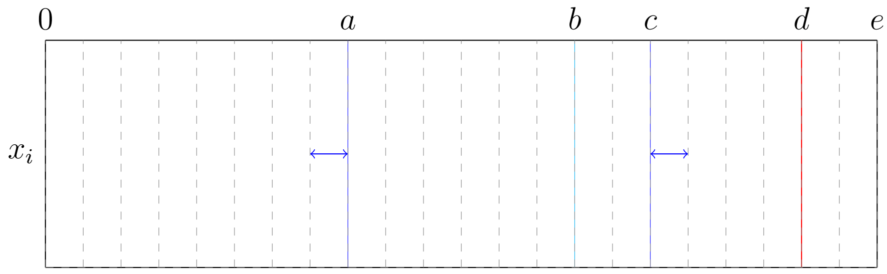


Figure 1: Feed forward models forecast : General case

Considering an interval $[x_{i,a}, ..., x_{i,c-1}]$ in the time series i, our goal is to estimate the distribution of future window $[x_{i,b}, ..., x_{i,c-1}]$ given its history $[x_{i,a}, ..., x_{i,b-1}]$. The estimated distribution is defined as :

$$p_\theta(x_{i,b}, ..., x_{i,c-1}|x_{i,a}, ..., x_{i,b-1}) \tag{2}$$

Where θ denotes the models parameters. This distribution can be factorised as :

$$p_\theta(x_{i,b}, ..., x_{i,c-1}|x_{i,a}, ..., x_{i,b-1}) = \prod_{t=b}^{c-1} p_\theta(x_{i,t}|x_{i,a}, ..., x_{i,b-1}) \tag{3}$$

We define NN_θ, a neural network parametrized by θ. It hypothesis space is defined as $\{h : \mathbb{R}^{b-a} \to R^{k(c-b)}, h \in H\}$. With k the number of parameters needed to defines the output distribution. We consider the case of a gaussian distribution, with $k = 2$. The evaluation of NN_θ can be described as :

$$[\mu_{\theta i,b}, .., \mu_{\theta i,c-1}, \sigma_{\theta i,b}, .., \sigma_{\theta i,c-1}] = \text{NN}_\theta(x_{i,a}, .., x_{i,b-1}) \tag{4}$$

The estimated distribution of a variable $x_{i,t}$ of the future window ($t \in [b, c-1]$) is expressed as, with $\phi_{\mu,\sigma} : \mathbb{R} \to \mathbb{R}$ a gaussian function parametrized by parameters μ and σ :

$$p_\theta(x_{i,t}|x_{i,a}, ..., x_{i,b-1}) = \phi_{\mu_{\theta i,t},\sigma_{\theta i,t}}(x_{i,t}) \tag{5}$$

The estimated distribution of the future window as defined in equation 2 is expressed as :

$$p_\theta(x_{i,b}, ..., x_{i,c-1}|x_{i,a}, ..., x_{i,b-1}) = \prod_{t=b}^{c-1} \phi_{\mu_{\theta i,t},\sigma_{\theta i,t}}(x_{i,t}) \tag{6}$$

This estimation is performed for different intervals $[x_{i,a}, ..., x_{i,c-1}]$. We consider disjoint adjacent future windows, from the first possible future window ($b = context\_length$), see figure 2, to the last possible future window ($b = d - predict\_length$), see figure 4, and for all time series $i \in [0, I]$.

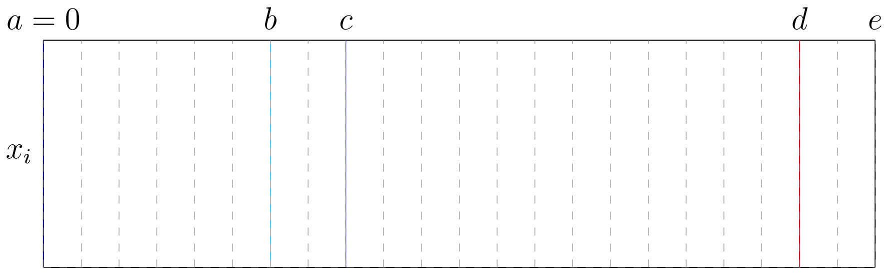


Figure 2: Feed forward models forecast : First possible interval

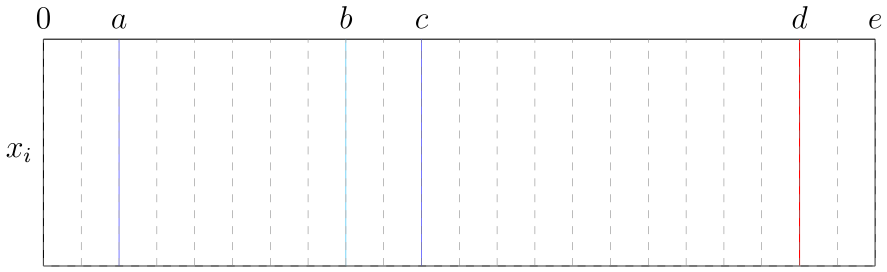


Figure 3: Feed forward models forecast : Second possible interval

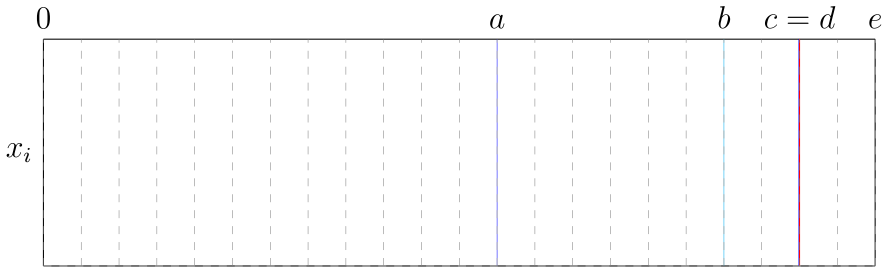


Figure 4: Feed forward models forecast : Last possible interval

Finally, the global loss of the estimation is defined as the sum, for each time series and each time step, of the loss function L with as argument the real value and the estimated distribution :

$$loss(\theta) = \sum_{i=1}^{I} \sum_{t=b-a}^{d-1} L(x_{i,t}, \phi_{\mu_{\theta i,t}, \sigma_{\theta i,t}}(x_{i,t})) \tag{7}$$

1.3.2 Second mathematical formulation : RNN models

This formulation corresponds to the way the models that can be described as RNN (*CanonicalRNN*) express the problem.

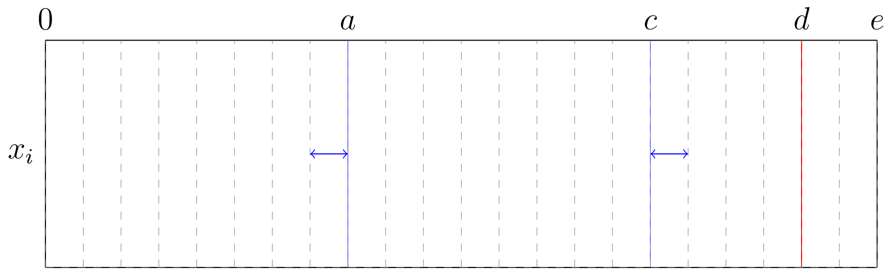


Figure 5: RNN models forecast: General case

Considering an interval $[x_{i,a}, ..., x_{i,c-1}]$ in the time series i, our goal is to estimate the distribution of window $[x_{i,a}, ..., x_{i,c-1}]$ given its hidden states $[h_{i,a}, ..., h_{i,c-1}]$. The estimated distribution is defined as :

$$p_\theta(x_{i,a}, ..., x_{i,c-1} | h_{i,a}, ..., h_{i,c-1}) \tag{8}$$

Where θ denotes the models parameters. This distribution can be factorised as :

$$p_\theta(x_{i,a}, ..., x_{i,c-1} | h_{i,a}, ..., h_{i,c-1}) = \prod_{t=a}^{c-1} p_\theta(x_{i,t} | h_{i,a}, ..., h_{i,t}) \tag{9}$$

We define f_θ, a function parametrized by θ. It hypothesis space is defined as $\{h_f : \mathbb{R}\mathbb{R} \to \mathbb{R}, h_f \in H_f\}$. It evaluation can be described as, for $t \in [a+1, c-1]$:

$$h_{i,t} = f_\theta(x_{i,t}, h_{i,t-1}) \tag{10}$$

We define g$_\theta$, a function parametrized by θ. It hypothesis space is defined as $\{h_g : \mathbb{R} \to \mathbb{R}^k, h_g \in H_g\}$. With k the number of parameters needed to defines the output distribution. We consider the case of a gaussian distribution, with $k = 2$. It evaluation can be described as, for $t \in [a+1, c-1]$ ::

$$[\mu_{\theta i,t}, \sigma_{\theta i,t}] = g_\theta(h_{i,t}) \tag{11}$$

The estimated distribution of a variable $x_{i,t}$ of the interval $(t \in [a, c-1])$ is expressed as, with $\phi_{\mu,\sigma} : \mathbb{R} \to \mathbb{R}$ a gaussian function parametrized by parameters μ and σ :

$$p_\theta(x_{i,t} | h_{i,a}, ..., h_{i,t}) = \prod_{t=b}^{c-1} \phi_{\mu_{\theta i,t}, \sigma_{\theta i,t}}(x_{i,t}) \tag{12}$$

The distribution of interval as defined in equation 8 is expressed as :

$$p_\theta(x_{i,a}, ..., x_{i,c-1} | h_{i,a}, ..., h_{i,c-1}) = \prod_{t=b}^{c-1} \phi_{\mu_{\theta i,t}, \sigma_{\theta i,t}}(x_{i,t}) \tag{13}$$

This estimation is performed for different intervals $[x_{i,a}, ..., x_{i,c-1}]$. We consider disjoint adjacent interval, from the first possible future interval ($c = context\_length + predict\_length$), to last possible future window ($c = d$), and for all time series $i \in [0, I]$.

Finally, the global loss of the estimation is defined as the sum, for each time series and each time step considered, of the loss function L with as argument the real value and the estimated distribution :

$$loss(\theta) = \sum_{i=1}^{I} \sum_{t=b-a}^{d-1} L(x_{i,t}, \phi_{\mu_{\theta i,t}, \sigma_{\theta i,t}}(x_{i,t})) \tag{14}$$

2 Second Part : GluonTS

2.1 GluonTS historical

At the time of writing this master thesis, GluonTS is a very recent toolkit, which is coherent with the fact that the deep learning forecasting field is early itself. It first release version (*v0.1.0*) is dated from March 3rd 2019. The implementation of the code has been performed on the version *v0.4.2* released on November 26th 2019 and as been updated for version *v0.5.0*, released on May 12nd 2020. These versions are/was still considered as beta versions. This implied several bugs and implementation issues. In particular some models (for example an implementation of the *DeepState* model **deepstate_paper**) failed to run correctly, and the run of "custom" models (implemented using the provided model template) is incompatible with the hybridization functionality, which provides important time optimization, because of a bug. And some functionalities that would be useful are not currently implemented, as alternative loss functions (see 4.3 section).

2.2 GluonTS Functioning

The GluonTS toolkit defines a dataset structure, containing in particular the time series. This dataset, is given to a GluonTS model (pre-implemented or implemented using the provided model template), which use the dataset information (to train and test the model). When the model is evaluated, it gives in all cases as output a distribution for each of the testing time steps, for each time series (see 1.3). The type of output distribution is defined by the user and discussed in section 2.7. Concretely, the output information is a set of randomly drawn sample of the predicted distribution, not the distributions parameters themselves. The model, if belongs to deep learning models category, must be trained before evaluation, using the chosen loss function (see section 2.5). The resulting output distributions can be evaluated using the appropriate metrics (see section 2.6).

2.3 Datasets

GluonTS interface imposes a certain structure of datasets, object used for training and testing models. There are composed of some mandatory and some optional components. '*dataset.train*' is an iterable collection of data entries used for training. Each entry corresponds to one time series. *'dataset.test* is an iterable collection of data entries used for inference. '*dataset.metadata*' contains metadata of the dataset such as the frequency of the time series, the context length, the prediction length.

GluonTS is optimized to handle multiple time series in parallel. The training time will be significantly increased if the dataset is composed of only one time series of very long length compared to a dataset composed of multiples time series of smaller length.

As mentioned in section 1.3, in general the training and testing set is composed of the same time series. The definition of the task whereabouts this master thesis is dedicated, the forecasting of wind turbines production, is not compatible with this general affirmation. In fact, the goal is to forecast the production of an energy generator using a model which couldn't be trained on the current generator production history, but on others generator production history. It is nowadays useful to test different configurations of training/testing data, including where the training data and testing data comes from the same generator(s) history. This allow us to test how the forecast quality is influenced by the variation of configuration of input data.

Input data provided by "Blacklight Analytics" is composed of different (raw) time series :

- A continuous 6 months history of a wind turbine production , in kWh (*6months-minutes.csv*)
- An history of a wind turbine production, in negative MWh (*mesure_p_gestamp.csv*)
- A 2 days history of two wind turbines production, in negative MWh (*2eol_measurements.csv*)

All the data is at a frequency of 1 minute, and is converted in positive MWh. Nowadays the configuration, because of the GluonTS behaviour optimized to handle multiple time series, these original times series are manually splitted before putting them in dataset structure. Each time series is manually splitted in certain number of time series of same size. This size is considered as a hyper parameter of the problem (*data_size*).

The configurations considered are the following :

- All the disposable times series are used as training and testing data. It is referred as "Config A"
- *2eol_measurements.csv* time series used as testing data and the other time series as training data. It is referred as "Config B".

GluonTS allow the use of ”features”, i.e. information used as input to the models additionally to time series information. Not all the models are compatible with this functionality. In our case, the different sources of information could be identified using static categorical features. For example, in the configuration A, splitted time series coming from time series *6months-minutes.csv* will have a categorical value of 0, time series from *mesure_p_gestamp.csv* the value of 1, times series from *2eol_measurements.csv* values of 2 and 3. This functionality useful if the different sources have significant differences.

The size of the prediction window *BlackLight Analytics* is interested in is 10 minutes, i.e a predict length of 10 for a frequency of 1 minute. The context length, as defined in section 1.3 is a hyper parameter of the problem indicating the number of time steps considered as input for the model network.

2.4 Quantiles

GluonTS allow selecting the values of quantiles that will be computed, to evaluate the metrics and to possibly use these information in loss function. This functionality is used in section 4.3.

2.5 Loss

GluonTS implements in it predefined models only one type of loss function. This loss function is, for the majority of the models, the inverse of the log-density of the output distribution, as defined in the equation 15. In some models, the fundamental difference in terms of architecture justify a different loss. For example the *MQRNN* and *MQCNN* models use different quantile losses as loss function.

If $\phi(x)$ is the output density distribution and y the observed value, the loss value is :

$$base\_loss(y, \phi(x)) = -ln(\phi(x = y)) \tag{15}$$

This loss express efficiently that the predicted distribution must corresponds to the real distribution. In the chapter 3 of this master thesis, the model comparison will be done using this default loss, considering that the objective pursued is the minimization of the difference between predicted density distribution and the real density distribution.

The chapter 4 of this master thesis will discuss the relevance of this affirmation and the use of this default loss.

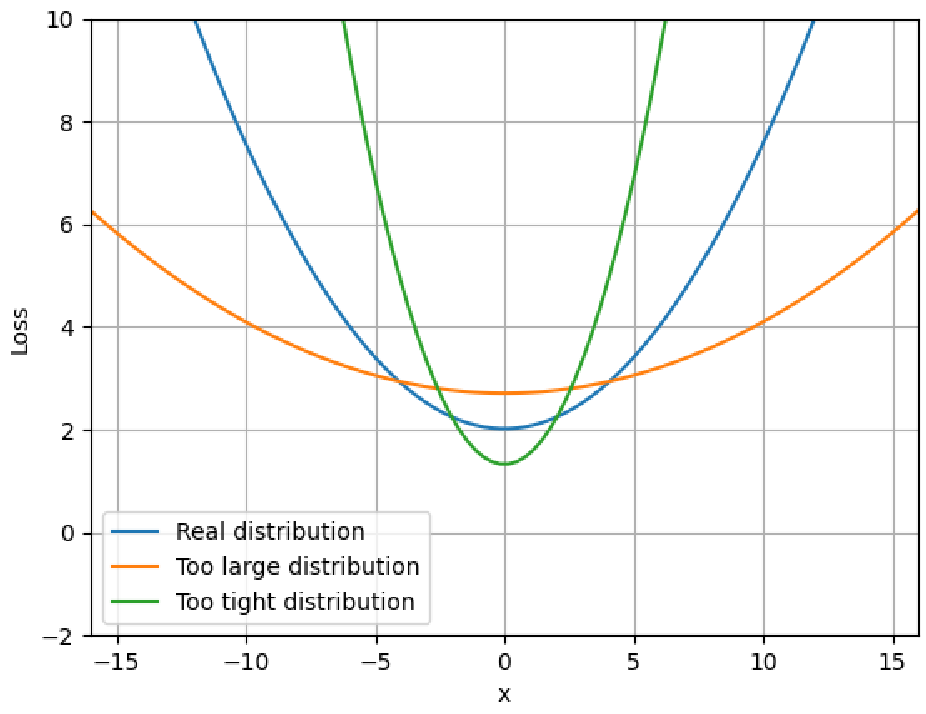


Figure 6: Comparison of the loss for a perfectly predicted distribution, a too large predicted distribution and a too tight distribution, with x the difference between the observed value and the median of the predicted distribution

2.6 Metrics

Concerning the evaluation of the results, GluonTS provides a module using the trained model and testing data to provide quantitative results, as metrics ("Aggregate", for all time series, or "Item", for each series separately). Metrics proposed by GluonTS includes *MSE, MSIS, RMSE*, and other classical measures.

All metric function takes as argument N randomly drawn sample values of the predicted probability distribution ($[x_1, ..., x_N]$) and one observed value y. For example the MSE metric function is defined as :

$$MSE(y, [x_1, ..., x_N]) = \frac{1}{N}\sum_{i=0}^{N}(y - x_i)^2 \tag{16}$$

"Item" metric value is the mean of metric value for each time step of the prediction window. "Aggregate" metric value is the mean of "Item" metrics values for each time series.

In chapter 3, as we use the default loss, metric function must correspond to this loss function, as it has been defined in section 2.5.

The problem is that the metric function cannot be equivalent to loss function, as these are functions with different inputs, the loss function taking as input the predicted distribution instead of randomly drawn point sample. Nowadays, in fine, metrics like *MSE, MSIS, RMSE* expresses the same goal than the log-negative likelihood of the loss function, i.e. the minimization of the difference between predicted and real distribution. A high value of one of these metrics is proportional to a high value of the loss. The metric chosen for the first model comparison in chapter 3 is *MSE*.

2.7 Distribution

GluonTs proposes different types of output distribution. In most of the models the output is technically a vector of values which is transformed to a vector of distribution parameters and put in a distribution object. In current implementation, three different output distribution are functional. *Gaussian*, *Laplace* and *PiecewiseLinear*. *Student* has been rejected because the quantiles of the distribution are not obtainable. *Uniform* has been rejected because of loss problems in execution. Other options are available, as multiple kernel gaussian. The GluonTS models are mandatory to output an object containing point sample of the distribution, not the distribution itself (parameters values). Considering that the parameter information could also be useful, as asked by the client, a modification done in custom models allows saving these information.

Figure 7: Comparison between Gaussian distribution and Laplace distribution

Figure 8: Comparison between Gaussian distribution and PiecewiseLinear distribution

2.8 Models presentation

All the pre-implemented deep learning models that have been tested are presented here. The models individual hyperparameters are mentioned in corresponding section. The results for different values of hyperparameters are studied in section 3.3.

2.8.1 Simple

A manually implemented model which uses a simple 2 fully connected layers neural network. The individual hyperparameter is the number of cells in layers.

2.8.2 FeedForward

A MLP (multi-layer perceptron) model. The individual hyperparameter is the number of hidden nodes in each layer.

2.8.3 Recurrent Neural Network

A model which uses a recurrent neural network **??**. The individual hyperparameters are the number of layers and number of cells of the network

2.8.4 DeepAr

Implementation of DeepAr estimator, a RNN based model, close to the one described in paper *"DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks"* **deepar_paper**. The individual hyperparameters are the same than for CanonicalRNN.

2.8.5 Deep Factors

Implementation of the 2019 ICML paper *"Deep Factors for Forecasting"* **deepar_factor**. Uses a global RNN model to learn patterns across multiple related time series and an arbitrary local model to model the time series on a per time series basis. In the current implementation, the local model is a RNN (DF-RNN). The individual hyperparameters are the number of units per hidden layers and the number of layers in the global model, the number of units and layers in the local model (not studied) , the number of global factors.

2.8.6 Gaussian Process

Model using Gaussian Processes (GP) **gp_paper**. Each time series has a GP with its own hyper parameters. There are no hyperparameters to tune.

2.8.7 NPTS

Implementation of the Non-Parametric Time Series Forecaster **npts_paper**, which falls into the class of simple forecasters that use one of the past observed targets as the forecast for the current time step. It randomly samples a past time index as the prediction for the time step T (auto regressive model, which predict step by step). There are no hyperparameters to tune, but it exists some variants.

2.8.8 MQCNN

Discriminative Sequence to Sequence model constructed using the *SeqtoSeq* framework of GluonTS to reproduce the model of the paper *"A Multi-Horizon Quantile Recurrent Forecaster"* **mqcnn_paper**. Sequence to sequence models are composed of two parts. The encoder network, that reads the training window of the time series and encodes information about the sequence in a latent state. And the decoder network, which generates the forecast by combining the latent information with the features in the prediction range. In MQCNN, the encoder is a Convolutionnal Neural Network and the decoder an MLP. Unlike other presented models, the output is not technically a distribution but the quantiles itself, that are point predicted values obtains by optimizing the corresponding quantile loss. The individual hyperparameter is the dimension of the MLP decoder. Considering the fundamental difference in the intern functioning, it not seems to be possible to modify the loss, see discussion in 4.3 section.

2.8.9 MQRNN

Same as MQCNN but with a Recurrent Neural Network encoder instead.

2.9 Others

Other models, recently added to GluonTS, will be added. At least 3 (Wavenet, DeepVar, NBEATS)

3 First Models Comparison

3.1 Testing protocol definition

Each comparison is presented using as element of comparison the metric *MSE*, as defined in section 2.6. The models uses the default loss as defined in section 2.5

The plot results are all presented with the same values of quantiles, the quantile $quantile(0.99)$ and the respective $quantile(0.01)$ but also the $quantile(0.9)$, $quantile(0.1)$ and the median. Before the comparison between different models, we need to compare the impact of the values of the different hyperparameters of the problem. The hyperparameters of the problem are :

- The size of time series
- The learning rate of the model training
- The number of epochs of the training
- The output distribution, defined in section 2.7
- The individual hyperparameters of the different models

Each comparison section is interested in one particular hyperparameter of the problem. In general the other hyperparameters takes default values, as given by GluonTS.

3.2 Global Hyperparameters comparison

3.2.1 Different dataset configurations

There is no quantitative comparison to do, considering that the goal is not to find the configuration which gives the better results. The quantitative results are not significant, except noticing that the results are indeed influenced by the configuration. In general the following section will consider configuration A as default, but if necessary other configuration will be tested.

3.2.2 Time series size

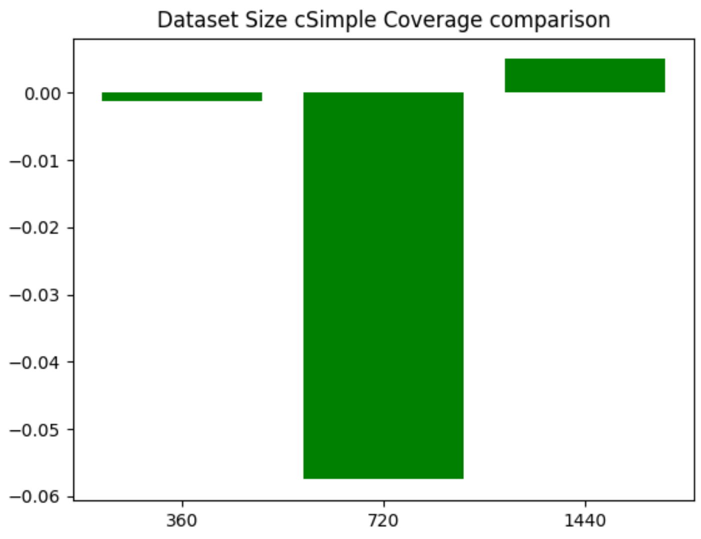


Figure 9: Comparison of different time series size (Model: Simple (Default parameter values), Epochs = 100, $\alpha = 0.9$, Distribution = Gaussian, Config A)

3.2.3 Learning rate

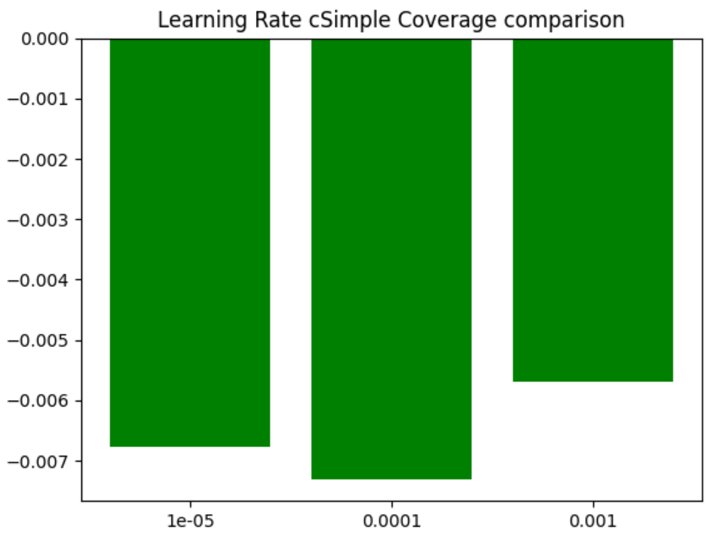


Figure 10: Comparison of different learning rate (Model: Simple (Default parameter values), Epochs = 100, α = 0.9, Distribution = Gaussian, Config A)

The default learning rate in GluonTs for all deep learning models is 1^{-3} Learning rate as a great impact on the results.

3.2.4 Epochs

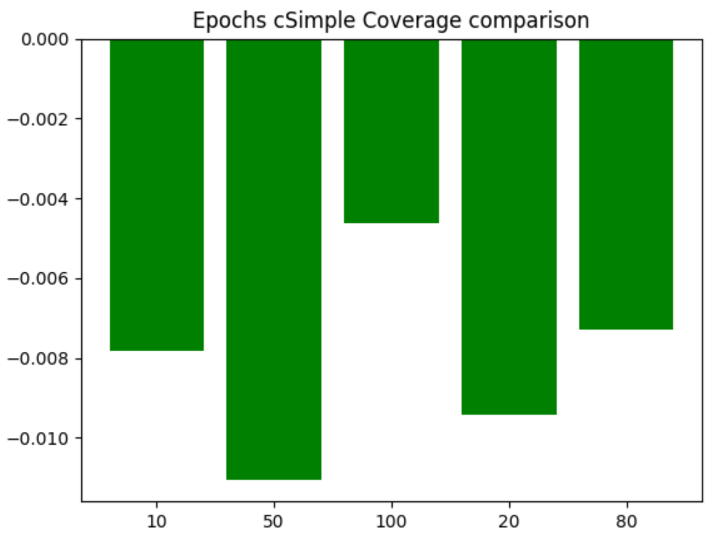


Figure 11: Comparison of different epochs number (Model: Simple (Default parameter values), α = 0.9, Distribution = Gaussian, Config A)

All the deep learning models implemented must be trained a certain number of epochs. The default number is 100 epochs. This value is a good choice (better bandwidth of the comparison), as are values slightly inferior (which have better Coverage).

3.2.5 Output Distribution

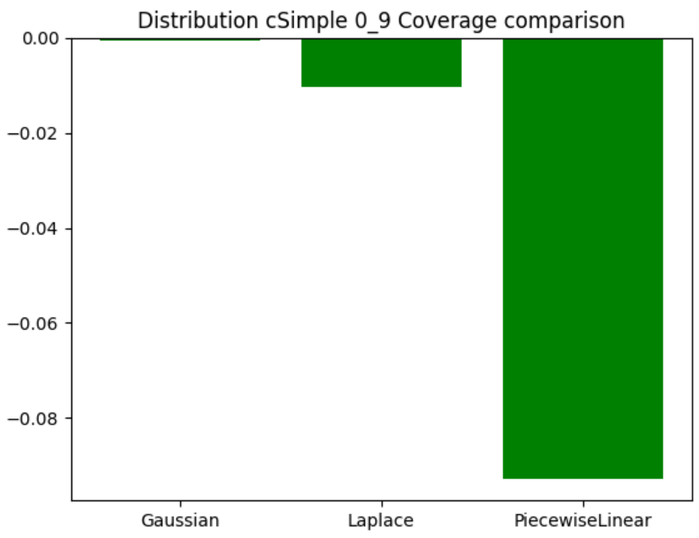


Figure 12: Comparison of different output distribution (Model: Simple (Default parameter values), Epochs = 100, α = 0.9, Config A)

3.3 Different models parameters comparison and results

The different models are described in the 2.8 section

3.3.1 Simple

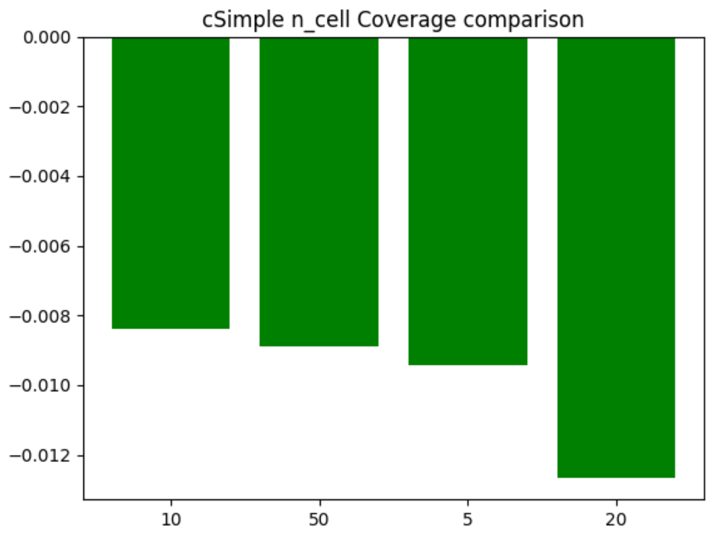
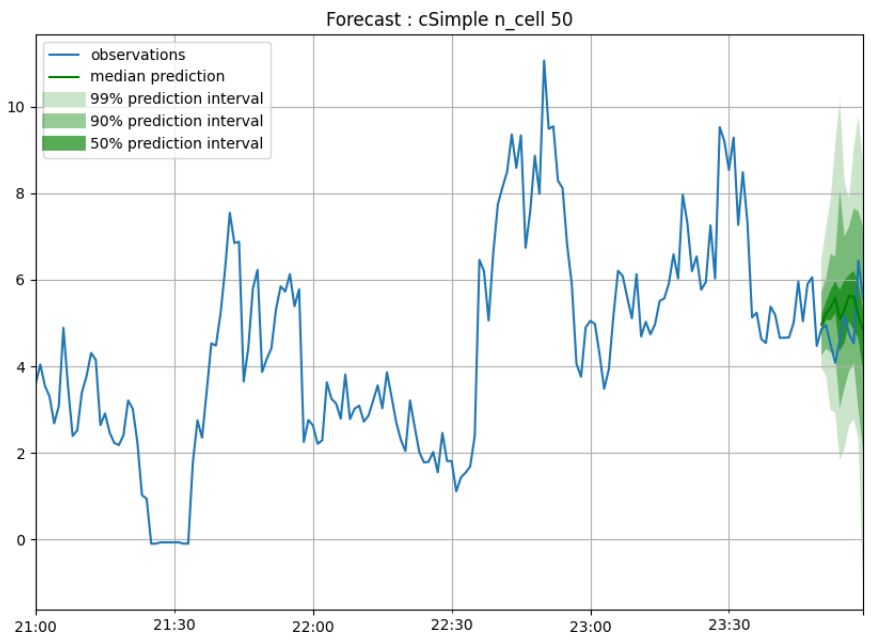


Figure 13: Comparison of different *n_cell* values for Simple model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)



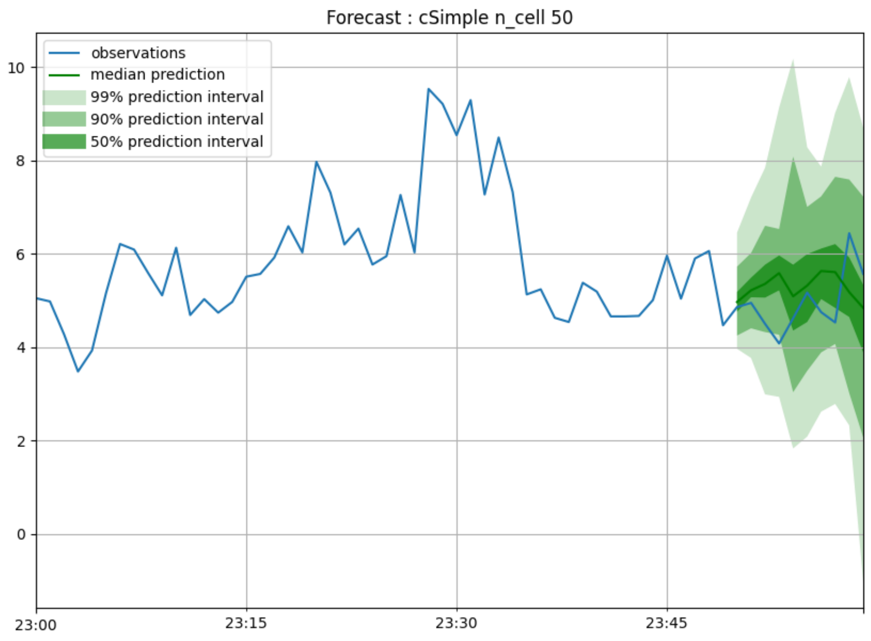


Figure 14: Forecast result of Simple model at 3 hours and 1 hour scale ($n\_cell = 50$ Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

3.3.2 Simple Feed Fordward

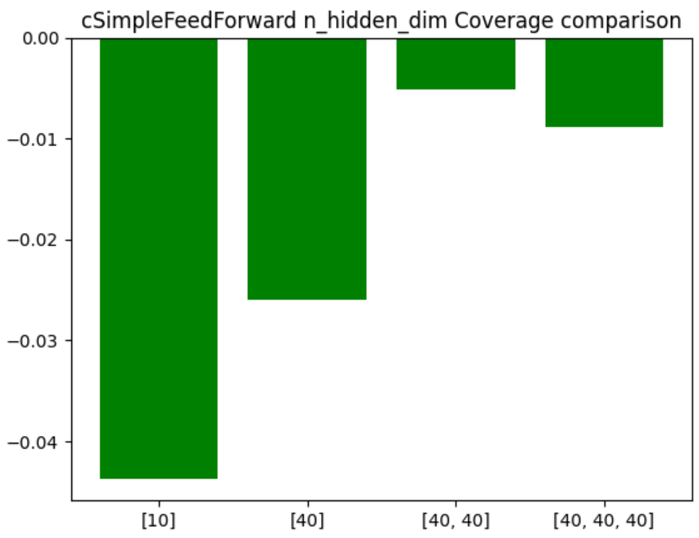
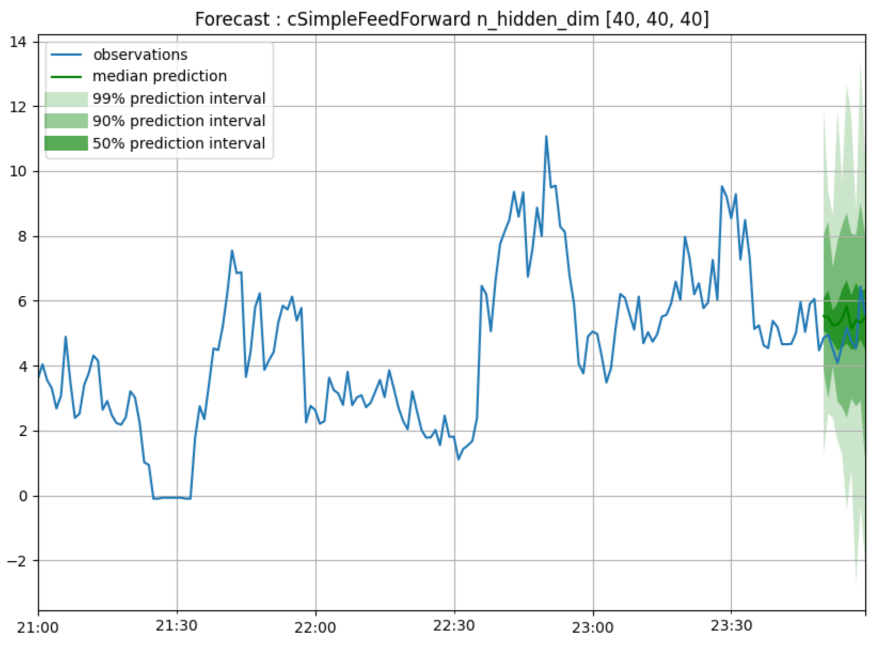


Figure 15: Comparison of different *n_hidden_dim* values for Simple FeedForward model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

40, 40]/180.png



40,

40]/60.png

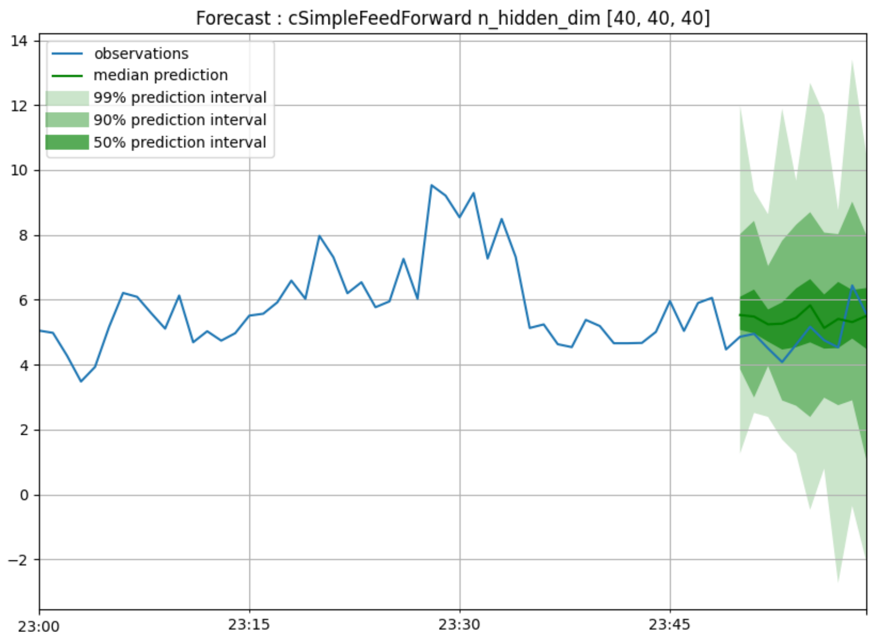


Figure 16: Forecast result of Simple FeedForward model at 3 hours and 1 hour scale ($n\_hidden_dim = [40, 40, 40]$ Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

3.3.3 Canonical RNN

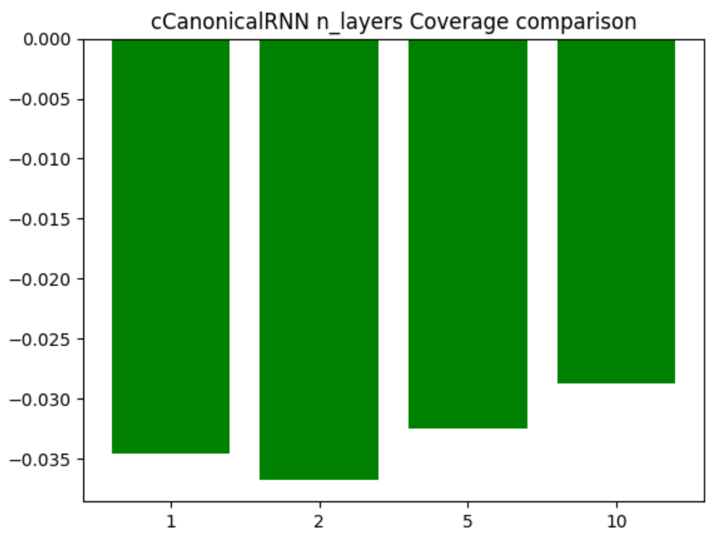
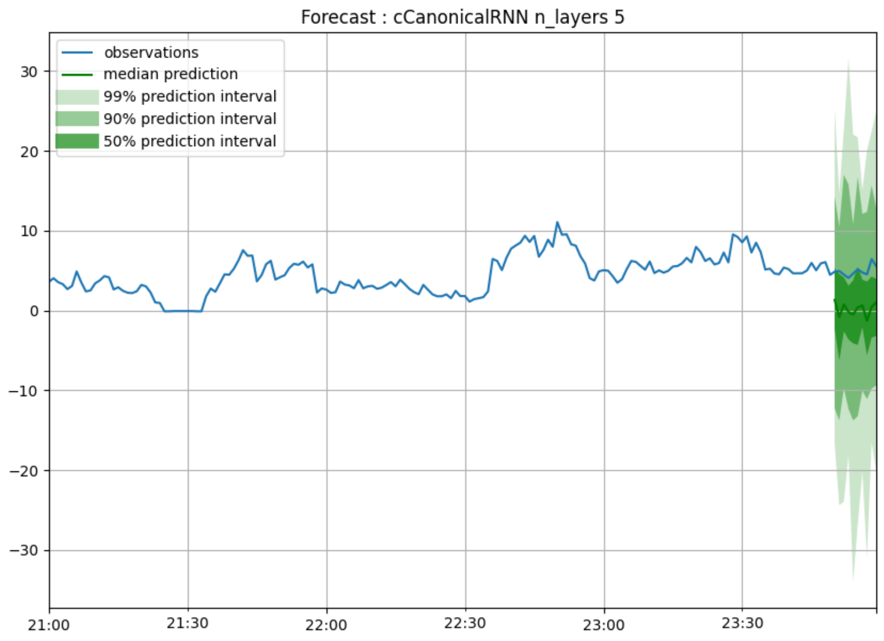


Figure 17: Comparison of different n_layers values for Canonical RNN model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)



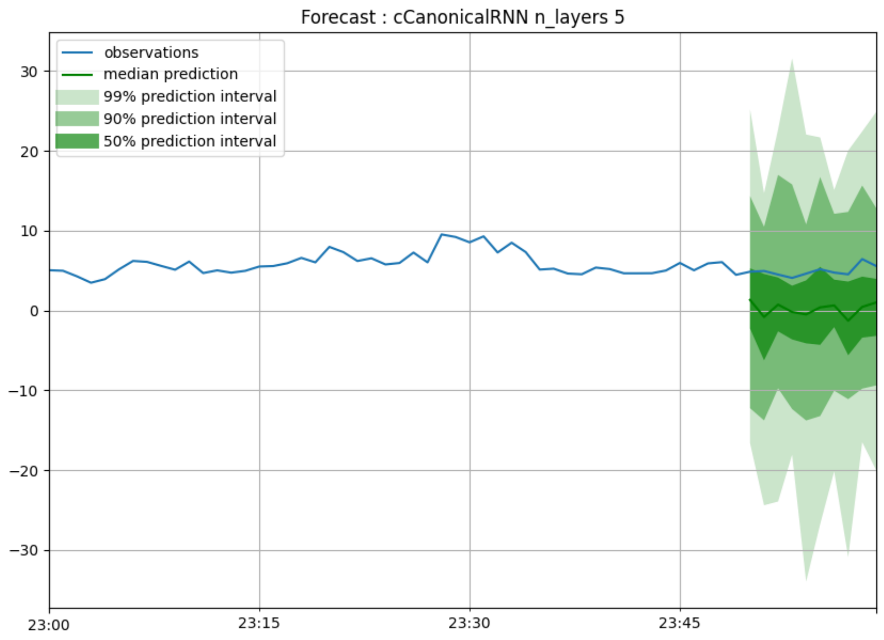


Figure 18: Forecast result of Simple FeedForward model at 3 hours and 1 hour scale ($n\_layers = 5$ Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

3.3.4 Deep AR

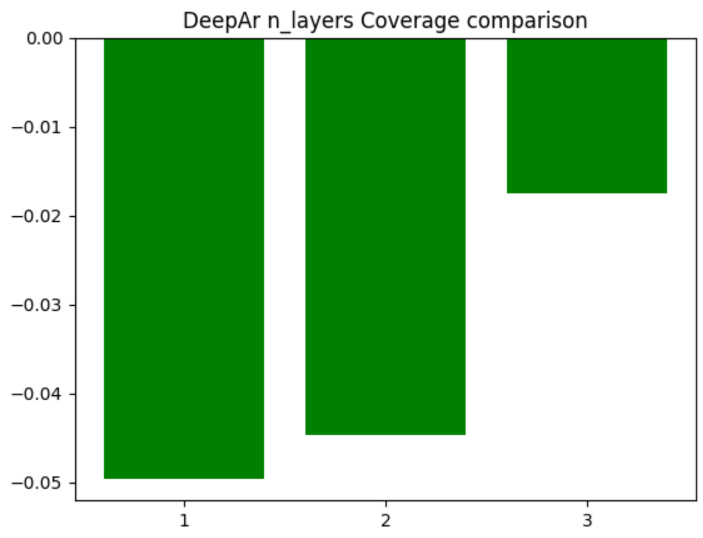


Figure 19: Comparison of different *n_layers* values for Deep AR model (Epochs = 100, Distribution = Gaussian, Config A)

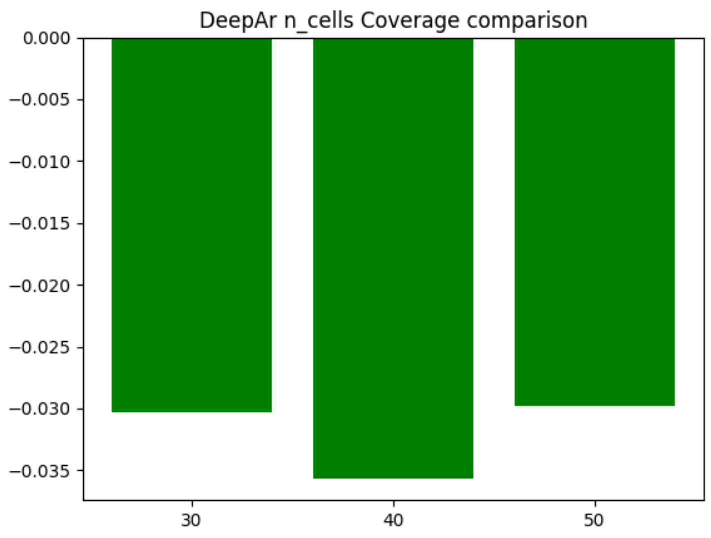
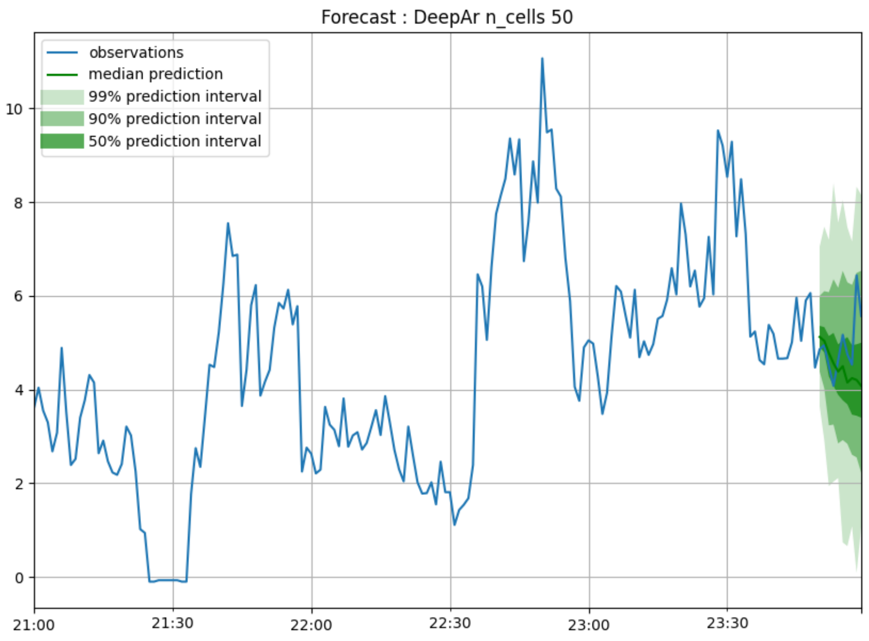


Figure 20: Comparison of different *n_cells* values for Deep AR model (Epochs = 100, Distribution = Gaussian, Config A)



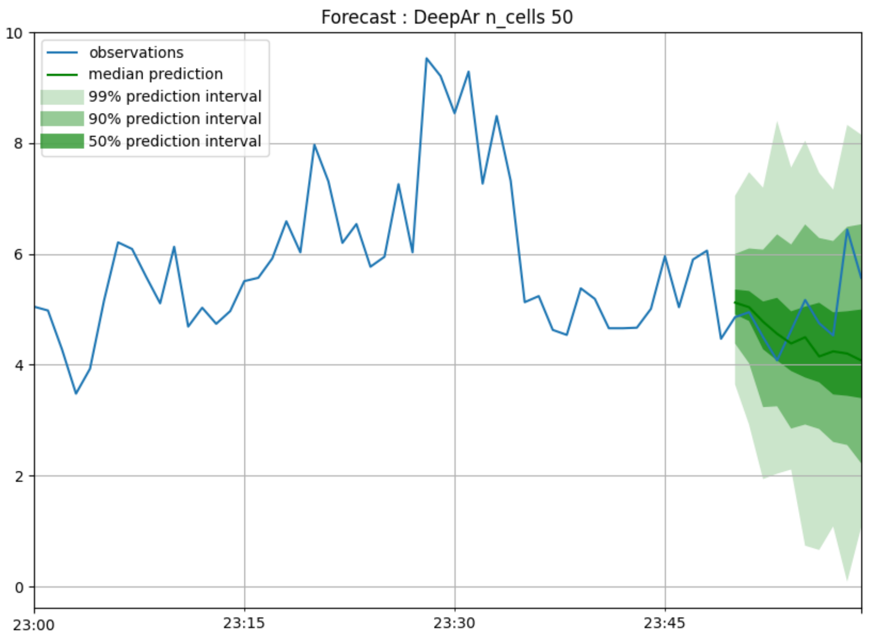


Figure 21: Forecast result of Deep Ar model at 3 hours and 1 hour scale ($n\_layers = 2$, $n\_cells = 50$, Epochs = 100, Distribution = Gaussian, Config A)

3.3.5 Deep Factor

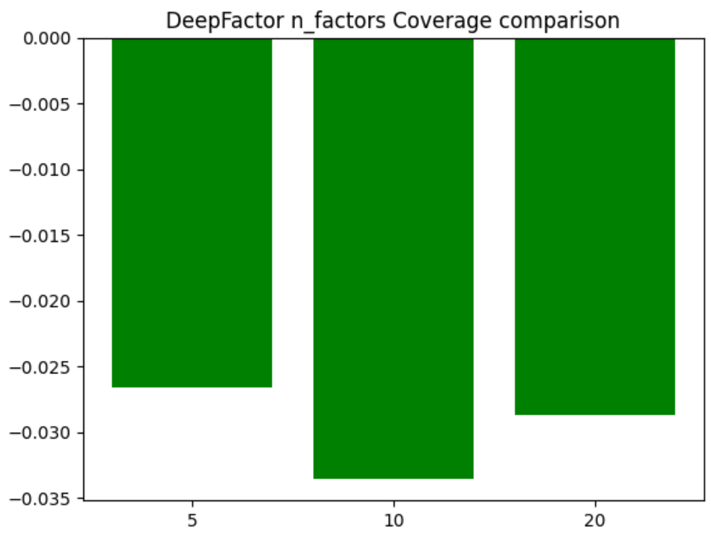


Figure 22: Comparison of different *n_factors* values for Deep Factors model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

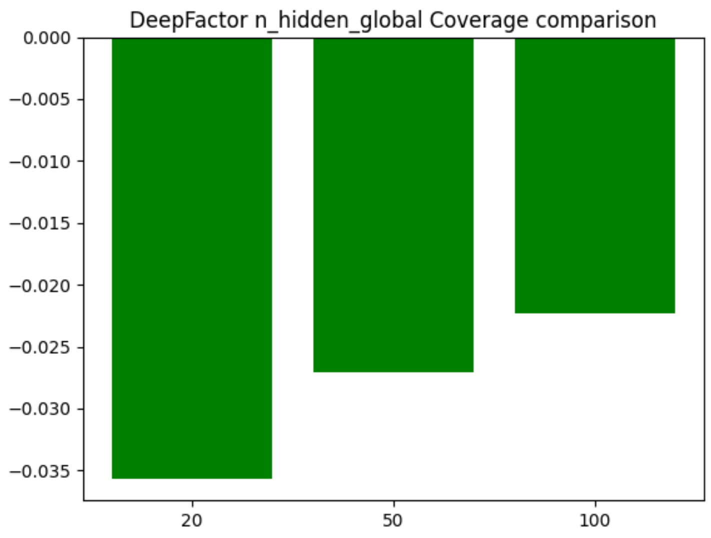


Figure 23: Comparison of different *n_hidden_global* values for Deep Factors model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

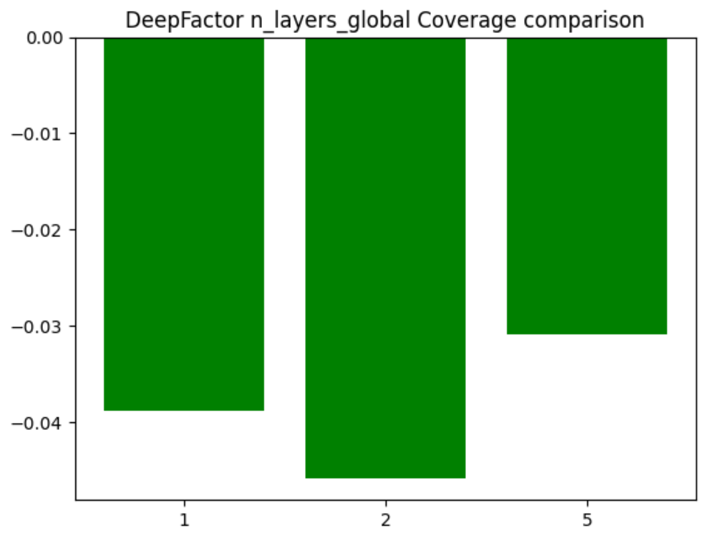
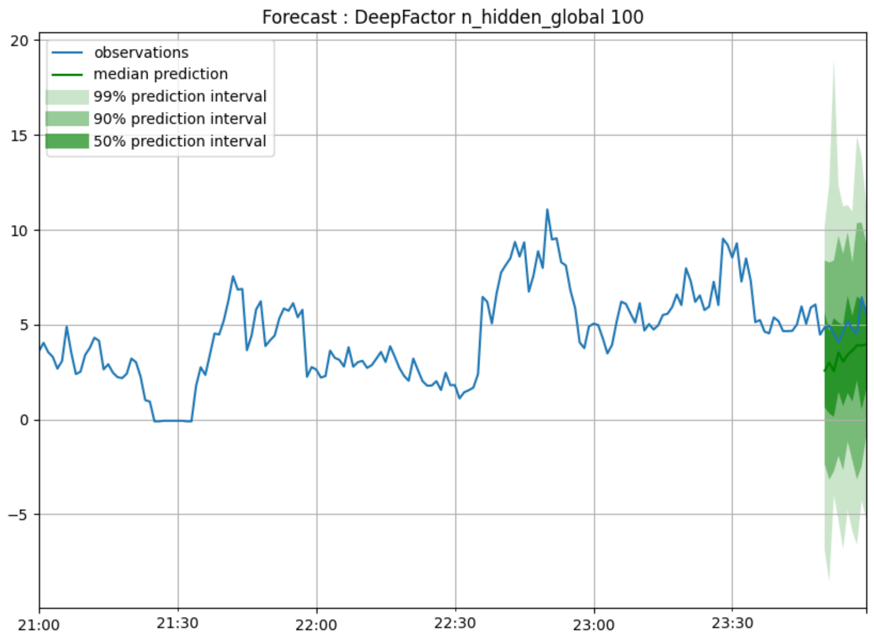


Figure 24: Comparison of different n_layers_global values for Deep Factors model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)



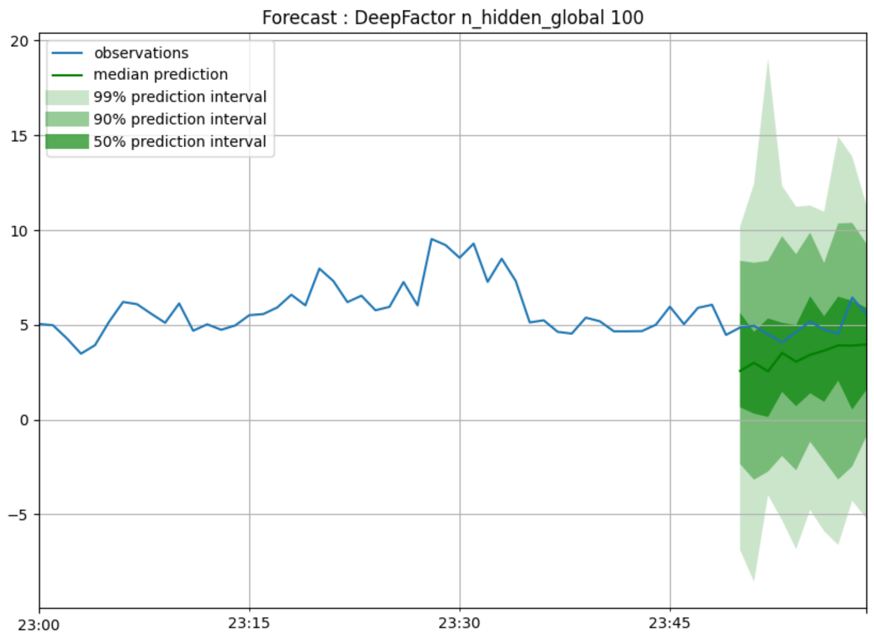


Figure 25: Forecast result of Deep Factors model at 3 hours and 1 hour scale ($n\_factors = 5$, $n\_hidden\_global = 100$, $n\_layers\_global = 1$ Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

Contrary to another model, different hyperparameters of Deep Factors had a relatively low impact on the quality of the model.

3.3.6 MQCNN

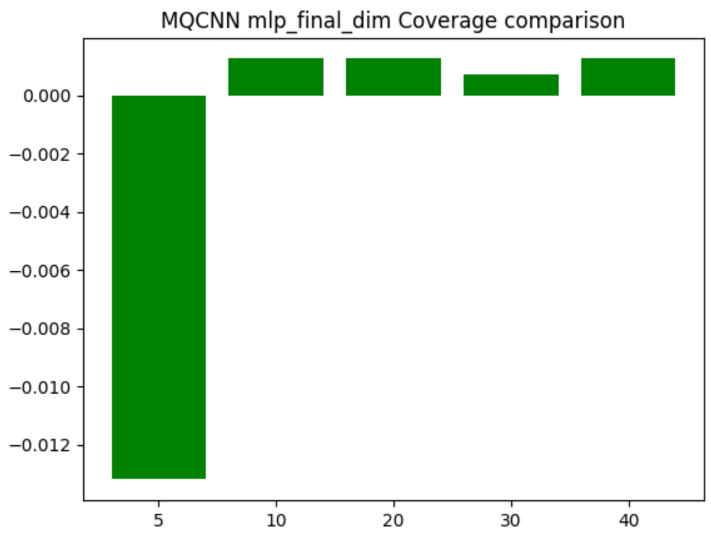
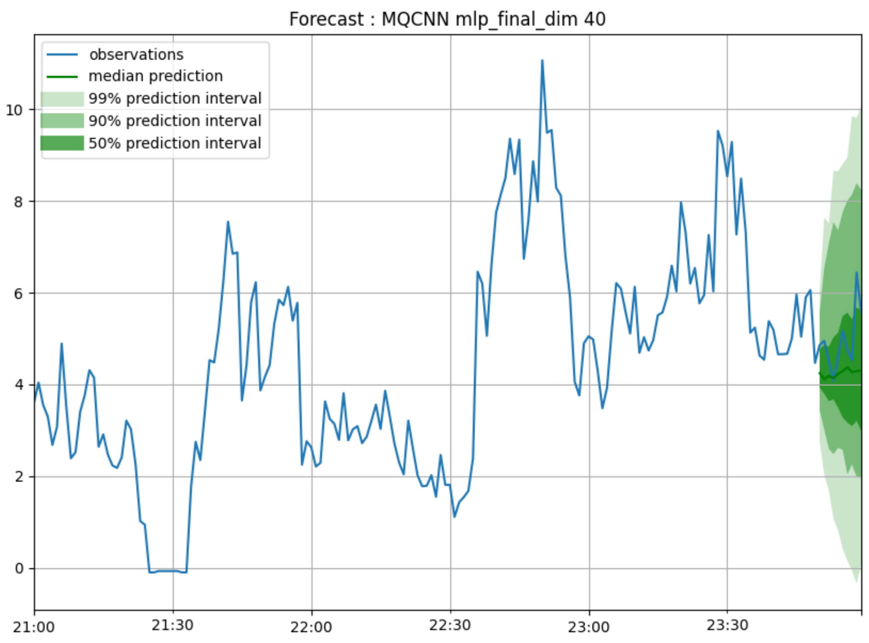


Figure 26: Comparison of different *mlp_final_dim* values for MQCNN model (Epochs = 100, Distribution = Gaussian, Config A)



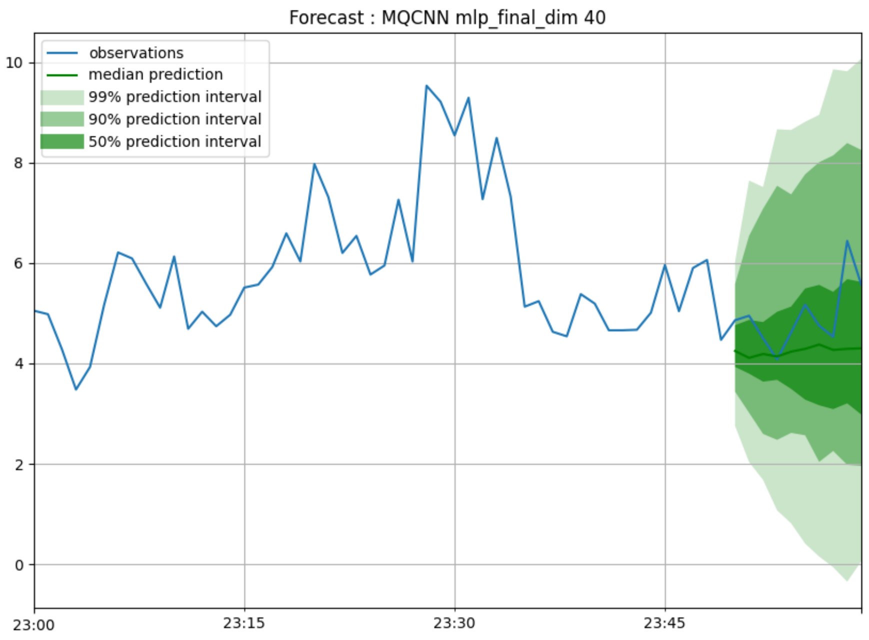


Figure 27: Forecast result of MQCNN model at 3 hours and 1 hour scale (mlp_final_dim = 40, Epochs = 100, Distribution = Gaussian, Config A)

3.3.7 MQRNN

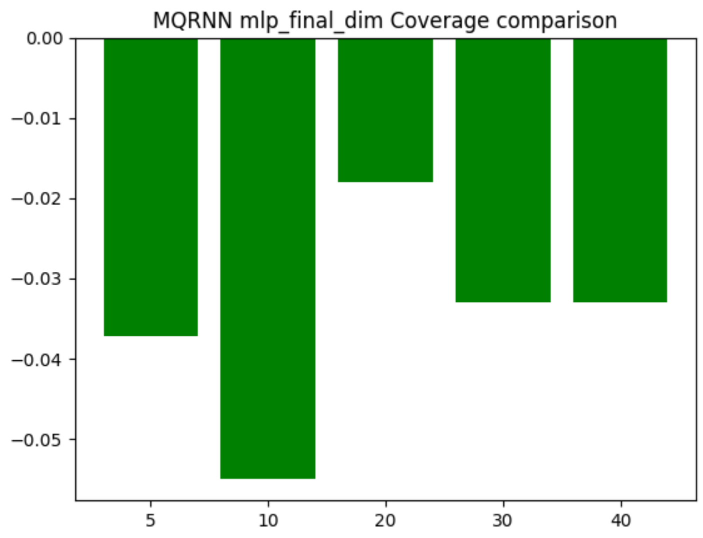
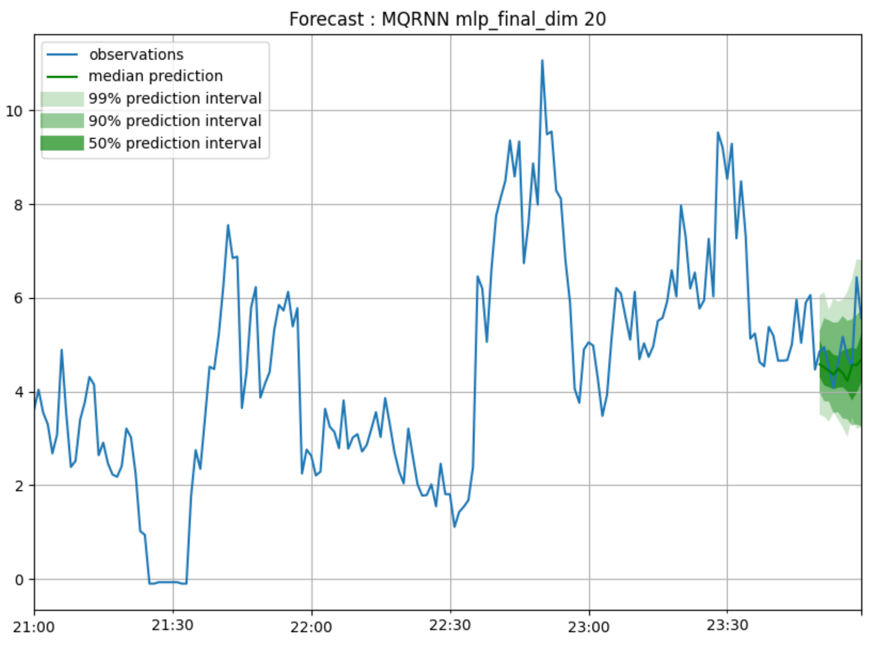


Figure 28: Comparison of different *mlp_final_dim* values for MQRNN model (Epochs = 100, Distribution = Gaussian, Config A)



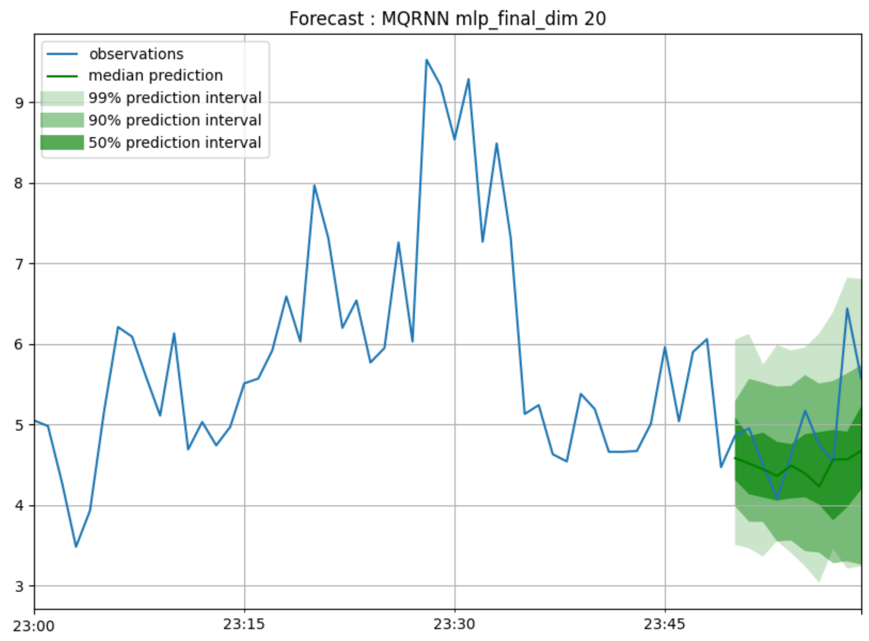
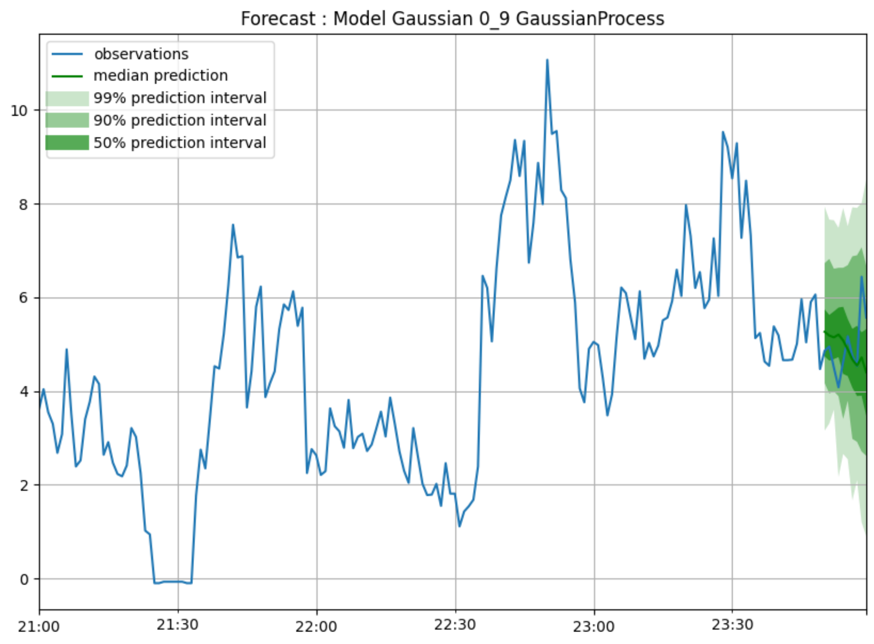


Figure 29: Forecast result of MQRNN model at 3 hours and 1 hour scale ($mlp\_final\_dim = 20$, Epochs = 100, Distribution = Gaussian, Config A)

3.3.8 Gaussian Process



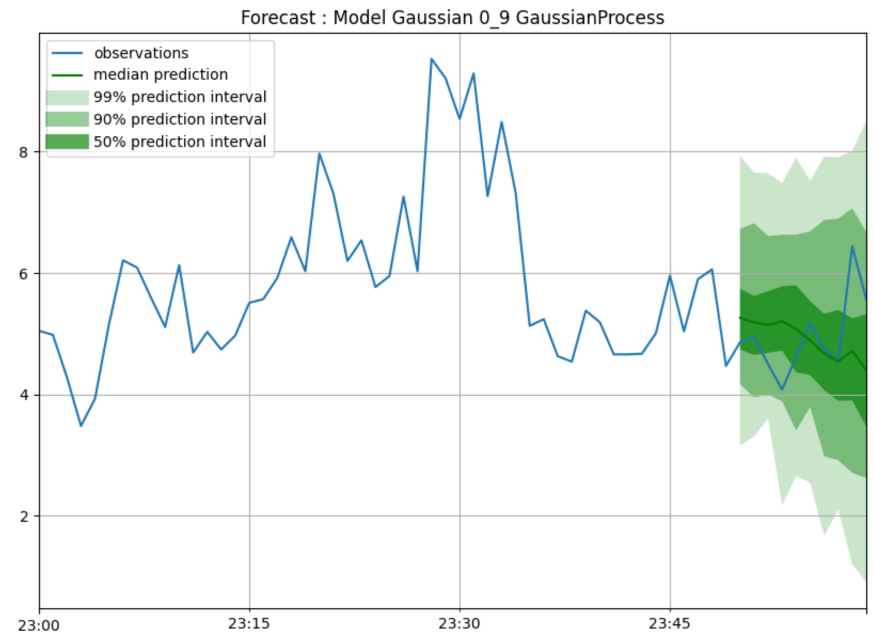
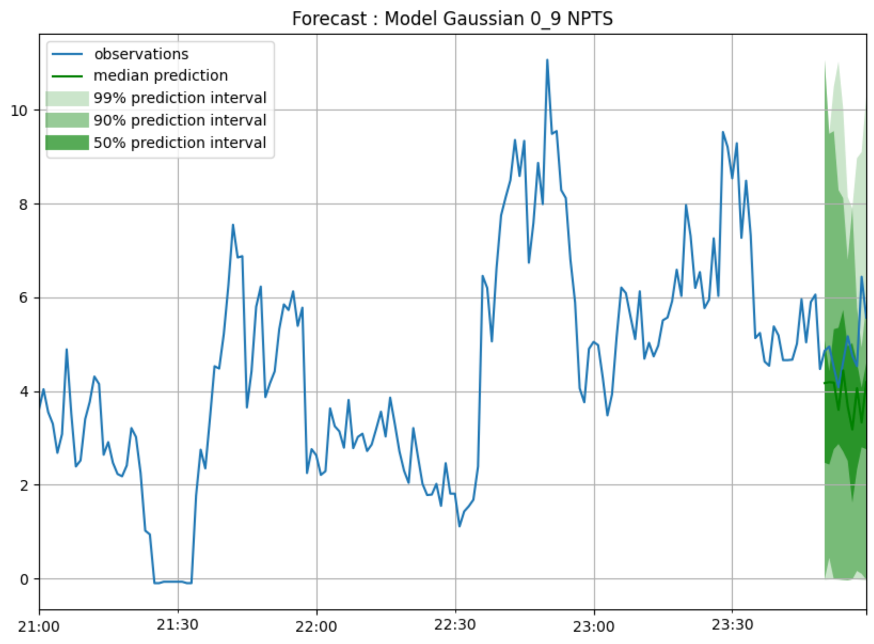


Figure 30: Forecast result of Gaussian Process model at 3 hours and 1 hour scale (, Epochs = 100, Distribution = Gaussian, Config A)

3.3.9 NPTS



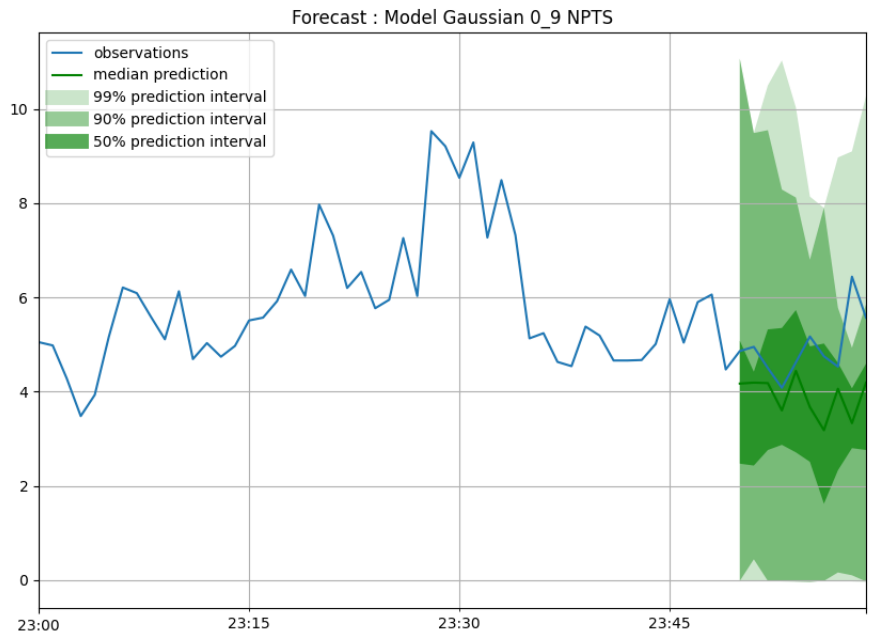
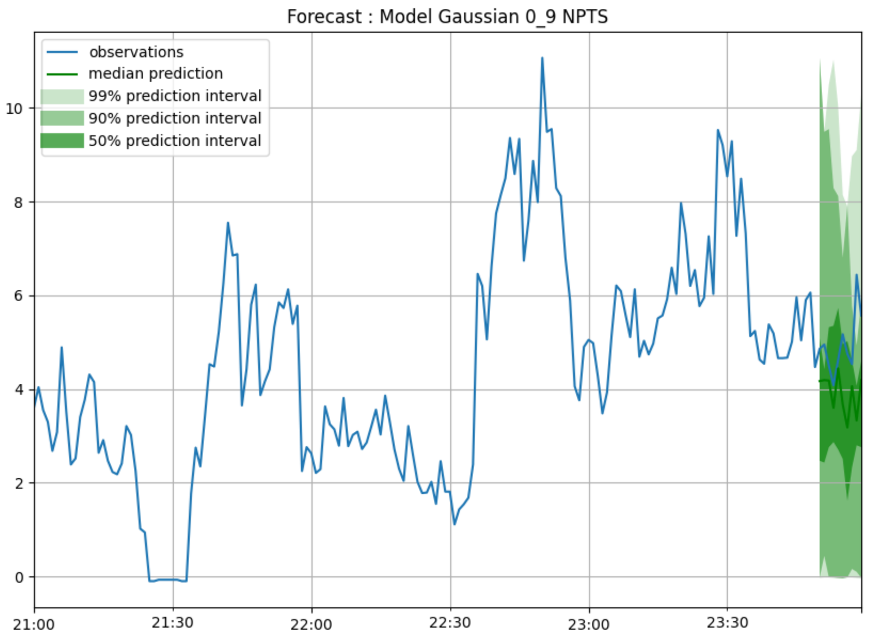


Figure 31: Forecast result of NPTS model at 3 hours and 1 hour scale (Epochs = 100, Config A)

3.3.10 ETS



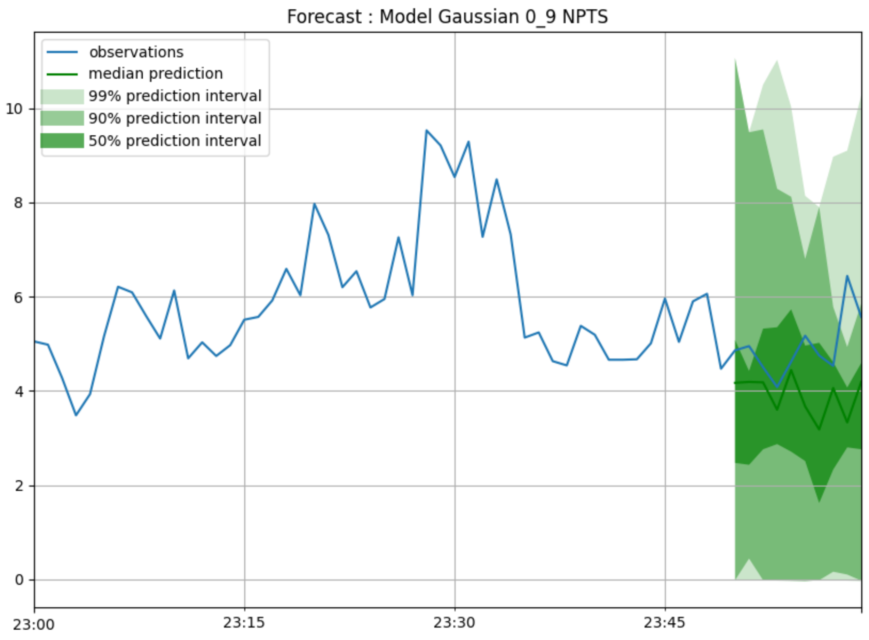


Figure 32: Forecast result of ETS model at 3 hours and 1 hour scale (Epochs = 100, Config A)

3.4 Global comparison

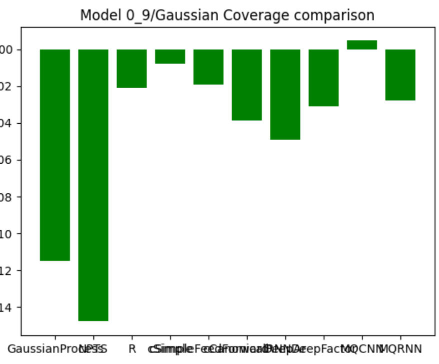


Figure 33: Comparison of different models (Epochs = 100, Distribution = Gaussian, Config A)

4 Fourth Part : Goal discussion

4.1 Discussion about forecasting goal

Now that the comparison of the models as been done with as goal the minimization of the prediction error, as defined in sections concerning the loss 2.5 and the metrics 2.6, we can interrogate the pertinence of this ”by default” goal in the particular context the forecast models must be used.

The third step of the stochastic decision process described in section 1.1 is, more than everything, interested in a forecasting that gives a precise estimation of the value of production for which there is a very low probability (value that could be discussed) that the production in the given time window outnumber this value. That value will be called the ”security limit” and will be used to determine, knowing what is the maximum acceptable production value, if the generator limits must be modified. In other terms, considering that the model output is a probability distribution, the goal fixed is not to minimize the difference (in any ways to define it) between the predicted and observed time series but to minimize the difference between the predicted ”security quantile”, i.e the $quantile(1 - e)$ of the predicted distribution, knowing that e is the probability of error considered as acceptable, and the real security quantile (the quantile of the real probability distribution of the source). A too low predicted security quantile signify that the risk of production oversupply is underestimated, leading to situations where the system will estimate that the network is safe wrongly. Inversely a too high predicted security quantile signify that the risk is overestimated, leading to situations where the production limit will be decreased unnecessarily low. This situation is nevertheless less problematic than the first one. The minimization must be for all time steps of the prediction range.

This objective is not disconnected from the classical objective consisting in minimizing the difference between the predicted and observed time series. If the predicted distribution is close to the real distribution, the predicted security quantile tends to be close to the real security quantile.

4.2 New metric

Knowing the goal pursued (4.1), the metric must indicates if the predicted quantiles are over/under-estimated. The defined objective (4.1) cannot be expressed literally as the quantiles of the real distribution are unknown. The proposed metric, *Coverage* uses an implemented metric, *coverage*. This metric is defined as, taking as argument N randomly drawn sample values of the predicted probability distribution ($[x_1, ..., x_N]$), one observed value y, q the quantile considered and $quantile_q([x_1, ..., x_N]$ a function giving the quantile q value for the approximated distribution constructed on the samples $[x_1, ..., x_N]$:

$$coverage(q, y, [x_1, ..., x_N]) = \frac{1}{N} \sum_{i=0}^{N} \mathbb{I}(y < quantile_q([x_1, ..., x_N]) \tag{17}$$

This metric gives the observed probability having a value below the quantile q.

The Coverage metric is defined as, for e is the probability of error considered as acceptable :

$$Coverage(q, y, [x_1, ..., x_N]) = coverage(1 - e, y, [x_1, ..., x_N]) - (1 - e) \tag{18}$$

The Coverage of the predicted distribution must tends to 0 to achieve the goal. For the upper quantile, if coverage(x) > x, the quantile is too high (the window is too large) and if coverage(x) < x the quantile is too tight.

This metric is not sufficient to evaluate the quality of a model prediction. The model could have a good Coverage value because the predicted probability distribution is very spread out.

Another useful indicator is how much the output probability distribution is spread out. Between two models with a Coverage of 0, the best model is the one which gives the thinner probability distribution, or in other terms, the lesser difference between the security quantile and the median. A proposed metric is the mean of the difference between the $quantile(1 - e)$ and $quantile(e)$ values. We could use $quantile(0.5)$, the median, instead of $quantile(e)$. For a time series :

$$Bandwidth(y, [x_1, ..., x_N]) = \frac{1}{T} \sum_{t=0}^{T-1} quantile_{1-e}([x_1, ..., x_N]) - quantile_e([x_1, ..., x_N]) \tag{19}$$

4.3 Custom Loss

Considering that, as we have seen in section 4.1, the model training must follow an non-traditional goal, the loss function must be discussed. The loss must take account if the predicted quantiles are over/under-estimated.

The loss could be equivalent to the metric Coverage described in section 4.2.

We could also use a different loss that penalize values of x bigger than $quantile(1-e)$ This proposed loss, which is non binary on the contrary to the metrics, is the following, for $\phi(x)$ is the output density distribution and y the observed value :

$$alt\_loss(y, \phi(x)) = e^{y - quantile_{1-e}(\phi(x))} \tag{20}$$

The loss increase exponentially if x superior to the security quantile, and is remains low if not.

The only way to use other loss than the default loss in GluonTS is to define custom models, based on the original but with intern modifications to change loss function.

4 custom models (one original, *"Simple"*, and the copy of the pre-implemented *"SimpleFeedForward"* and *"CanonicalRNN"*, *DeepAr*) are implemented.

Some models are different and cannot use another loss, some cannot use the defined custom loss because of implementation problematics, notably because in the current version of GluonTS the use of a custom model implies that the hybridisation, important part of the implementation optimization, cannot be used, resulting in very important training time.

The loss implemented in these custom model is the following :

$$custom\_loss(x, \phi) = base\_loss(x, \phi) + \alpha * alt\_loss(x, \phi) \tag{21}$$

With α an hyper parameter, It value is a subject of a study in section 5.2.1.

5 Second Models Comparison

5.1 Testing protocol definition

Each comparison is presented using as element of comparison the metrics *Coverage* and *bandwidth*, as defined in section 4.2 The models uses the custom loss as defined in section 2.5 if this loss can be implemented.

The plot results are all presented with the same values of quantiles, the quantile $quantile(0.99)$ and the respective $quantile(0.01)$ but also the $quantile(0.9)$, $quantile(0.1)$ and the median. Before the comparison between different models, we need to compare the impact of the values of the different hyperparameters of the problem. The hyperparameters of the problem are :

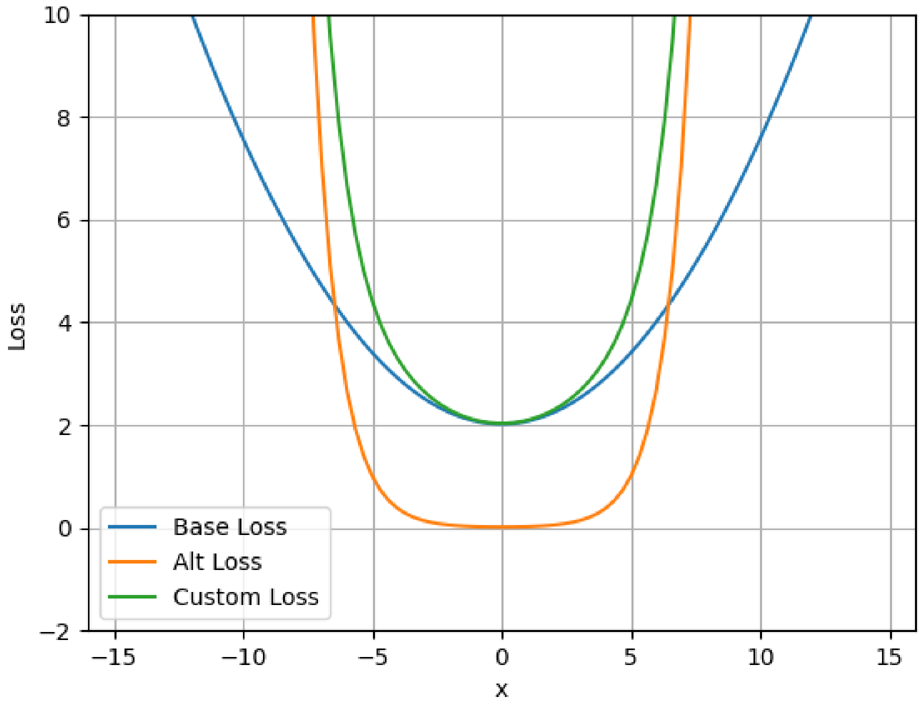


Figure 34: Comparison of Base Loss, Alt Loss and Custom Loss, with x the difference between the observed value and the median of the predicted distribution

- The alpha value, as defined in section 4.3
- The learning rate of the model training
- The number of epochs of the training
- The output distribution, defined in section 2.7
- The individual hyperparameters of the different models

Each comparison section is interested in one particular hyperparameter of the problem. In general the other hyperparameters takes default values, as given by GluonTS.

5.2 Global Hyperparameters comparison

5.2.1 Alpha

The value of α, defined in section 2.5 influence the model behaviour. Bandwidth value decrease with alpha, as Coverage increase with alpha, which tends to show the efficiency of the custom loss.

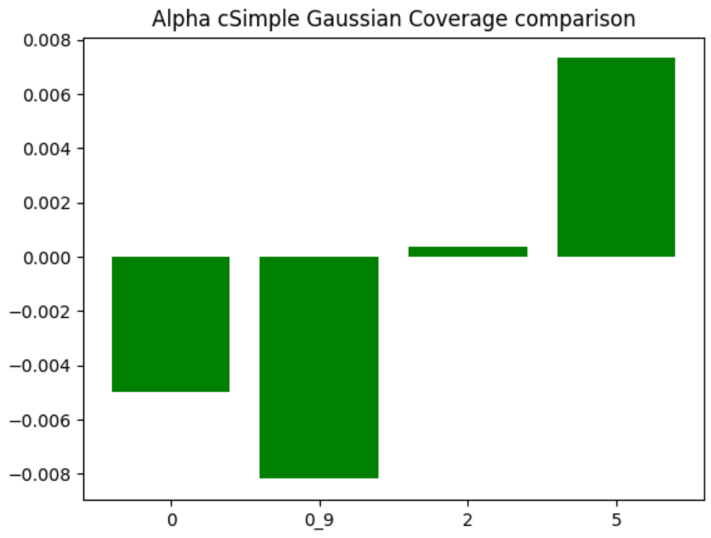


Figure 35: Comparison of different α values (Model: Simple (Default parameter values), Epochs = 100, Distribution = Gaussian, Config A)

5.2.2 Time series size

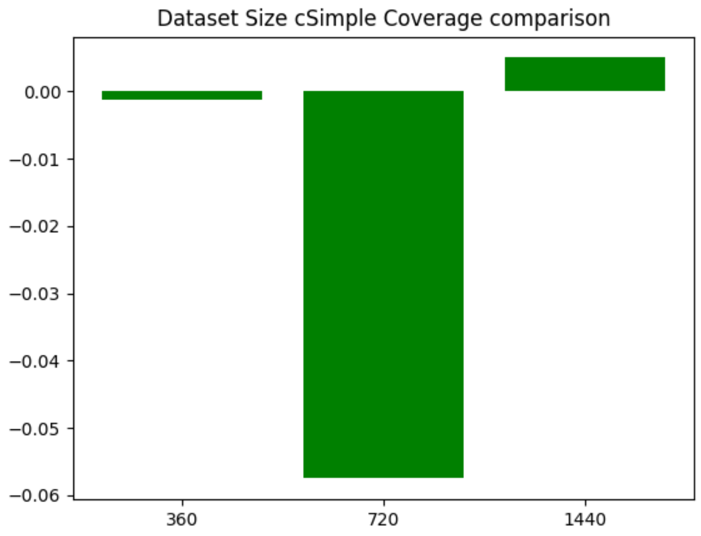


Figure 36: Comparison of different time series size (Model: Simple (Default parameter values), Epochs = 100, $\alpha = 0.9$, Distribution = Gaussian, Config A)

5.2.3 Learning rate

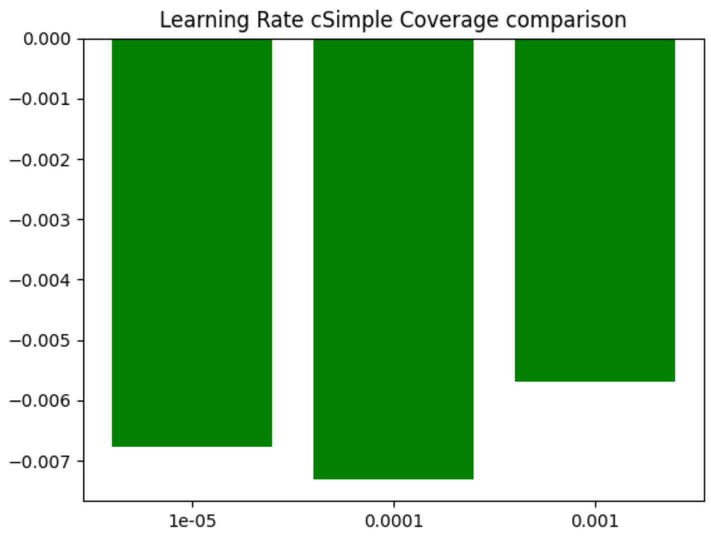


Figure 37: Comparison of different learning rate (Model: Simple (Default parameter values), Epochs = 100, $\alpha = 0.9$, Distribution = Gaussian, Config A)

The default learning rate in GluonTs for all deep learning models is 1^{-3} Learning rate as a great impact on the results.

5.2.4 Epochs

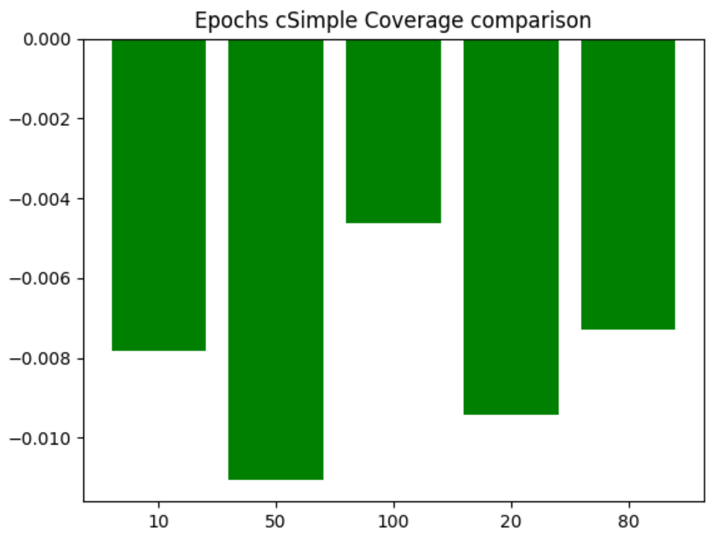


Figure 38: Comparison of different epochs number (Model: Simple (Default parameter values), $\alpha = 0.9$, Distribution = Gaussian, Config A)

All the deep learning models implemented must be trained a certain number of epochs. The default number is 100 epochs. This value is a good choice (better bandwidth of the comparison), as are values slightly inferior (which have better Coverage).

5.2.5 Output Distribution

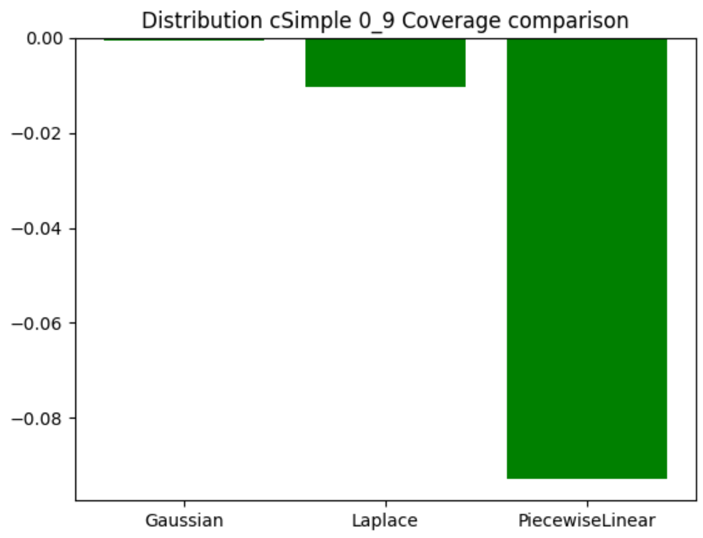


Figure 39: Comparison of different output distribution (Model: Simple (Default parameter values), Epochs = 100, α = 0.9, Config A)

5.3 Different models parameters comparison and results

The different models are described in the 2.8 section

5.3.1 Simple

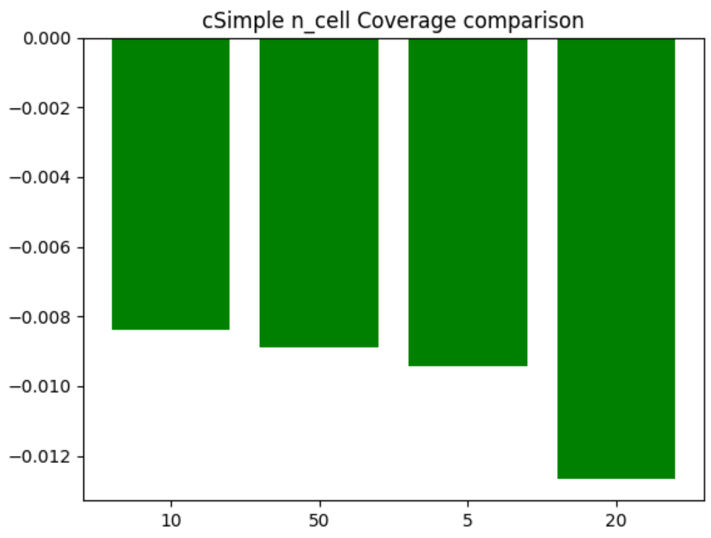


Figure 40: Comparison of different *n_cell* values for Simple model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

5.3.2 Simple Feed Fordward

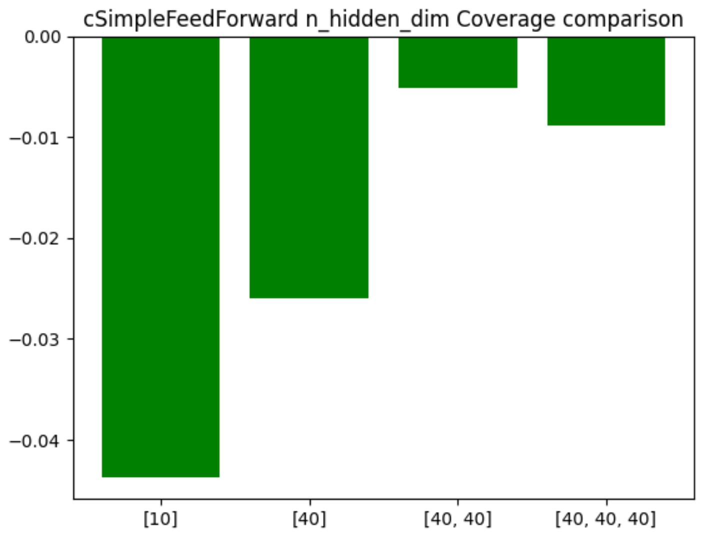


Figure 41: Comparison of different *n_hidden_dim* values for Simple FeedForward model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

5.3.3 Canonical RNN

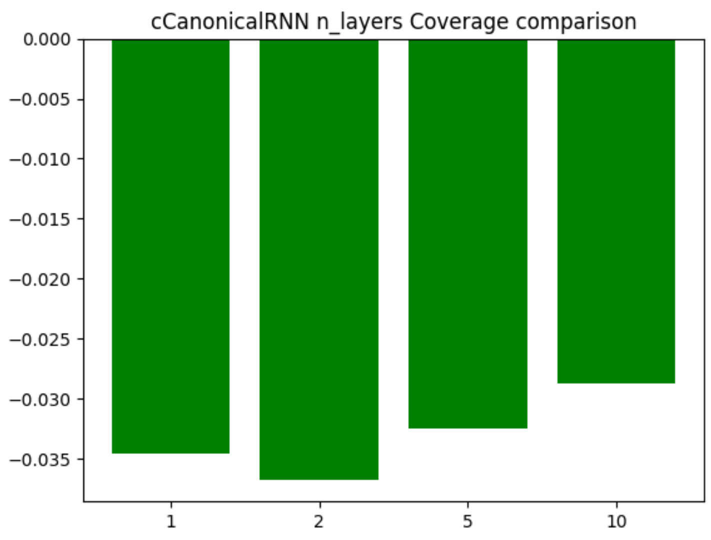


Figure 42: Comparison of different n_layers values for Canonical RNN model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

5.3.4 Deep AR

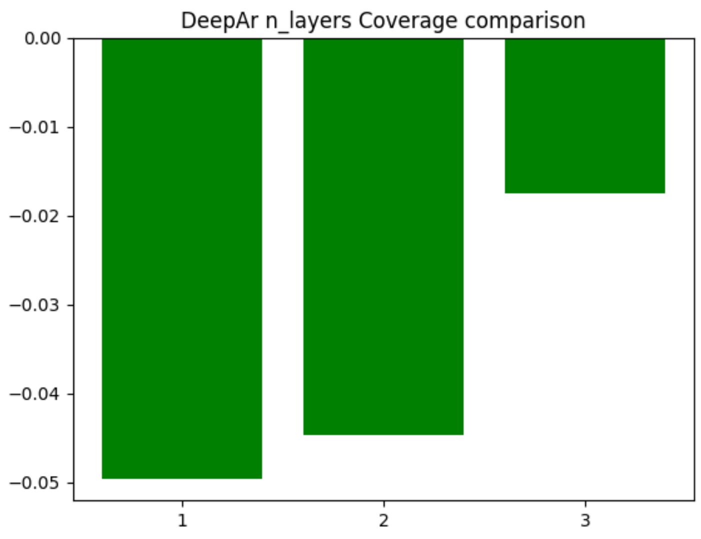


Figure 43: Comparison of different n_layers values for Deep AR model (Epochs = 100, Distribution = Gaussian, Config A)

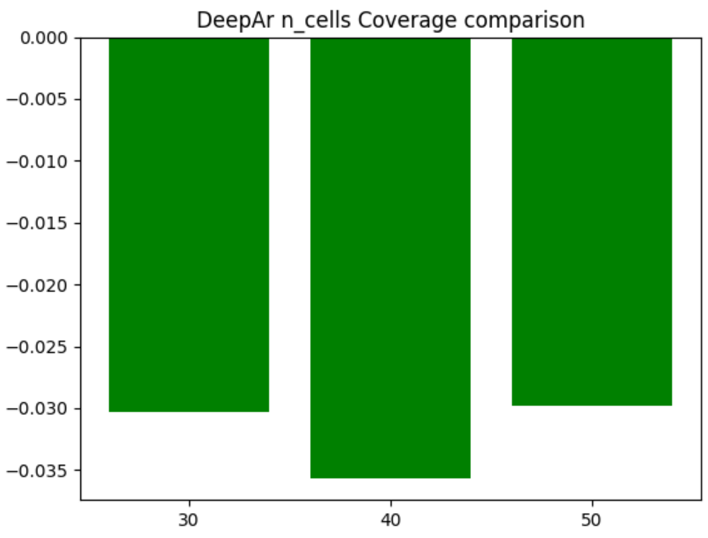


Figure 44: Comparison of different n_cells values for Deep AR model (Epochs = 100, Distribution = Gaussian, Config A)

5.3.5 Deep Factor

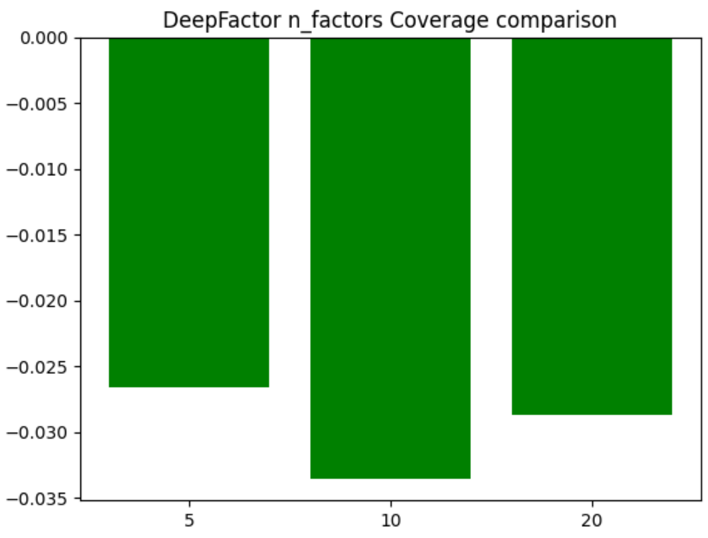


Figure 45: Comparison of different $n\_factors$ values for Deep Factors model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

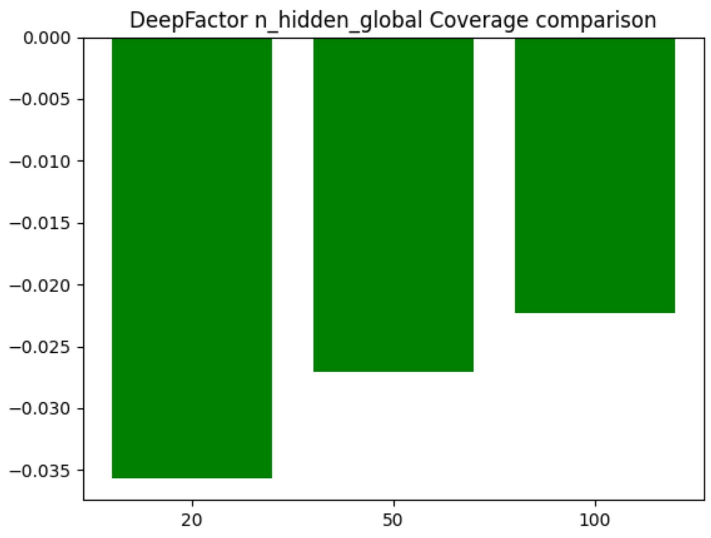


Figure 46: Comparison of different *n_hidden_global* values for Deep Factors model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

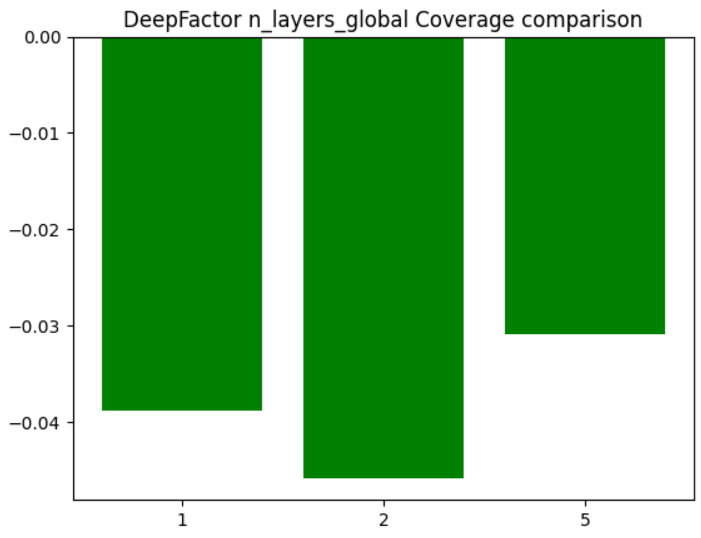


Figure 47: Comparison of different *n_layers_global* values for Deep Factors model (Epochs = 100, Distribution = Gaussian, $\alpha = 0.9$, Config A)

Contrary to another model, different hyperparameters of Deep Factors had a relatively low impact on the quality of the model.

5.3.6 MQCNN

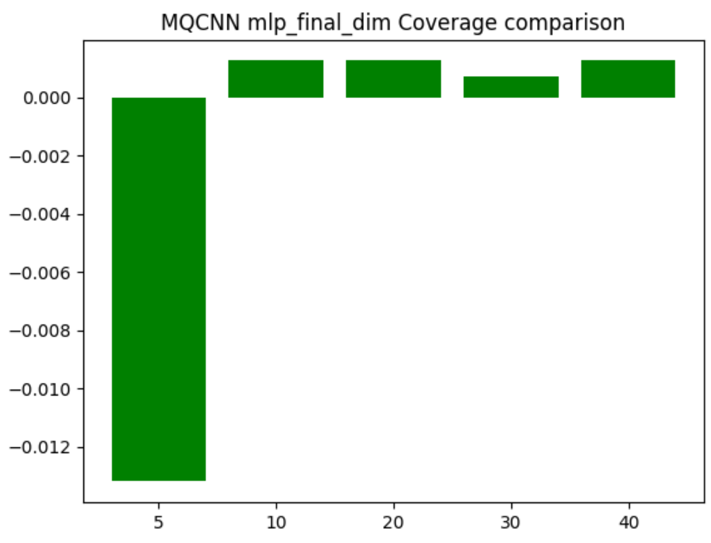


Figure 48: Comparison of different *mlp_final_dim* values for MQCNN model (Epochs = 100, Distribution = Gaussian, Config A)

5.3.7 MQRNN

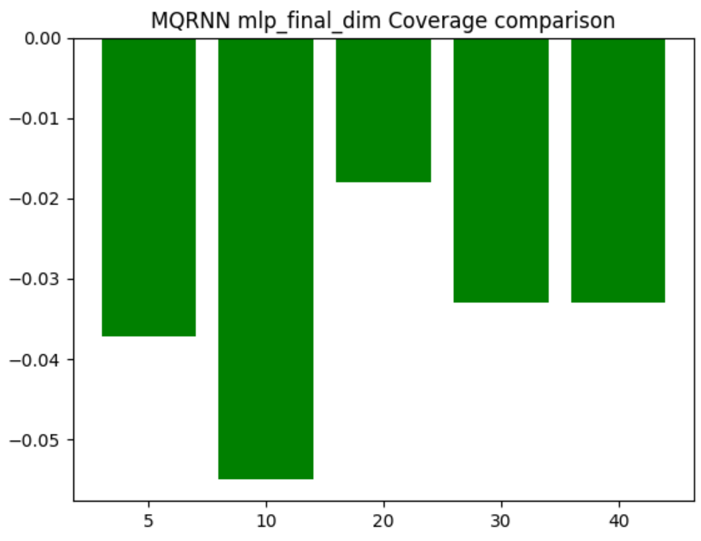


Figure 49: Comparison of different *mlp_final_dim* values for MQRNN model (Epochs = 100, Distribution = Gaussian, Config A)

5.4 Global comparison

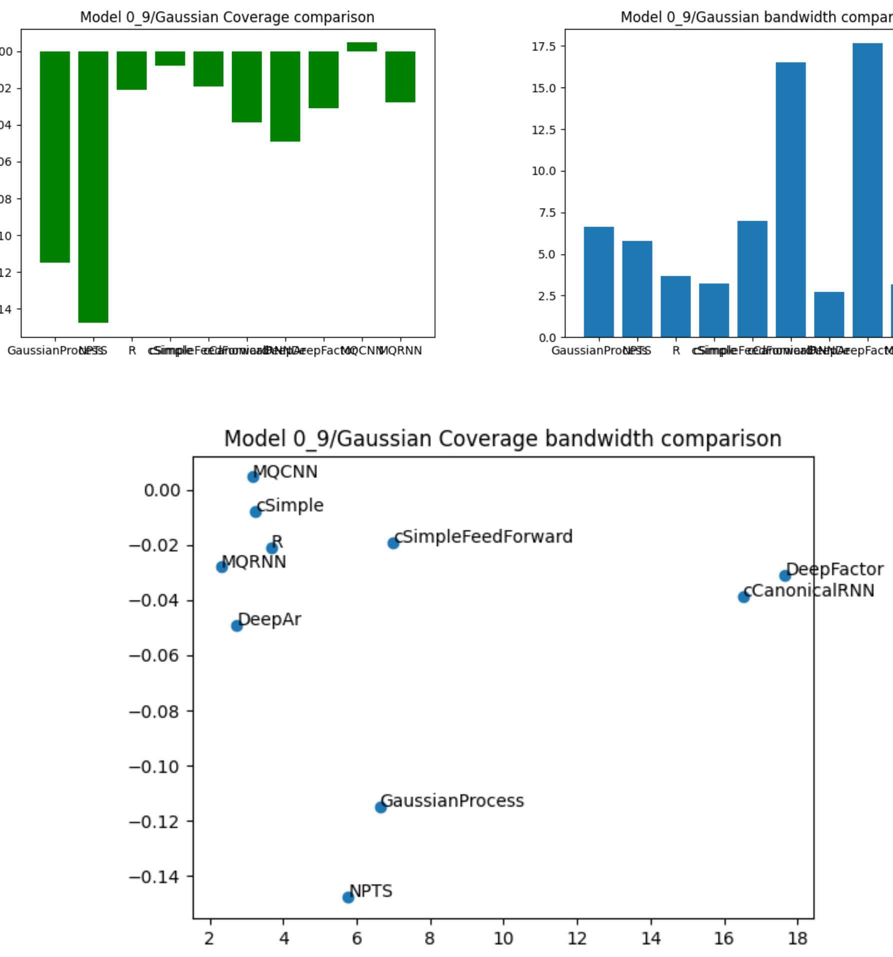


Figure 50: Comparison of different models (Epochs = 100, Distribution = Gaussian, Config A)

Now that all the hyperparameters has been studied, all the tuned models can be compared to infer a hierarchy of the model which forecast the different configuration the better. This will leave us to the question at the base of this work.

The first observation is that DeepFactor and CanonicalRNN underperform in term of bandwidth compared to the others. The second is that Gaussian Process and NPTS underperform in terms of Coverage compared to the others. It leaves us 6 models. The better model in terms of bandwidth, with an acceptable Coverage if we consider that this window of value is acceptable is MQRNN. The better model in terms of Coverage is MQCNN. It Coverage is not only higher than others, but positive. Simple model gives impressive results considering the simplicity of it implementation. The ETS model gives good results as the only not deep learning model.