Optimization of LSTM Time-series Forecasting

Safaa Alnabulsi
TU Berlin
Berlin, Germany
safaa.alnabulsi@campus.tu-berlin.de

Samira Yarollahi
TU Berlin
Berlin, Germany
samira.yarollahi@campus.tu-berlin.de

Yamen Ali
TU Berlin
Berlin, Germany
yamen.ali@campus.tu-berlin.de

***Abstract*—Time-series forecasting is an important topic with various real world applications such as stock prices prediction, weather temperatures forecasting or Hbbtv audience size prediction. In this paper, we show different approaches to use LSTM neural networks for Multi-step ahead forecasting. The suggested models will not only output points prediction but also prediction intervals (i.e. lower and upper bounds) in which the predicted points can lie with high probability. The M4-Competition [3] hourly dataset was used for training, testing and evaluation. Also, all the metrics used to evaluate and compare the obtained results are taken from M4-Competition [3] organizers suggestions.**

***Index Terms*—time-series, LSTM, Multi-step, forecasting**

I. Introduction

Time-series forecasting can be broadly divided into two types: One-step ahead and Multi-step ahead forecasting. Given a series of observations, One-step ahead forecasting tries to predict the next observation value. On the contrary, Multi-step ahead forecasting aims to predict the next (n) observations. Different strategies exist for Multi-step ahead forecasting (e.g. recursive strategy, direct strategy, multiple output strategy, etc.). In this paper we will focus only on multiple output strategy that depends on using one model to predict the entire forecast horizon in one shot.

Another aspect in time-series forecasting is the prediction intervals which gives intervals that the predicted observation lies in. These intervals represents how certain we are about our predictions, so narrower intervals mean more certainty and vice versa. Each interval will be represented by (lower bound, upper bound) and the final aim of our model is to predict the next (n) observations, a lower bound and an upper bound for each predicted observation.

The main building block of our models is the Long short-term (LSTM) neural networks which are known for yielding good results on tasks that involve temporal pattern recognition. Later on we will describe the architecture of the used models but not the details of LSTM since it is beyond the scope of this paper.

It is worth mentioning that originally our work was aiming to improve the results that Berken [1] obtained in his thesis where he tried to forecast the net reach of HbbTV. Due to GDPR restrictions we were not able to use the HbbTV data for training. Instead we used the open-source M4-Competition hourly data. We believe retraining our models on the original HbbTV data should be straight forward and would achieve better results.

II. M4-Competition Dataset

The M4-Competition [3] is the fourth version of the Makridakis competitions series which aims to improve and evaluate the forecasting accuracy on a dataset of time-series collected from different domains. All time-series of this competition are univariate which means every observation in the time-serie is represented by only one value and all values are numerical so no need for specific preprocessing or representation change. For this paper, we concentrate our work only on the hourly time-series dataset but in principle any other dataset can be chosen.

A. The hourly dataset

The hourly dataset consists of two files **(Hourly-train.csv, Hourly-test.csv)**. The **Hourly-train.csv** file contains 414 time-series varying in length from 700 to 960 time steps. The **Hourly-test.csv** file contains 414 time-series each of them has 48 time steps. Given a time-serie from the **Hourly-train.csv** file we should predict the values of its counterpart time-serie in the **Hourly-test.csv** file alongside an upper and a lower bounds for each step. Figure 1 shows an example from the hourly dataset.

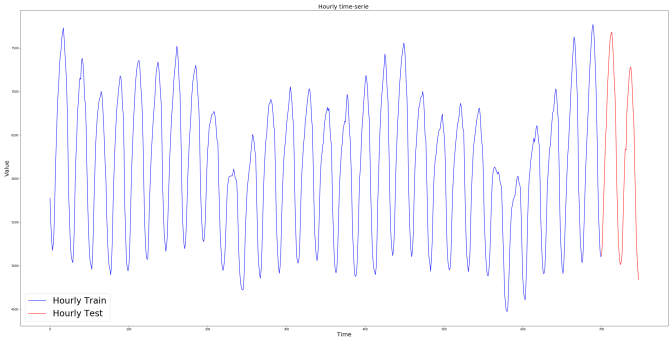

Fig. 1. A time-serie taken from the hourly dataset. The blue part is coming from the Hourly-train.csv. The red part is coming from Hourly-test.csv.

This dataset has a seasonality of 24 and no domain info was associated with it.

B. Dataset preparation

In order to get a better idea about our models generalization capabilities, we took the last 20% $\approx$ 83 time-series from the dataset as a holdout. Thus, we have three types of data:

- **Training Data:** These are the first 331 time-series from the Hourly-train.csv file and they are used to train the model.
- **Test Data:** These are the first 331 time-series counterparts from the Hourly-test.csv file and they are used to test the model ability to predict 48 values of the time-series that were used for training.
- **Holdout Data:** These are the last 83 time-series from both Hourly-train.csv and Hourly-test.csv files and they are used to test the model ability to predict 48 values of time-series that it had never seen before (i.e. never trained on).

Since we don't have enough training data we will generate multiple time-series pairs (input, target) from each time-serie. In order to do this, we will introduce two terms:

- **Lookback window:** A number that describes how many steps from a time-serie will be fed to the model. In principle, this number can be treated as an additional hyperparameter, but through all our experiments a lookback window of size 48 was selected. The intuition behind this selection is that we have a seasonality of 24 in the data, so a reasonable lookback window would be 24 or any of its multiples. Keeping in mind we have only cpus for training, we chose 48 so that the number of the generated training pairs is adequate and manageable.
- **Horizon:** A number that indicates how many steps a head we want to predict. This number is always 48 since we want to predict the time-series from the Hourly-test.csv file and they are all of length 48.

To generate time-series training pairs we take a moving window of size 96 and slide it on each time-serie with step size = 48. The first 48 values will represent the model input and the last 48 values will represent the targets.

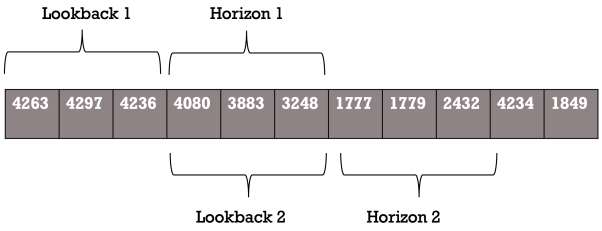

Fig. 2. An example of generating 2 pairs of (input, target) time-series from the same time-serie using the sliding window approach

Following this approach we were able to generate more than 5000 (input, target) pairs for training.

Finally, the (input, target) pairs used for evaluation on test and holdout data are constructed by taking the last 48 steps of each time-serie from **Hourly-train.csv** file as the input and its counterpart time-serie from **Hourly-test.csv** file as the target.

C. Data preprocessing

The only preprocessing needed is the standardization of the data. This is done by merging the data from the **Hourly-train.csv** file and **Hourly-test.csv** file into one matrix then standardizing it (i.e. subtract the mean and divide by standard deviation).

III. Methodology

A. Overview

We worked on three different approaches that we will refer to them as **(Berken's approach, Modified Berken's approach, KL divergence approach)**. All of these approaches depend on LSTM and have similar model architecture which consist of : Multiple LSTM layers followed by a dense layer with linear activation function.

B. Hyperparameters

Following are the hyperparameters that were used in the training :

- **Learning Rate**: The learning rate of the RMSprop optimizer used for training.
- **Batch Size**: The number of time-series fed to the model at once.
- **Dropout Ratio**: The probability of training a node in a layer. This is mainly used to prevent over fitting.
- **Clipping value**: A value to clip the gradient at, in order to avoid exploding gradient problem.

C. Berken's Approach

Here we tried to train the model as suggested by Berken [1] in his thesis. The model input is of size **lookback=48** and it represents the points of the time-serie. The model output is 48*3 values representing the predicted horizon, the lower bounds and the upper bounds. .

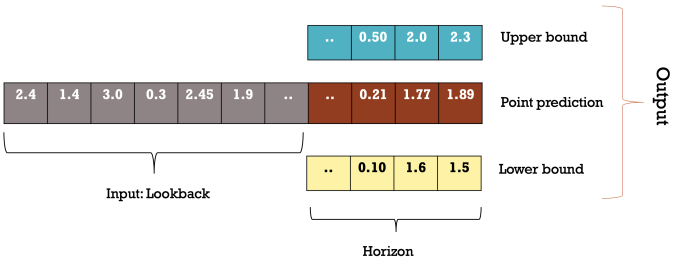

Fig. 3. An example of the model input and output in Berken's approach

The used loss function has the following formula:

$$Loss_{QD.c} = MPIW_{bal} + \lambda(\frac{n}{\alpha(1-\alpha)} max(0,(1-\alpha) - PICP)^2 + wMASE) \quad (1)$$

This function depends on three main components:

- $MPIW_{bal}$: The mean prediction interval width controls the predicted interval (i.e. the upper and lower bounds) and tries to make it as narrow as possible.
- $PICP$: The prediction interval coverage probability assures that the future points (i.e. the targets) lie between

the predicted lower bounds and the predicted upper bounds.

- $wMASE$: The weighted mean absolute scaled error represents the error in the points prediction.

The $Loss_{QD.c}$ aims to minimize $MPIW$ and $wMASE$ and maximize PICP. In other words, it aims to predict time-serie points as close as possible to the targets and a prediction interval that is as narrow as possible and contains the future points.

For more details about this function and the full formulas refer to Berken's thesis [1].

D. Modified Berken's Approach

In the original Berken's approach [1] $wMASE$ is used in order to have better accuracy on the predicted values close to the present. So different weights will be assigned to the errors in a decreasing manner. For example, if the errors associated with the points prediction are $[3, 5, 7, 6, 4]$ then these values will be weighted with $[1, 0.9, 0.84, 0.76, 0.5]$. Such weighting for the errors forces the model to focus more on predicting the points in the near future more accurately than the ones in the far future.

As will be shown later on, we were not able to get good results using the original $Loss_{QD.c}$ function and we believe this is because $wMASE$ is hard to optimize. This is why we worked on a modified version of $Loss_{QD.c}$ where we used $MASE$ instead of $wMASE$. Thus in our approach we have no weighting for the errors and all predicted points are equally important regardless how far in the future they are. This modification makes the loss function $Loss_{QD.c}$ easier to minimize and the model indeed gives better results.

In this approach all other aspects regarding the model input and output are identical to the original Berken's approach [1].

E. KL Divergence approach

A Gaussian distribution is a type of probability distribution that can be defined by **(mean, standard deviation**). If we consider that each point in the time-serie represents the mean of a Gaussian distribution and that we can obtain its upper and lower bounds like the following:

$$UpperBound = Mean + Standard\ Deviation$$

$$LowerBound = Mean - Standard\ Deviation$$

then every point in the time-serie can be represented by the duality **(mean, standard deviation**). Hence, we can reformulate our task to be minimizing the distance between a vector of target Gaussian distributions and a vector of predicted Gaussian distributions.

In this approach, the model input is of size **lookback*2** because each point in the lookback time-serie is represented by 2 values (mean = point, std) later on we discuss how to generate a proper std for each point in the time-serie. Similarly the model output is of size **horizon*2** because each point in the horizon is represented by 2 values (mean = point, std). Figure 4 shows and example of the model input and output.

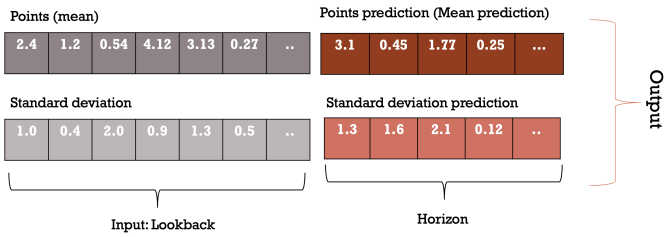

Fig. 4. An example of the model input and output in the KL Divergence approach

In order to minimize the distance between the target Gaussian distributions (i.e. the target points and bounds) and the predicted Gaussian distributions (i.e. the predicted points and bounds) we depend on KL divergence function.

In general, the Kullback–Leibler divergence (KL divergence) is a measure of how one probability distribution is different from a second, reference probability distribution. The KL divergence between two Gaussian distributions can be given in the following formula:

$$KL(p, t) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\mu_2^2} - \frac{1}{2}$$

where (σ_1, μ_1) are the mean and standard deviation of the first Gaussian (i.e. the one that represents the predicted point and its bounds) and (σ_2, μ_2) are the mean and standard deviation of the second Gaussian (i.e. the one that represents the target point and its bounds).

The final loss function that is used to train the model is simply given by :

$$Loss_{kl} = \frac{1}{horizon} \sum_{i=1}^{horizon} KL(p_i, t_i)$$

As seen from the previous discussion, we need to generate a value for each point in the time-serie that would represent the standard deviation of the associated Gaussian distribution. To do this we came up with the following simple algorithm :

1) Take the absolute values of the time-serie points.
2) Divide the values into ranges.
3) Assign each range a weight such that the range that contains the smallest value gets the biggest weight and the one that contains the biggest value gets the smallest weight.
4) The standard deviation associated with a time-serie point is obtained by multiplying the absolute value of the point by the weight assigned to its range.

The intuition behind this algorithm is that we want to have different and reasonable standard deviation values that are not too big nor too small. For example, if $point = 0.1$ then a good choice would be $std = 0.1 * 3 = 0.3$. But if $point = 4$ then a good choice is $std = 4 * 0.3 = 1$. Figure 5 shows a time-serie and its upper and lower bounds that we obtain by adding and subtracting the associated standard deviation from each point.

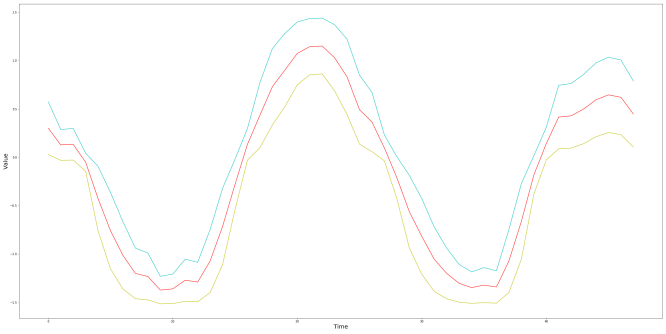

Fig. 5. An example of the time-serie and its upper and lower bounds we obtain from generating the standard deviation in KL divergence approach

In the previous standard deviation generating algorithm we use three parameters to control the generated values :

- The maximum weight that can be assigned to a range.
- The minimum weight that can be assigned to a range.
- The step size we make to get from the minimum weight to the maximum weight. Hence, the number of ranges is $ranges = \frac{max\ weight - min\ weight}{step\ size}$

Choosing appropriate values for these parameters will guarantee that the bounds we want to predict are neither too wide which will result in bad evaluation, nor too narrow which will make the problem harder for the model.

IV. Empirical Results

Here we show the final results we were able to achieve in the previously mentioned approaches. The training was carried out on our personal Laptops so only cpus were used. To keep it concise, only the results of the top three models will be shown for each approach.

A. Evaluation Metrics

In order to compare our results with the ones from M4-Competition and the other teams, we used four metrics that were suggested by the M4-Competition organizers.

MASE [2] and sMAPE [2] are used to measure the error in the points forecast :

$$sMAPE = \frac{2}{h}\sum_{t=1}^{h}\frac{\left|Y_t - \hat{Y}_t\right|}{|Y_t| + |\hat{Y}_t|} * 100(\%)$$

$$MASE = \frac{1}{h}\frac{\sum_{t=1}^{h}\left|Y_t - \hat{Y}_t\right|}{\frac{1}{l}\sum_{t=2}^{l}|X_t - X_{t-1}|}$$

where :

- h : The horizon (i.e the predicted time-serie length).
- l : The lookback (i.e the input time-serie length).
- Y: The target time-serie.
- $\hat{Y}$: The predicted time-serie.
- X: The in-sample time-serie (i.e. the one used as model input).

As we can notice, $sMAPE$ uses percentage errors that are scale independent which makes it easy to understand and interpret. On the other hand, $MASE$ compares our predictions to the ones made by one-step Naïve S forecast.

To measure the errors in bounds forecast, we use MSIS [2] and ACD [2] :

$$ACD = |Average\ Coverage - 0.95|$$

where $AverageCoverage$ is simply how many target points lie between the predicted upper and lower bounds on average.

$$MSIS = \frac{1}{h}\frac{\sum_{1}^{h}(U_t - L_t) + \frac{2}{a}(L_t - Y_t)1_{Y_t<L_t} + \frac{2}{a}(Y_t - U_t)1_{Y_t>U_t}}{\frac{1}{l}\sum_{t=2}^{l}|X_t - X_{t-1}|}$$

where :

- h : The horizon (i.e the predicted time-serie length).
- l : The lookback (i.e the input time-serie length).
- Y: The target time-serie.
- L: The predicted lower bounds.
- U: The predicted upper bounds.
- a: The significance level (= 0.05).
- $1_{Y_t<L_t}, 1_{Y_t>U_t}$: The indicator function.

Hence, to get low MSIS error, the upper and lower bounds should be as narrow as possible and the future values should lie within the corresponding bounds.

B. Berken's Approach Results

Table I, shows the top three models architecture and the used hyper parameters in the training.

TABLE I
Architecture & Hyper Parameters

ID	LSTM Layers	Layer Size	Learning Rate	Batch Size	Drop out Ratio	Clipping Value
1	2	120	0.002	120	0.2	0.2
2	3	120	0.002	120	0.1	0.2
3	6	120	0.002	120	0.2	0.2

Tables II & III, shows the MASE, sMAPE, ACD and MSIS errors of the previous models when evaluated on test data. It shows also the improvement achieved over the errors of a Naive model.

TABLE II
Points Prediction Errors on Test Data

ID	MASE	Improvement over Naive	sMAPE	Improvement over Naive
1	25.56	-84.50 %	49.40	-42.71 %
2	29.49	-112.87 %	56.83	-64.16%
3	33.06	-138.63 %	63.46	-83.33 %

Figure 6 shows the best predictions we were able to get from the best model. It is apparent that the learning algorithm is not converging and the predictions are worse than a Naive model. As mentioned before, this is mainly because of the $wMASE$ in the used $Loss_{QD.c}$ function.

TABLE III
BOUNDS PREDICTION ERRORS ON TEST DATA

ID	ACD	Improvement over Naive	MSIS	Improvement over Naive
1	0.05	-158.22 %	203.23	-146.03%
2	0.07	-213.77 %	204.48	-147.54%
3	0.1	-334.72 %	281.62	-240.93%

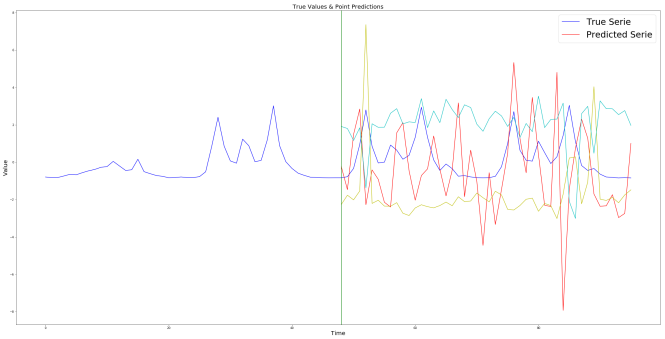

Fig. 6. The best predictions we were able to get with Berken's approach.

C. Modified Berken's Approach Results

Tables IV, V & VI show the top three models architecture, points prediction errors and bounds prediction errors respectively. As before the errors are reported on test data.

TABLE IV
ARCHITECTURE & HYPER PARAMETERS

ID	LSTM Layers	Layer Size	Learning Rate	Batch Size	Drop out Ratio	Clipping Value
1	6	70	0.003	120	0.2	0.2
2	6	120	0.002	120	0.2	0.2
3	4	70	0.003	120	0.2	0.2

TABLE V
POINTS PREDICTION ERRORS ON TEST DATA

ID	MASE	Improvement over Naive	sMAPE	Improvement over Naive
1	8.12	41.36 %	24.83	28.26 %
2	10.45	24.56 %	27.75	19.82 %
3	11.03	20.39 %	27.94	19.27 %

It is apparent that this approach achieved better results in general on both points and bounds predictions. Although, we were able to get a good improvement over the naive points prediction. Nevertheless, the bounds predictions still worse the naive ones.

We noticed that bigger models achieve better results when it comes to bounds prediction (e.g. In the previous tables, Model 2 achieved better bounds prediction than Model 1 but worse points prediction). Thus, training bigger models with hyper parameters tuning might enhance these results further, but since we don't have access to computational power we were not able to try that.

TABLE VI
BOUNDS PREDICTION ERRORS ON TEST DATA

ID	ACD	Improvement over Naive	MSIS	Improvement over Naive
1	0.03	-63.75 %	107.40	-30.02 %
2	0.02	-12.85 %	92.19	-11.61 %
3	0.03	-29.82 %	107.57	-30.22 %

Figure 7 shows the best predictions we were able to get from the best model. The model was able to predict the main pattern and seasonality in the time-serie, but the bounds were random and too wide.

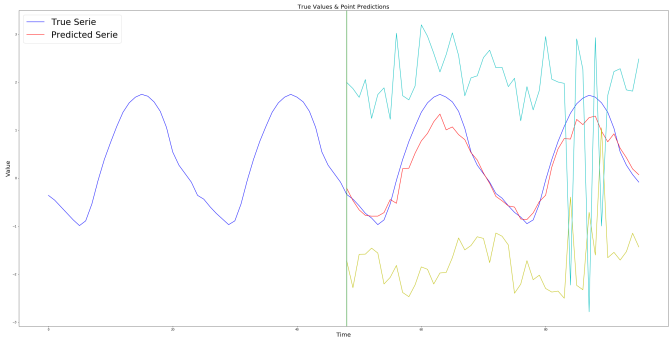

Fig. 7. The best predictions we were able to get with Modified Berken's approach.

D. KL Divergence Approach Results

Tables VII, VIII & IX show the top three models architecture, points prediction errors and bounds prediction errors respectively. As before the errors are reported on test data.

TABLE VII
ARCHITECTURE & HYPER PARAMETERS

ID	LSTM Layers	Layer Size	Learning Rate	Batch Size	Drop out Ratio	Clipping Value
1	6	10	0.001	128	0.1	2
2	2	20	0.001	128	0.2	2
3	2	30	0.001	128	0.2	0.08

TABLE VIII
POINTS PREDICTION ERRORS ON TEST DATA

ID	MASE	Improvement over Naive	sMAPE	Improvement over Naive
1	1.80	87 %	13.76	60.25 %
2	2.08	84.93 %	13.10	62.15 %
3	2.12	84.64 %	13.63	60.60 %

As shown in the previous tables, this approach achieved the best results and we were able to get improvements in both points and bounds predictions over the naive model. It is also worth mentioning that the trained models were relatively small when compared to the ones used in Berken's approach [1].

TABLE IX
BOUNDS PREDICTION ERRORS ON TEST DATA

ID	ACD	Improvement over Naive	MSIS	Improvement over Naive
1	0.02	-6.28 %	16.66	79.82 %
2	0.03	-37.48 %	21.88	73.51 %
3	0.01	16.15 %	19.11	76.85 %

Figure 8 shows the best predictions we were able to get from the best model. The model was able to predict the forecast horizon almost perfectly and the bounds are narrow and containing the time-serie entirely.

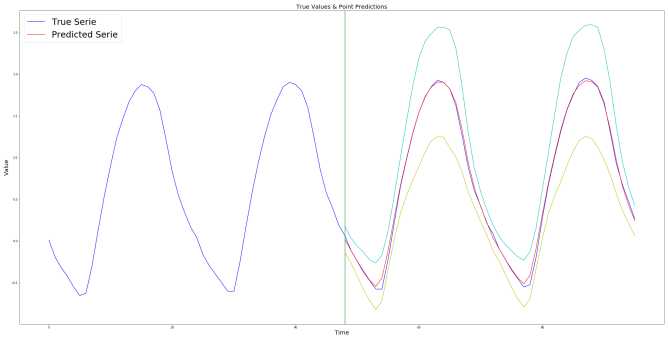

Fig. 8. The best predictions we were able to get with KL divergence approach.

Figure 9 shows the worst predictions we get from the best model. It is obvious that the time-serie horizon is hard to predict even for humans especially it has no apparent seasonality or trend.

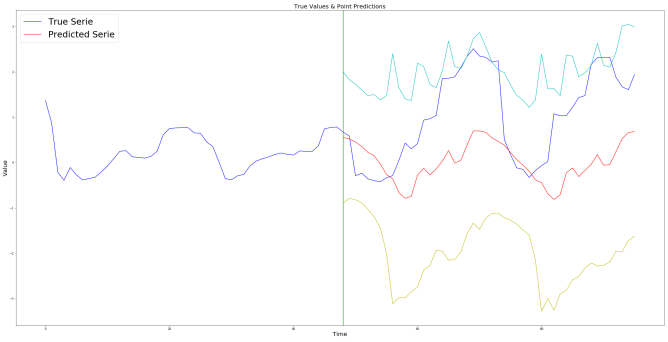

Fig. 9. The worst prediction from KL divergence approach.

E. Comparison

Here we compare the best models from Berken's approach, Modified Berken's approach and KL divergence approach based on their errors on the holdout data. Tables X & XI show the points prediction errors and bounds prediction errors on the holdout data.

The KL divergence approach was able to achieve the best results although the model was never trained on any past values for the holdout data time-series. It is also apparent that all of the approaches achieved lower errors on the holdout data

TABLE X
POINTS PREDICTION ERRORS ON HOLDOUT DATA

Approach	MASE	Improvement over Naive	sMAPE	Improvement over Naive
Berken	4.549	-72.42 %	105.07	-37.45 %
Modified Berken	1.54	41.59 %	43.61	42.94 %
KL Divergence	1.18	54.95 %	31.96	58.18 %

TABLE XI
BOUNDS PREDICTION ERRORS ON HOLDOUT DATA

Approach	ACD	Improvement over Naive	MSIS	Improvement over Naive
Berken	0.07	-120.58 %	32.62	-25.71 %
Modified Berken	0.03	-13.83 %	18.60	28.30 %
KL Divergence	0.02	41.08 %	10.16	60.82 %

compared to the errors on the test data. This can be attributed to the fact that the test data probably contains more challenging time-series than the holdout data. Nevertheless, these errors give us a good idea about the generalization power of the models.

Figures 10, 11 & 12 show the best predictions we were able to get from the three approaches on the holdout data.

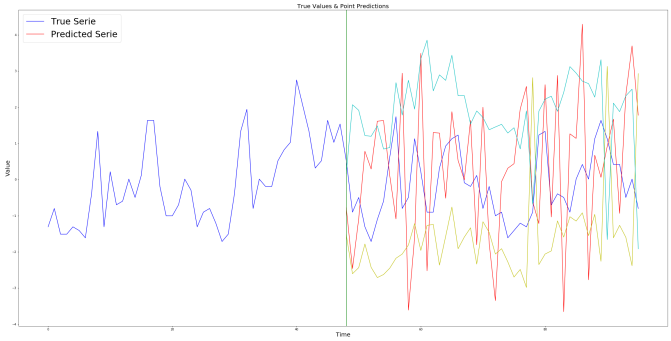

Fig. 10. The best prediction from Berken's approach on the holdout data.

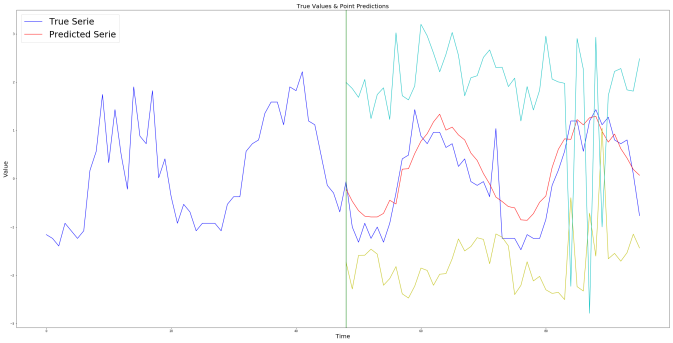

Fig. 11. The best prediction from modified Berken's approach on the holdout data.

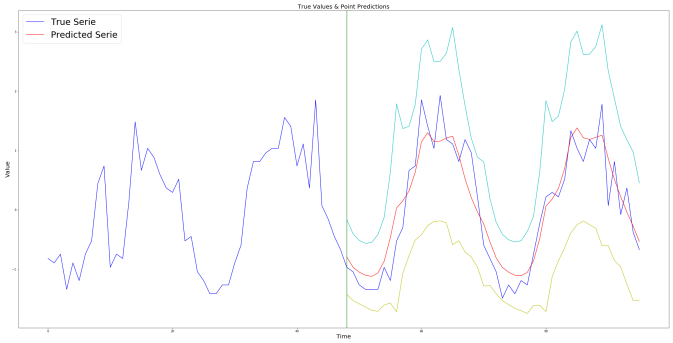

Fig. 12. The best prediction from KL divergence approach on the holdout data.

V. Discussion

From the previously showed results, we can conclude that the original $Loss_{QD.c}$ function suggested by Berken [1] is complex and hard to optimize which results in bad predictions for the time-series points and bounds. This complexity is due to the usage of $wMASE$ which gives different weights to the points prediction errors.

Substituting $wMASE$ with $MASE$ simplifies the $Loss_{QD.c}$ function and enables the model of predicting the horizon time-series more accurately. Nevertheless, models trained with modified $Loss_{QD.c}$ function were not able to give good lower and upper bounds. We think that using bigger and deeper models for this approach might achieve better results on both points and bounds predictions, but we were not able to verify our hypothesis because we didn't have access to a proper computational powers (i.e. GPUs).

The best results were obtained by KL Divergence approach, mainly due to the simplicity of the used loss function. We can summarize the pros of the approach by the following points:

- Simple loss function.
- Relatively small models were able to achieve good results.
- In principle, it is possible to add any prior domain knowledge we have into the learning process. For example, if the time-series data represent the weather temperature in summer, then it might be more suitable to use a heavy tail Gaussian distribution instead of the normal one.

On the other hand, the main drawback of this approach is that both lower and upper bounds depends mainly on one learned parameter which is the standard deviation.

Finally, it is worth mentioning that having more training data might enhance the results of the discussed approaches. Decreasing the lookback window size will generate more data, but as mentioned before, the lack of proper computational power prevented us from successfully trying with smaller lookback window sizes.

Acknowledgment

We thank Dr. Christopher Krauß for his mentoring and suggestions.

References

[1] C. Krauß, I. Schieferdecker, M. Hauswirth and B. Bayat "A concept and design for short-term HbbTV net reach forecasting," Master Thesis, TU Berlin.

[2] S. Makridakis, E. Spiliotis, and V. Assimakopoulos "The M4competition: 100, 000 time series and 61 forecasting methods," International Journal of Forecasting, Volume 36, Issue 1, January–March 2020, Pages 54-74.

[3] M4 Competition dataset and results [Online] https://github.com/M4Competition/M4-methods.