IMPROVING THE PERFORMANCE OF TIME-SERIES DATA ANALYSIS

Domain: Deep Learning

Deep Learning which is part of machine learning achieves huge flexibility by learning to represent the world with multiple processing layers to learn data representation with multiple levels of abstraction. These methods have improved drastically many domains such as visual object recognition, drug discovery and speech recognition. Deep learning determines a complex structure in huge datasets to demonstrate how machine learning should change its internal parameters by using backpropagation algorithm which are used to compute each layer's representation from the previous layer representation. In processing images, speech, audio and video – deep convolutional networks have brought development, where in recurrent neural networks (RNN) are used in processing sequential data like speech and text.

We humans never start our thinking process from scratch every second. Our thoughts have persistence. Let us take an instance- when we are reading a description about something, we tend to understand every word based on our understanding of the previous words. We don't leave everything away and begin thinking again from the scratch. This seems to be a major shortcoming with the traditional neural networks as it is not clear as how they use its previous event's analysis to notify the later ones.

RNN:

Recurrent neural networks address this issue. They are networks which maintain in their hidden units, a state vector which contains the past elements history of the sequence, allowing information to persist. They have loops in them as shown in the below diagram.

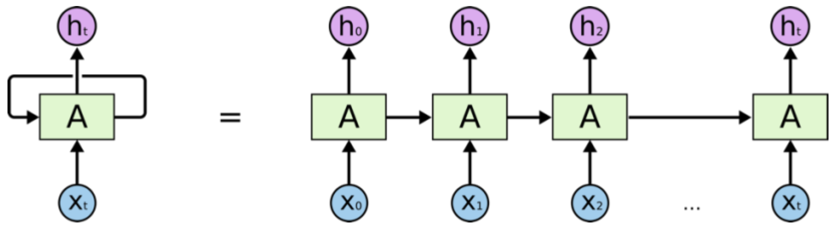


Fig. 1. A loop unrolled RNN

A part of network, A, takes some input x_t and outputs h_t and the loop will pass the information from one step to other in the network. These loops look mysterious but they all are not so much different than the traditional neural network. A RNN can be assumed as multiple copies of same network where each message is passed to the successor. If we unroll the loop, this is how it looks as shown in the Fig 1.

It is observed that recurrent neural networks are related to lists and sequences due to their chain-like behavior. So, they are the natural architecture to use for tasks that involve inputs which are sequential like language and speech. And they are used well! They are pretty amazing as there have been great success employing recurrent neural networks to various problems. Crucial to these successes is making use of LSTM's which is a special kind of RNN which works much better than the standard version for many jobs. Most of the interesting results are achieved on RNN with the use of them.

BI-DIRECTIONAL RNN:

RNN is unidirectional which means that it takes as input only one sequence. As a result, it cannot take full advantage of the input data as it only learns information from the past. To overcome this problem, bi-directional RNN concept is introduced where the recurrent neural network learns from both past and future. It is in fact two unidirectional RNNs combined together where one of them learns from the future and the other learns from the past and the results of both are combined(merged) to calculate the final result.

One of the important characteristics of RNN's is they connect previous information to the present task. It would be extremely useful if they could do it, but can they do? It depends on the gap between the required information and the place where it is needed. If the gap is small, recurrent neural networks can learn to use the past information. But in certain cases where the gap is very large, RNN cannot learn to connect the information. So unluckily RNNs are not successful as the gap becomes large. Such long-term dependencies are capable of RNNs in theory but in practice, they cannot learn them. Fortunately, LSTM networks don't face this problem.

LSTM:

LSTMs are Long Short-Term Memory networks which are capable of learning long term dependencies. They work well on various problems and are widely used now. They are specifically designed to avoid this problem of long-term dependencies as their default behavior is remembering information for very long periods of time. Every RNN have the repeating modules of the network in the chain like form. This repeating module will have a structure which is very simple in recurrent neural network such as just a single tanh layer.

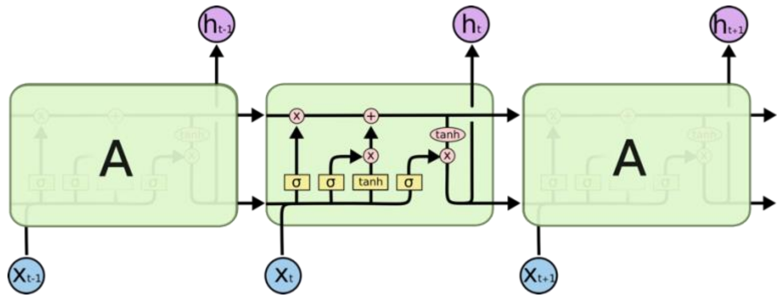


Fig. 3. Four layers in the repeating module of LSTM

In the similar way, LSTMs also have a structure of repeating modules in the chain like form, but the structure of the repeating module is different as it has four neural network layers instead of one which are interacting in a unique way.

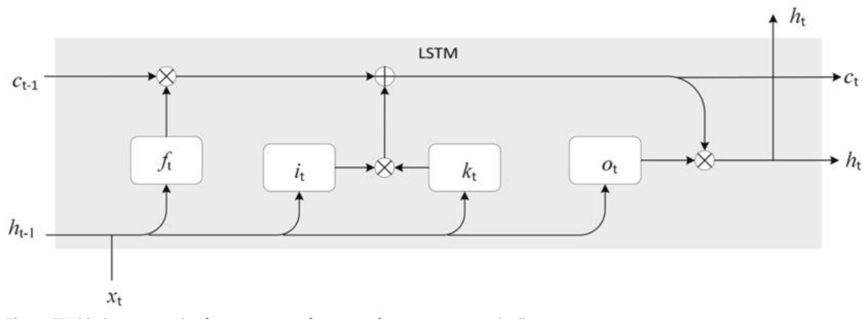


Fig. 4. A block in LSTM at step t

As shown in the above figure, the h_t which is the activation at step t is calculated using four gates- the input gate i_t , o_t which is the output gate, forget gate f_t , and c_t which is the cell gate. The input at the input gate at step t will be $\mathbf{i_t = \sigma(U_i . h_{t-1} + W_i . x_t + b_i)}$

where b_i is bias, U_i and W_i are the weight matrices, x_t is the input vector, σ(.) is the sigmoid function. The output gate and the forget gate inputs are calculated as

$\mathbf{o_t = \sigma(U_o . h_{t-1} + W_o . x_t + b_o)}$

$\mathbf{f_t = \sigma(U_f . h_{t-1} + W_f . x_t + b_f)}$

where b_o and b_f are biases and W_o, W_f , U_o and U_f are the weight matrices.

The input from the cell gate is calculated as

$c_t = f_t \cdot c_{t-1} + i_t \cdot k_t$ with $k_t = \tanh(U_k \cdot h_{t-1} + W_k \cdot x_t + b_k)$,

where b_k is the bias, tanh is the hyperbolic function, W_k and U_k are weights. **$h_t = o_t \cdot \tanh(c_t)$.** Naturally, the cell state is the main component of LSTM, which is present throughout the network. Given the forget gate f_t and input h_{t-1} and x_t, it determines what information has to be deleted from c_{t-1}, the previous cell state. For each value in c_{t-1}, the forget gate generates a value between 0 and 1 by taking x_t and h_{t-1} as input. Then the input gate i_t decides in two steps what information in the present state c_t is to be stored. The two steps would be: based on the present input, k_t calculates a set of candidate values. Then the input gate i_t decides which candidate values c_t will store using $\sigma(.)$ function. Then the c_t is calculated by the cell gate. based on c_t and o_t, finally h_t is calculated where o_t is information from output gate.

HYPERPARAMETERS:

They are the parameters available for the user to set before actually starting the training of the learning model. They include various parameters like:

- Initialization of weights, weight decay
- Activation functions and momentum
- Topology of the networks
- Number of epochs
- When to stop the training
- K- value in KNN etc.,

All these parameters can be potential factors in training the data so, tuning them will certainly lead to better performing learning models.

MODEL-BASED OPTIMIZATION:

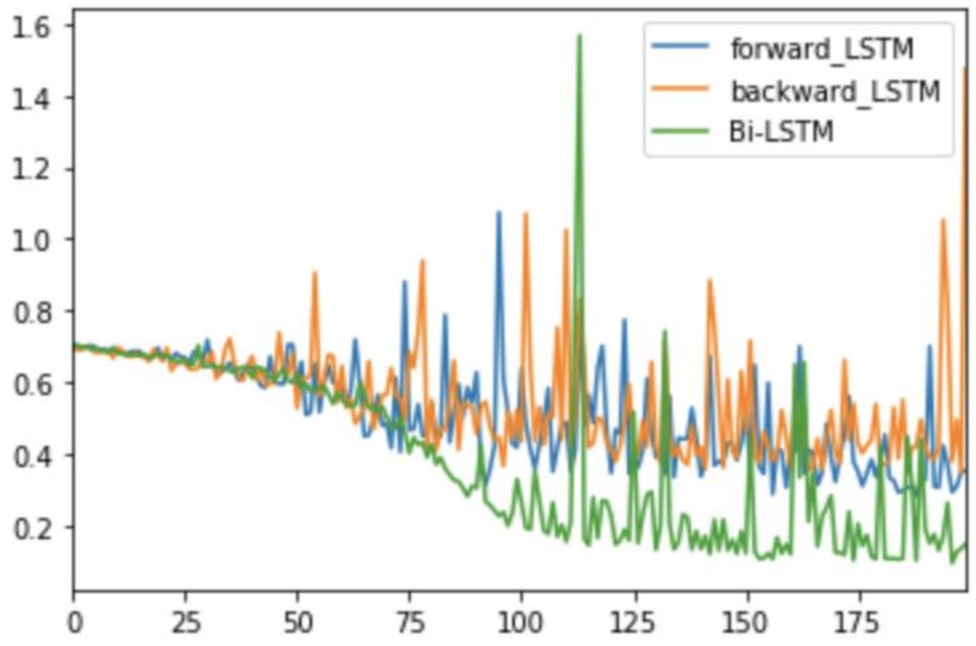
It is in contradiction to model-free optimization where we use the methods like grid search and random search where we take fixed number of parameters to tune which are independent of the past experiences but model-based optimization uses techniques like Bayesian optimization where a defined structure is used to optimize the hyperparameters. Here the past experiences are taken into consideration to predict the future hyperparameters thus making a specific set of hyperparameters have significant importance. Much detailed view of how these can be used is explained in the empirical analysis part.

EMPERICAL ANALYSIS:

In this section, we will mainly look into the possible case to optimize the time series analysis for better prediction results. We know that the main idea behind using Bi-directional Recurrent Neural Networks (BRNNs) is straight where we find the recursive loops to be in the hidden layers which learn both sides of the input sequence i.e., two leaning models are combined here one from the past sequence and one from the reverse side which means the future sequence creating an Bi-RNN but the RNN still suffers from vanishing gradient problem leading to short term memory. So, in this case we induce the concept of Long Short-Term memory (LSTM) into the Bi-RNN creating a Bi-LSTM. LSTM creates significant delays in between events leading to the elements present in memory for a longer time. Thus, stacks of LSTM are used for longer memory. This proves to have greater effect on correct predictions of the model thereby increasing the accuracy of the learning model.

Bi-LSTM Implementation:

The graph below shows the performance of Bi-LSTM on a sequence of random 0's and 1's and it is trained till 200 epochs only for the clear view and validating the performance of Bi-LSTM.
For this we used keras library's mainly and other python libraries (NumPy, pandas, matplotlib).



As we can see from the above graph that the rate of loss for Bi-LSTM is significant as the number of epochs increases which implies that there is a noteworthy increase performance of Bi-LSTM.

Why Bi-LSTM + SMBO over other methods?

Bi-LSTM is better than that of some of the really good time series forecasting statistical models like ARIMA, GARCH according to the research paper [1] where it's been proven that the Bi-LSTM works better. Now we are trying to make our time series analysis model much better with the introduction of concept of Sequential Model-Based Optimization (SMBO) which is a model-based optimization for the tuning of hyperparameters.

Though there are many methods for hyperparameter optimization like introduction of reinforcement learning where the rewards are given for right step taken towards improvement and punished for every wrong step taken and there's much more into it;

Grid search where a model is built on all the possible combinations of hyperparameters, it is computationally intensive as the number of hyperparameters increases thus suffering from curse of dimensionality. Grid search works well as long as the number of parameters range within 4.

Random Search works really well and shows significant increase in the accuracy of the learning model when compared to grid searching but the chosen parameters are very random which may lead to a very good model or maybe not. But it is still a method where future hyperparameters are not used to evaluate based on the past results.

Grid and Random search methods are not affected by past evaluations and are completely even throughout, they often work on optimizing the bad hyperparameters for a long time resulting in the wasting lot of time and computational power.

Consider the below graph, now consider applying both grid search and random search on it. As we know that most of the points are accumulated in a small range and the rest of the estimators are widely spread, even then the mentioned search techniques work on the randomly spread elements clearly wasting time as they are of probably zero significance.

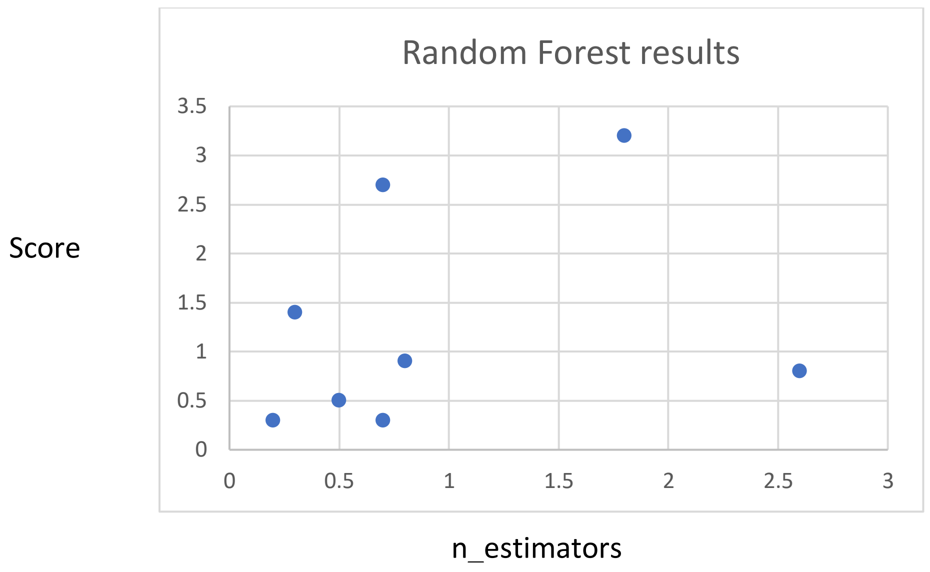


Fig 5. Just an example plot

Now comes the need for a model which is better than that of model-free approaches which are called model-based approaches like Bayesian optimization. So, in this paper we would like to integrate such significant optimizing solution to Bi-LSTM which would augment the accuracy of the time series analysis completely to a different level.

It works like this: **P(score/hyperparameters),**

$$x^\star = \arg\min_{x \in \mathcal{X}} f(x)$$

```
SMBO($f, M_0, T, S$)
1   $\mathcal{H} \leftarrow \emptyset$,
2   For $t \leftarrow 1$ to $T$,
3       $x^* \leftarrow \text{argmin}_x \; S(x, M_{t-1})$,
4       Evaluate $f(x^*)$,        $\triangleright$ Expensive step
5       $\mathcal{H} \leftarrow \mathcal{H} \cup (x^*, f(x^*))$,
6       Fit a new model $M_t$ to $\mathcal{H}$.
7   return $\mathcal{H}$
```

Algorithm for SMBO paper [3] which is called as the surrogate for the objective function. The main aim of the surrogate function is to be more accurate with more data.

In the SMBO, we use the Tree Parzen Estimator (TPE) because according to paper [2] this method showed a great impact on the hyperparameter optimization.

To tell in one sentence Bayesian optimization is used to create a probabilistic model for and objective function and use it to find out the most effective hyperparameters and use them again in the evaluation of the true objective function.

<u>HOW BAYESIAN-OPTIMIZATION REDUCE COMPLEXITY IN COMPUTATION:</u>

It might seem as if it takes a lot of computation in the calculation of hyperparameters as we have to train the model first using Bi-LSTM which helps finding the effective hyperparameters(say R1) and again train the model with the effective parameters R1 to get a completely trained and accurate model. But here once we calculate the model, from the next time we can just work only with the effective hyperparameters(R1) instead of all the parameters (say R2) where **R1 << R2**. Thereby, reducing the complexity of the calculation. Otherwise, we always need to work on the bad hyperparameters as well. Thus, we need to spend some extra time at the beginning of training the NN for getting info about the next hyperparameters leading to making lot less calls to objective function.

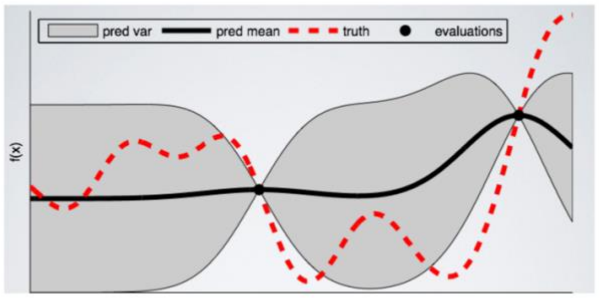


Fig 6

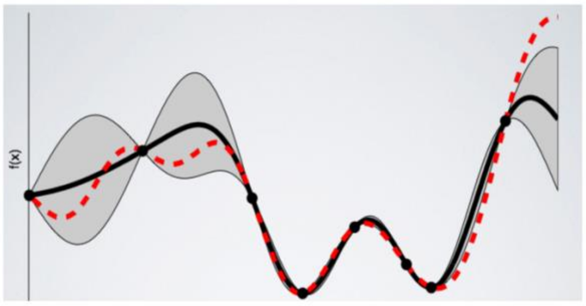
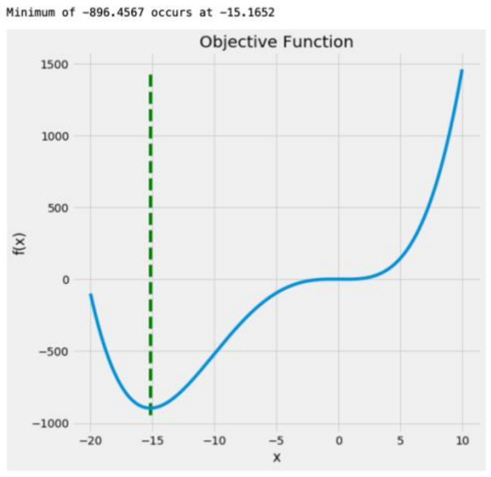


Fig 7

Consider the above images, where our SMBO works poorly in Fig 6 as you can see a lot of variance between the true objective function and the predicted function but on the Fig 7 after 8 iterations you can see how well out optimization works.

This is an example of how the SMBO i.e., Bayesian optimization along with TPE works:
For this implementation we mainly concentrated on the hyperopt and other python libraries (NumPy, pandas, matplotlib,seaborn).



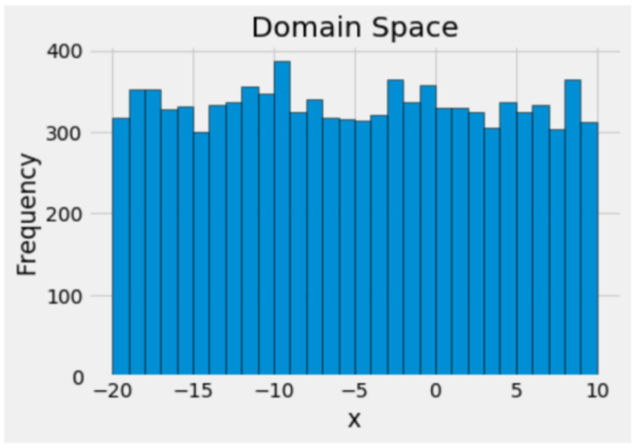
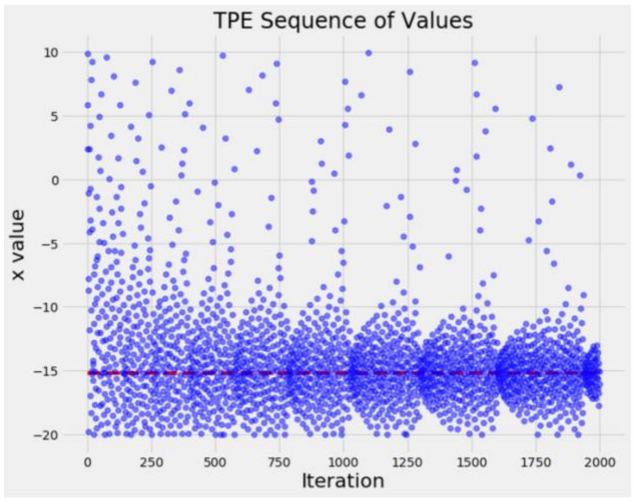


Fig 8 **Fig 9**

The above-mentioned fig 8 indicates our polynomial objective function (x^4 +20x^3 -6x^2 -20x +10) where minimum occurs at (-15.1652, -896.4567) and fig 9 indicates the domain space for th e given objective function with 30 bins.



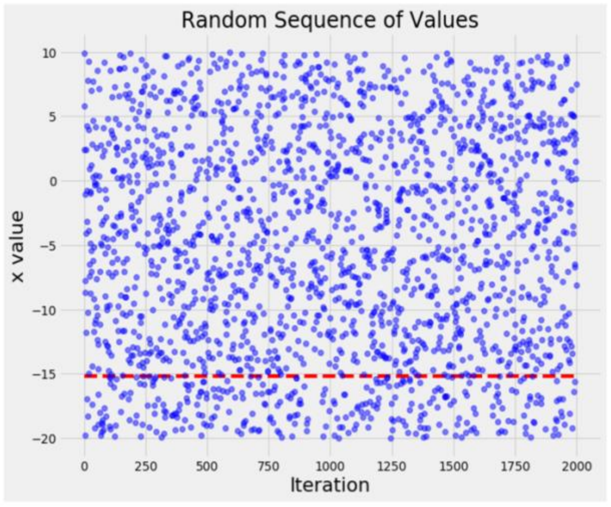
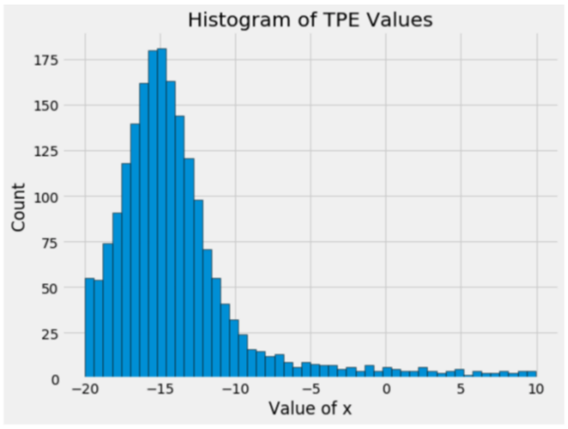


Fig 11 **Fig 12**

Both the above figures i.e., the Bayesian optimization and random sequence show that the minim um occurs at around 15.17 but it is clearly visible that values are much concentrated and become clustered at around the minimum in Fig 11 whereas the it is randomly chosen in the random sequ ence throughout. These figures are represented in the form of blue circle markers.



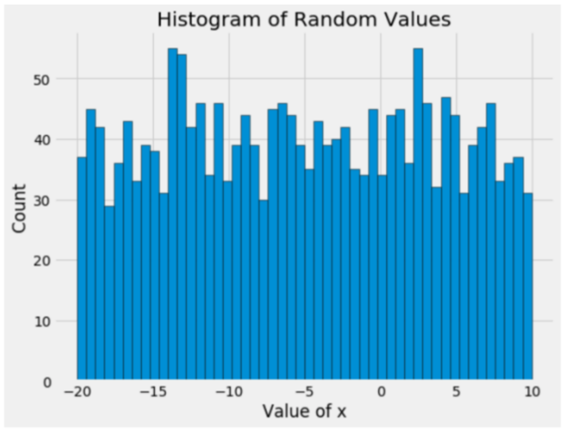
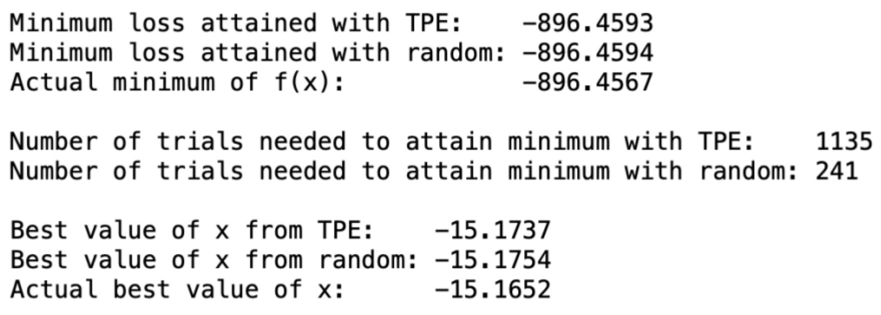
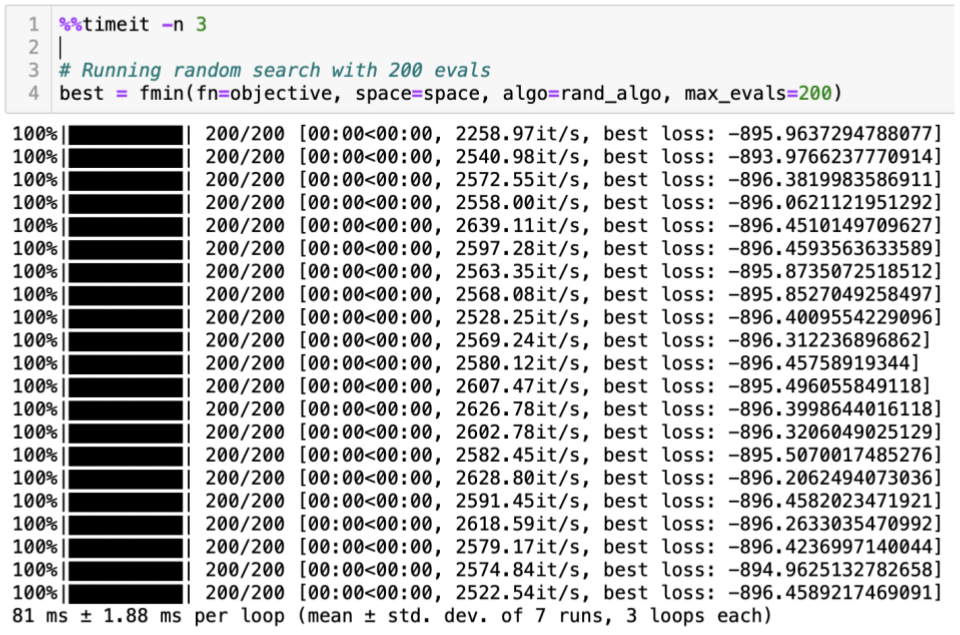


Fig 13 **Fig 14**

Hereby, from the above figures it is easy to predict in which range our hyperparameters fall in fro m Fig 13 i.e., Bayesian optimization with TPE when compared to an almost even range distributi on of Random search from Fig 14.

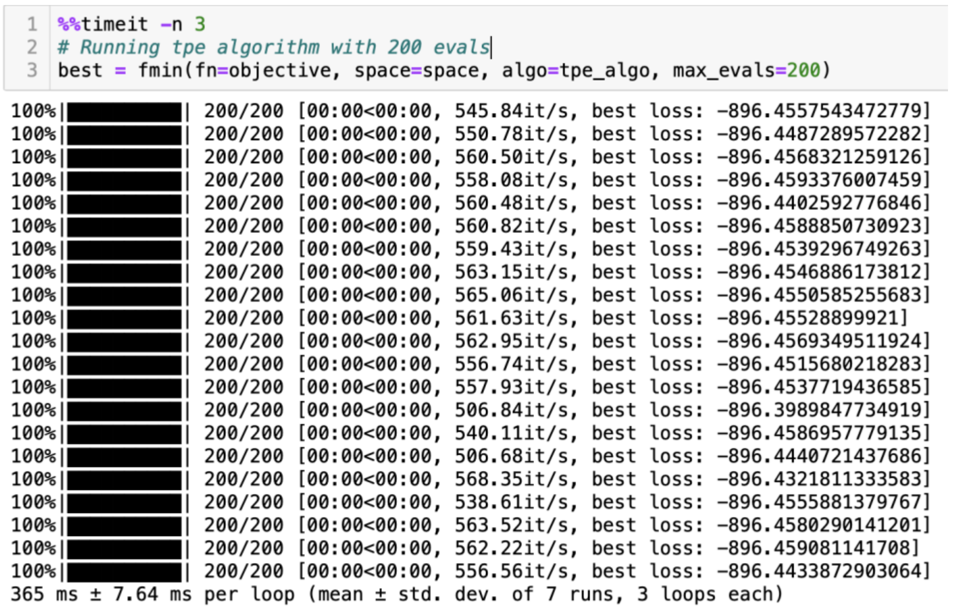


```
Minimum loss attained with TPE:    -896.4593
Minimum loss attained with random: -896.4594
Actual minimum of f(x):            -896.4567

Number of trials needed to attain minimum with TPE:    1135
Number of trials needed to attain minimum with random: 241

Best value of x from TPE:    -15.1737
Best value of x from random: -15.1754
Actual best value of x:      -15.1652
```



```
%%timeit -n 3

# Running random search with 200 evals
best = fmin(fn=objective, space=space, algo=rand_algo, max_evals=200)
```

```
100%|██████████| 200/200 [00:00<00:00, 2258.97it/s, best loss: -895.9637294788077]
100%|██████████| 200/200 [00:00<00:00, 2540.98it/s, best loss: -893.9766237770914]
100%|██████████| 200/200 [00:00<00:00, 2572.55it/s, best loss: -896.3819983586911]
100%|██████████| 200/200 [00:00<00:00, 2558.00it/s, best loss: -896.0621121951292]
100%|██████████| 200/200 [00:00<00:00, 2639.11it/s, best loss: -896.4510149709627]
100%|██████████| 200/200 [00:00<00:00, 2597.28it/s, best loss: -896.4593563633589]
100%|██████████| 200/200 [00:00<00:00, 2563.35it/s, best loss: -895.8735072518512]
100%|██████████| 200/200 [00:00<00:00, 2568.08it/s, best loss: -895.8527049258497]
100%|██████████| 200/200 [00:00<00:00, 2528.25it/s, best loss: -896.4009554229096]
100%|██████████| 200/200 [00:00<00:00, 2569.24it/s, best loss: -896.312236896862]
100%|██████████| 200/200 [00:00<00:00, 2580.12it/s, best loss: -896.45758919344]
100%|██████████| 200/200 [00:00<00:00, 2607.47it/s, best loss: -895.496055849118]
100%|██████████| 200/200 [00:00<00:00, 2626.78it/s, best loss: -896.3998644016118]
100%|██████████| 200/200 [00:00<00:00, 2602.78it/s, best loss: -896.3206049025129]
100%|██████████| 200/200 [00:00<00:00, 2582.45it/s, best loss: -895.5070017485276]
100%|██████████| 200/200 [00:00<00:00, 2628.80it/s, best loss: -896.2062494073036]
100%|██████████| 200/200 [00:00<00:00, 2591.45it/s, best loss: -896.4582023471921]
100%|██████████| 200/200 [00:00<00:00, 2618.59it/s, best loss: -896.2633035470992]
100%|██████████| 200/200 [00:00<00:00, 2579.17it/s, best loss: -896.4236997140044]
100%|██████████| 200/200 [00:00<00:00, 2574.84it/s, best loss: -894.9625132782658]
100%|██████████| 200/200 [00:00<00:00, 2522.54it/s, best loss: -896.4589217469091]
81 ms ± 1.88 ms per loop (mean ± std. dev. of 7 runs, 3 loops each)
```



```
%%timeit -n 3
# Running tpe algorithm with 200 evals
best = fmin(fn=objective, space=space, algo=tpe_algo, max_evals=200)
```

```
100%|██████████| 200/200 [00:00<00:00, 545.84it/s, best loss: -896.4557543472779]
100%|██████████| 200/200 [00:00<00:00, 550.78it/s, best loss: -896.4487289572282]
100%|██████████| 200/200 [00:00<00:00, 560.50it/s, best loss: -896.4568321259126]
100%|██████████| 200/200 [00:00<00:00, 558.08it/s, best loss: -896.4593376007459]
100%|██████████| 200/200 [00:00<00:00, 560.48it/s, best loss: -896.4402592776846]
100%|██████████| 200/200 [00:00<00:00, 560.82it/s, best loss: -896.4588850730923]
100%|██████████| 200/200 [00:00<00:00, 559.43it/s, best loss: -896.4539296749263]
100%|██████████| 200/200 [00:00<00:00, 563.15it/s, best loss: -896.4546886173812]
100%|██████████| 200/200 [00:00<00:00, 565.06it/s, best loss: -896.4550585255683]
100%|██████████| 200/200 [00:00<00:00, 561.63it/s, best loss: -896.45528899921]
100%|██████████| 200/200 [00:00<00:00, 562.95it/s, best loss: -896.4569349511924]
100%|██████████| 200/200 [00:00<00:00, 556.74it/s, best loss: -896.4515680218283]
100%|██████████| 200/200 [00:00<00:00, 557.93it/s, best loss: -896.4537719436585]
100%|██████████| 200/200 [00:00<00:00, 506.84it/s, best loss: -896.3989847734919]
100%|██████████| 200/200 [00:00<00:00, 540.11it/s, best loss: -896.4586957779135]
100%|██████████| 200/200 [00:00<00:00, 506.68it/s, best loss: -896.4440721437686]
100%|██████████| 200/200 [00:00<00:00, 568.35it/s, best loss: -896.4321811333583]
100%|██████████| 200/200 [00:00<00:00, 538.61it/s, best loss: -896.4555881379767]
100%|██████████| 200/200 [00:00<00:00, 563.52it/s, best loss: -896.4580290141201]
100%|██████████| 200/200 [00:00<00:00, 562.22it/s, best loss: -896.459081141708]
100%|██████████| 200/200 [00:00<00:00, 556.56it/s, best loss: -896.4433872903064]
365 ms ± 7.64 ms per loop (mean ± std. dev. of 7 runs, 3 loops each)
```

Fig 15 Computation times and info about implementation

The above collection of figures indicates that both Bayesian optimization with TPE and Random search find the minima of the objective function, but it also clearly indicates that random search converges to obtaining the minimum in a very short span which is almost 4.5x faster than TPE and in less number of iterations as well. This is because Bayesian optimization takes time to com pute the minima considering the dependencies between the hyperparameters which is random in case of random search but certainly gives a clear understanding of effective parameters from the before mentioned Fig 13 and Fig 14 based on concept of Fig 6 and Fig 7.

The advantage of Bi-LSTM which is storing of sequences from both the sides for longer duration for better predictability and the hyperparameter optimization technique may give drastic changes to the performance of the learning model.

Now it is clear from both the approaches that the combinations of hyperparameter optimization a nd Bi-LSTM yields much more promising and effective means of predictability in time series data analysis.

CONCLUSION:

Here we proposed a possible solution for the betterment of Time-series data analysis which deals with improving the performance of learning model by the combination of Bi-directional RNN + LSTM = Bi-LSTM + hyperparameter optimization (SMBO i.e., Bayesian optimization with TPE). This combination is proved to show better results when compared to other statistical models like ARIMA and GARCH which couldn't solve the problem of short-term memory according to paper [1]. It is shown that Bayesian optimization is much better than random search by proving how the dependency between hyperparameters helps in increasing the accuracy and reducing the complexity of the learning model. Thus model-based approach helps in the construction of a surrogate model which performs better with hyperparameter space than that of real space and thus less prone to be caught in local minima. This type of model certainly helps in various real time situations like predicting cyber-attack rates, gives information about which parameters actually affect the dataset, illustration of possible ways to predict climate changes, in the combination of CNN for image restoration, frame prediction in videos etc., etc., Further research on time series analysis can be made by the introduction of advanced concepts like reinforcement learning, other statistical methods like the combinations of powerful statistical models, introduction of ensemble learning, Bandit-based algorithm selection etc.,

REFERENCES:

[1] *https://link.springer.com/article/10.1186/s13635-019-0090-6*

[2] *http://proceedings.mlr.press/v28/bergstra13.pdf*

[3] *https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf*

[4] *https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f*

[5] *https://towardsdatascience.com/hyperparameters-optimization-526348bb8e2d*

[6] *https://machinelearningmastery.com/improve-deep-learning-performance/*

[7] *https://arxiv.org/abs/1906.11527*

[8] *https://medium.com/@elutins/grid-searching-in-machine-learning-quick-explanation-and-python-implementation-550552200596*

[9] *https://machinelearningmastery.com/how-to-scale-data-for-long-short-term-memory-networks-in-python/*