Comparing LSTM and ConvLSTM on multivariate time-series data generated from inviscid Burgers equation

Mohammad Daneshvar

May 6, 2020

1 Survey

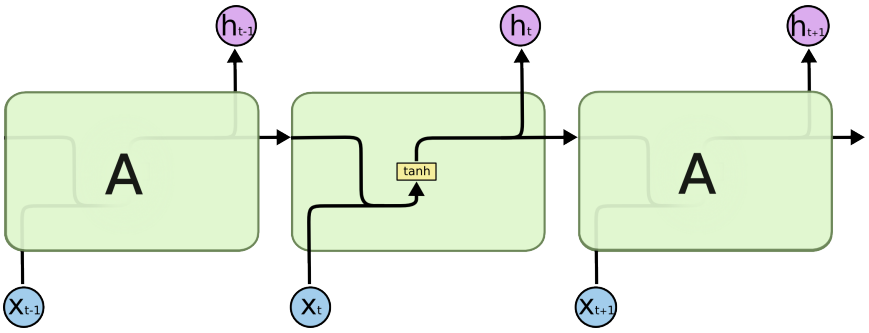
1.1 Recurrent Neural Networks

Recurrent Neural Networks are used for time-series models. They have recurrent layers with the purpose of transferring information from one time-step to the next one. They are used for speech recognition, handwriting recognition, translation, image captioning and time-series forecasting.

The simplest architecture for a recurrent neural network is Vanilla RNN. The dynamics of the model are given by :

$$\begin{aligned} h_t &= \sigma(W_h h_{t-1} + W_i x_t + b_h) \\ y_t &= tanh(W_o h_t + b_o) \end{aligned} \tag{1}$$

where σ is the sigmoid function, h_t is the state of recurrent state at time t, x_t is the input vector at time t and y_t is the output. Moreover weights W_o, W_i, W_h and biases term b_h, b_o are trained by the learning algorithm.



Picture is from `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

In practice simple RNN doesn't capture long-term dependency specially when number of steps is big and the reason is because it suffers from vanishing gradient problem. This problem arises from its dynamics. Information is always updated with no control on that.

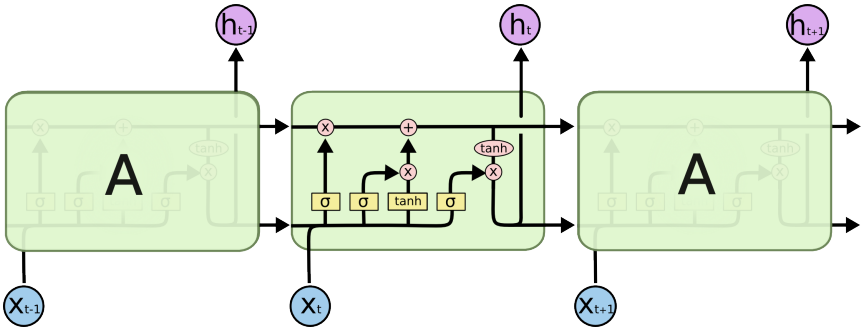
To resolve the previous problem, Long-Short Term memory or LSTM networks are devised. They solve the vanishing gradient problem by introducing a memory cell C_t and three gates:

1. forget gate (f_t): Decides what information to be thrown away
2. input gate (i_t): Controls how much information should be updated for the next step
3. output gate (o_t): Decides what information to be shown as output

Its dynamics are given by:

$$
\begin{aligned}
f_t &= \sigma(W_f h_{t-1} + W_{if} x_t + b_f) \\
i_t &= \sigma(W_i h_{t-1} + W_{ii} x_t + b_i) \\
o_t &= \sigma(W_o h_{t-1} + W_{io} x_t + b_o) \\
C_t^- &= \tanh(W_c h_{t-1} + W_{ic} x_t + b_c) \\
C_t &= f_t \odot C_{t-1} + i_t \odot C_t^- \\
h_t &= o_t \odot tanh(C_t) \\
y_t &= W_h h_t
\end{aligned}
\tag{2}
$$

Where C_t^- is a candidate for updating memory cell C_t at time t and $\odot$ stands for pointwise multiplication or Hadamard product.



Picture is from `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

The equation

$$
C_t = f_t \odot C_{t-1} + i_t \odot C_t^- \tag{3}
$$

explains how much information should be updated or thrown away at time t to get C_t. This is the essential part of LSTM for solving vanishing gradient problem because now it controls information that enters the memory cell at time t and memory cell is overwritten when it's necessary, whereas in simple RNN it always updates the memory cell (h_t) at each step so it's likely that it loses important information over time.

1.2 Data-Driven Model of Inviscid Burgers Equation

Data-Driven Models are models where they use data to analyze specific problems where it's difficult or sometimes impossible to find a analytic solution for them. They have been widely used in various area of mathematics such as predicting trajectories of chaotic systems or spatio-temporal models driven by discretization of partial differential equations in space and time. One can apply different techniques such

as recurrent neural networks to reproduce and predict short-time or long-term statistics of those systems.

In this article I use experiments to show that the combination of neural networks such as ConvLSTM network can produce a better result than a simple or deep LSTM network. I have used data generated by inviscid Burgers-Hopf equations

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\frac{u^2}{2} = 0$$

over a periodic domain of size L, by discretizing the equation in space using the difference method:

$$\frac{d}{dt}u_i + \frac{F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}}}{\Delta x} = 0$$

where $i \in 0, 1, ..., N-1$, $\Delta x = \frac{L}{N}$ and

$$F_{i+\frac{1}{2}} = \frac{1}{6}(u_i^2 + u_i u_{i+1} + u_{i+1}^2)$$

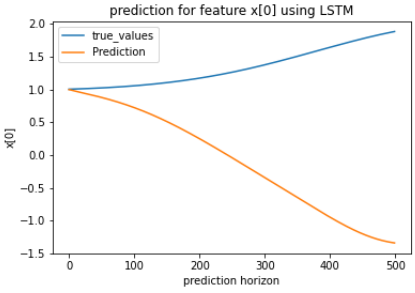
$$F_{i-\frac{1}{2}} = \frac{1}{6}(u_i^2 + u_i u_{i-1} + u_{i-1}^2).$$

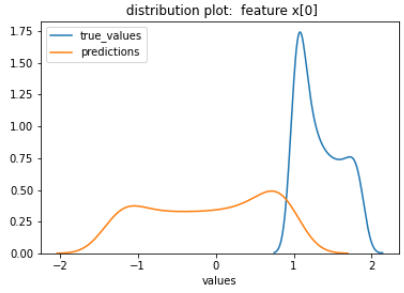
Then by local averaging u_i over n neighborhood we can define a new variable x_i as

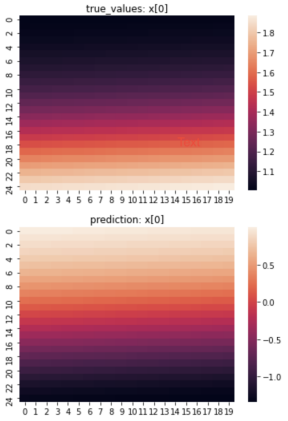
$$x_i = \frac{1}{n}\Sigma_{ni}^{n(i+1)-1} u_i$$

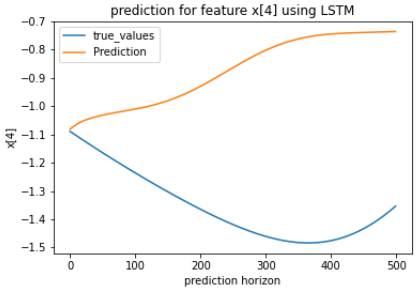
where $i \in \{0, 1, ..., \frac{n}{N} - 1\}$. see [1] for more details.
Data is generated using parameters $L = 100, N = 256, n = 16$ and $\Delta t = 0.02$ So data set includes samples of $x_i(t)$ for $i \in \{0, 2, ..., 15\}$ with step size $\Delta t = 0.02$ by discretizing the equation in time.

Data includes 16 features (spatial features) and the sample size is 320000 for each feature. First I used a deep layer LSTM network to predict trajectories and long-term statistics of data. The results show that LSTM doesn't do well on this data. I have used a 4-layer LSTM with time step = 3 and number of hidden units= 50 and 100 epochs. The train set includes 319500 and 500 samples are used for the test set.
Plots are for features x_0, x_4, x_5 and x_{12} where predictions are the worst among all other features. In plots, x-axis shows prediction horizon where step-size is $\Delta t = 0.02$. I have only used 500 steps for predictions. Data is normalized before processing.

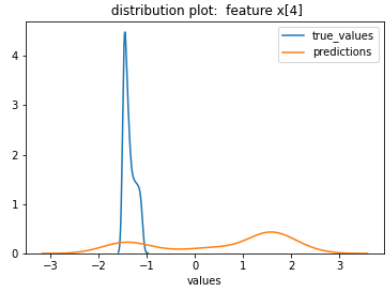


,


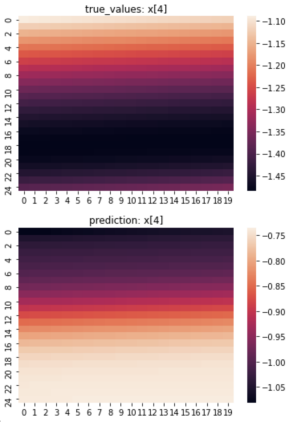
,


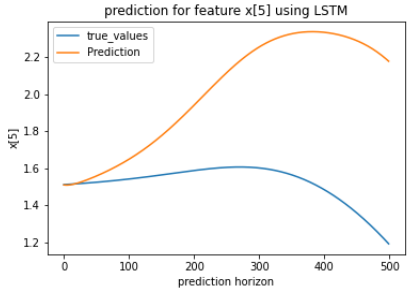


,

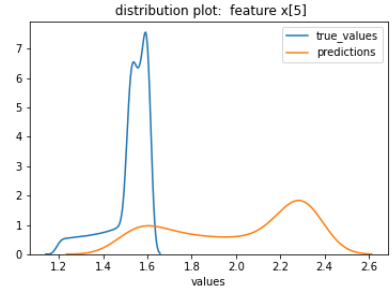


,

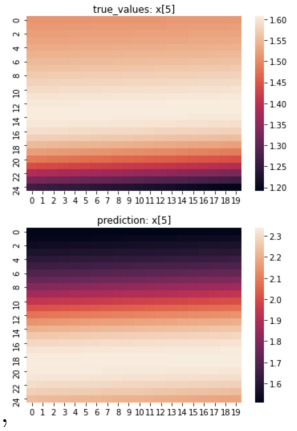


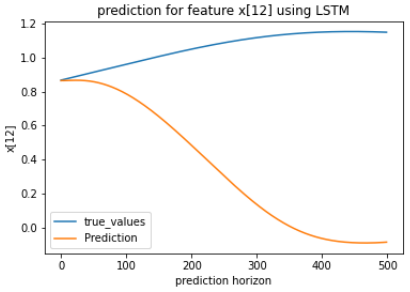


,

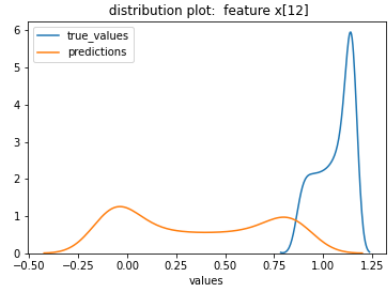


,

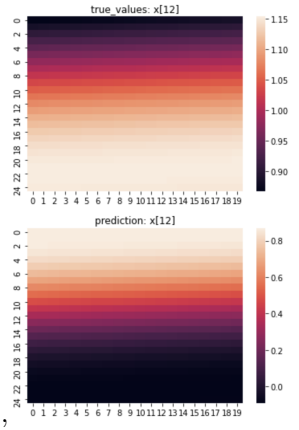




,



,



As we can see from above plots that LSTM was not able to produce close short-term evolution or long-term statistics of data generated from Inviscid Burgers equation. In the next section we try to produce better results for at least 50 steps.

2 Experiments

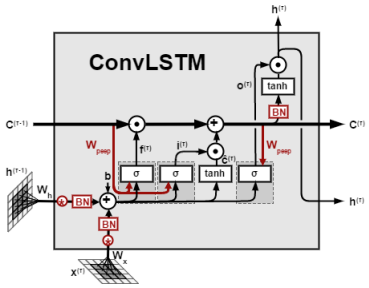
LSTM neural networks are very powerful on word prediction where input vectors are one-hot vectors which are sparse matrices. So it might be a good idea if we try to make sparse matrices of original data to see how accuracy on test-set changes. Moreover if we make 2D-images or matrices where c columns (assuming we have c columns in the matrix) are sequence of original data, then we are actually feeding a big sequence of data (a sequence of length c) one at a time instead of only one. So if LSTM learns these images, then only one-step close prediction for images means c-step close predictions for original data. We can increase c to get more accurate prediction steps however it might reduces the number of images and reduce the accuracy on training set. For this task I used a ConvLSTM which I breifly explained here.

2.1 Convolutional LSTM

Convolutional LSTMs (or ConvLSTM) are one special type of LSTM networks where linear combination of input vectors or hidden states are replaced by convolutions. In this network input can be a series of images. They are useful for object detection and motion prediction in autonomous cars.

$$
\begin{aligned}
f_t &= \sigma(W_f * h_{t-1} + W_{if} * x_t + b_f) \\
i_t &= \sigma(W_i * h_{t-1} + W_{ii} * x_t + b_i) \\
o_t &= \sigma(W_o * h_{t-1} + W_{io} * x_t + b_o) \\
C_t^- &= \tanh(W_c * h_{t-1} + W_{ic} * x_t + b_c) \\
C_t &= f_t \odot C_{t-1} + i_t \odot C_t^- \\
h_t &= o_t \odot tanh(C_t) \\
y_t &= W_h h_t
\end{aligned}
\tag{4}
$$

where $*$ stands for convolution operator as we have seen in CNN architecture and again $\odot$ stands for Hadamard product.



Picture is downloaded from website: `https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7`

2.2 Network Parameters

I have created one-channel images with 16 rows. This is the number of features, so each row corresponds to one feature. We can change the number of columns but for this experiment I used 50 columns for each image (matrix). So total number of images becomes $320000/50 = 6400$. This reduces the number of samples from 320000 in LSTM to 6400 in ConvLSTM, but still I observed a very low training error. I used one convolutional layer followed by batch normalization layer and a max-pooling layer. I used 30 kernels with kernel size $= (100, 1)$ for convolutional layer. One important hyperparameter is time-steps which is chosen empirically. I chose time-steps $= 2$, because it showed the best result. I trained the network with 100 epochs. The last 5 epoch in training the network, are shown here:

Epoch 95/100

- 9s - loss: 3.7384e-05

Epoch 96/100

- 9s - loss: 3.6531e-05

Epoch 97/100

- 9s - loss: 3.5723e-05

Epoch 98/100

- 9s - loss: 3.4972e-05

Epoch 99/100

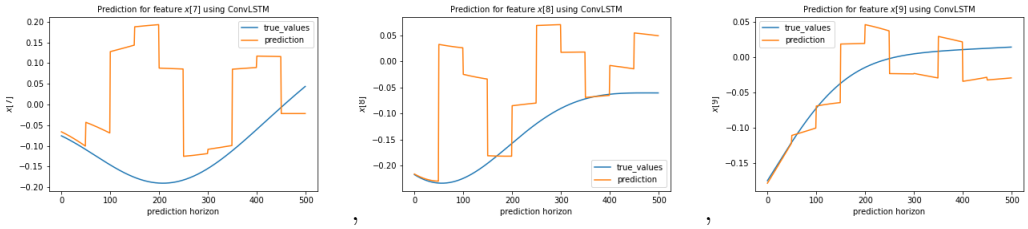
- 9s - loss: 3.4267e-05

Epoch 100/100

- 9s - loss: 3.3511e-05

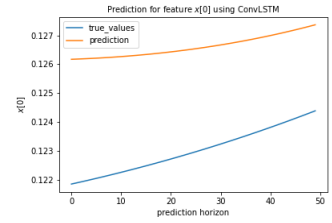
2.3 Results

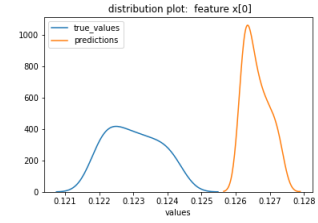
After training and predicting on test set, I have created one vector for test set and one vector for predictions (for each feature) by pasting columns of images. The ConvLSTM network with parameters described above, provides only one step accurate prediction as shown in plots below for features 7,8,9:

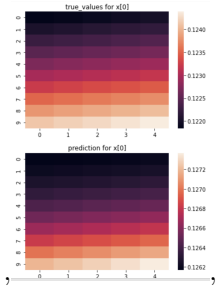


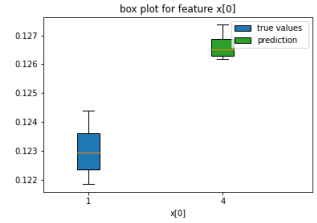
but this means 50 steps close predictions for original data. In above plots I have used 500 predictions similar to LSTM in previous section. But for plots on next pages I have only shown 50 predictions so that they can show the trend and close short-term evolution of original data for 50 steps with step-size $\Delta t = 0.02$.

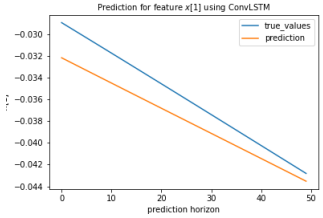
Plots for 11 features are presented here.

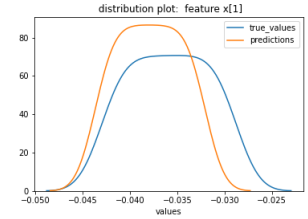

Prediction for feature x[0] using ConvLSTM
true_values
prediction
x[0]
prediction horizon

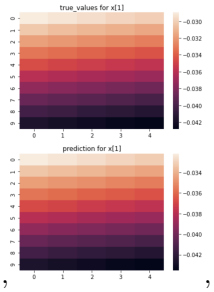

distribution plot: feature x[0]
true_values
predictions
values

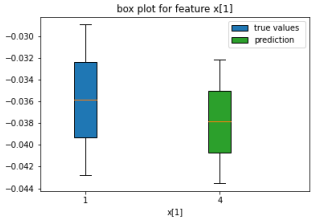

true_values for x[0]
prediction for x[0]

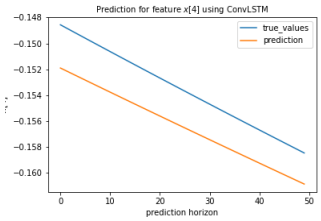

box plot for feature x[0]
true values
prediction
x[0]

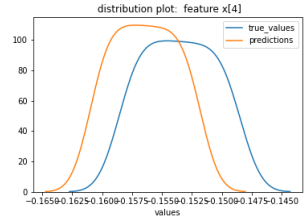

Prediction for feature x[1] using ConvLSTM
true_values
prediction
prediction horizon

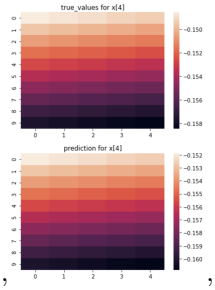

distribution plot: feature x[1]
true_values
predictions
values

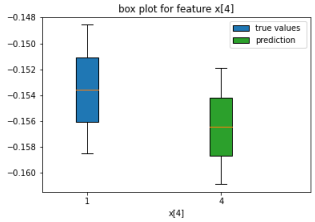

true_values for x[1]
prediction for x[1]

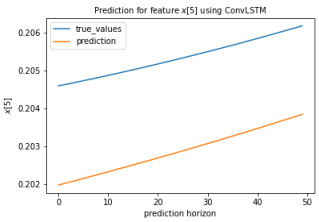

box plot for feature x[1]
true values
prediction
x[1]

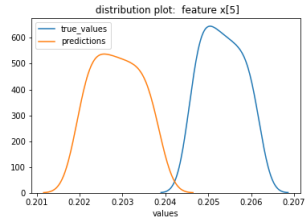

Prediction for feature x[4] using ConvLSTM
true_values
prediction
prediction horizon

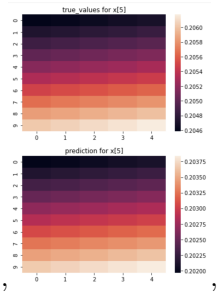

distribution plot: feature x[4]
true_values
predictions
values

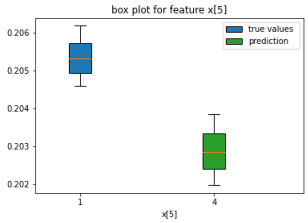

true_values for x[4]
prediction for x[4]

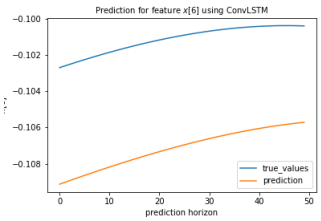

box plot for feature x[4]
true values
prediction
x[4]

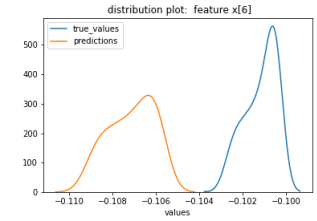

Prediction for feature x[5] using ConvLSTM
true_values
prediction
x[5]
prediction horizon

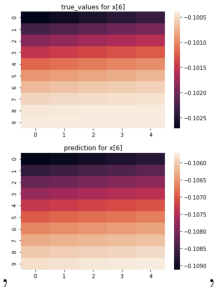

distribution plot: feature x[5]
true_values
predictions
values

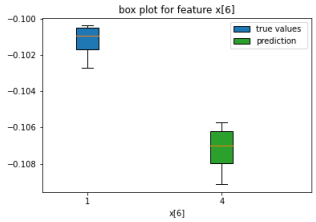

true_values for x[5]
prediction for x[5]

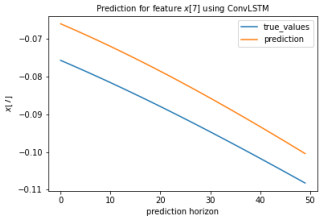

box plot for feature x[5]
true values
prediction
x[5]

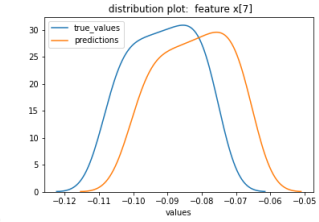

Prediction for feature x[6] using ConvLSTM
true_values
prediction
prediction horizon

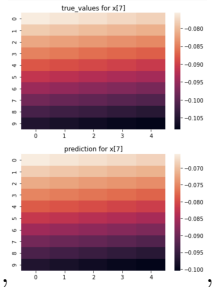

distribution plot: feature x[6]
true_values
predictions
values

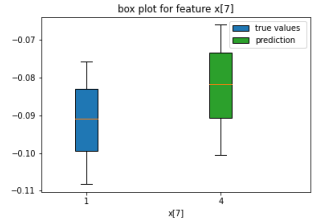

true_values for x[6]
prediction for x[6]

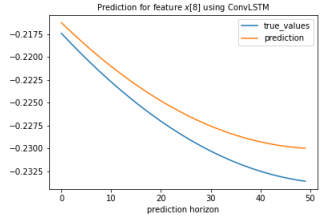

box plot for feature x[6]
true values
prediction
x[6]

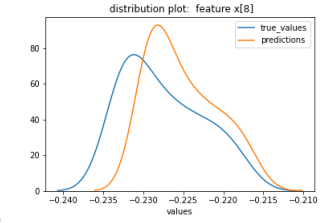

Prediction for feature x[7] using ConvLSTM
true_values
prediction
prediction horizon

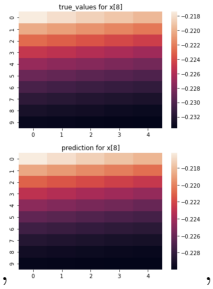

distribution plot: feature x[7]
true_values
predictions
values

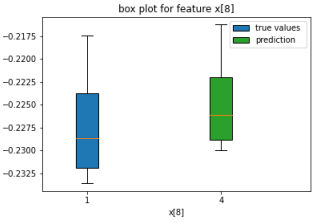

true_values for x[7]
prediction for x[7]

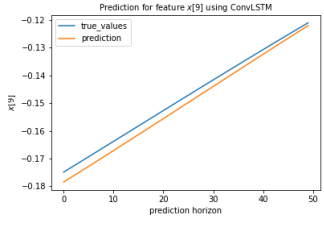

box plot for feature x[7]
true values
prediction
x[7]

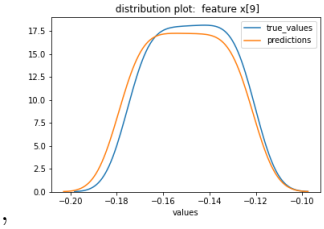

Prediction for feature x[8] using ConvLSTM
true_values
prediction
prediction horizon

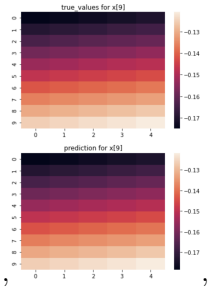

distribution plot: feature x[8]
true_values
predictions
values

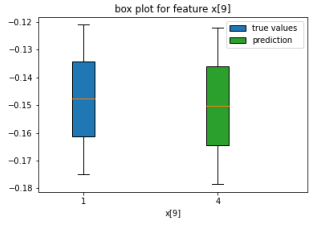

true_values for x[8]
prediction for x[8]

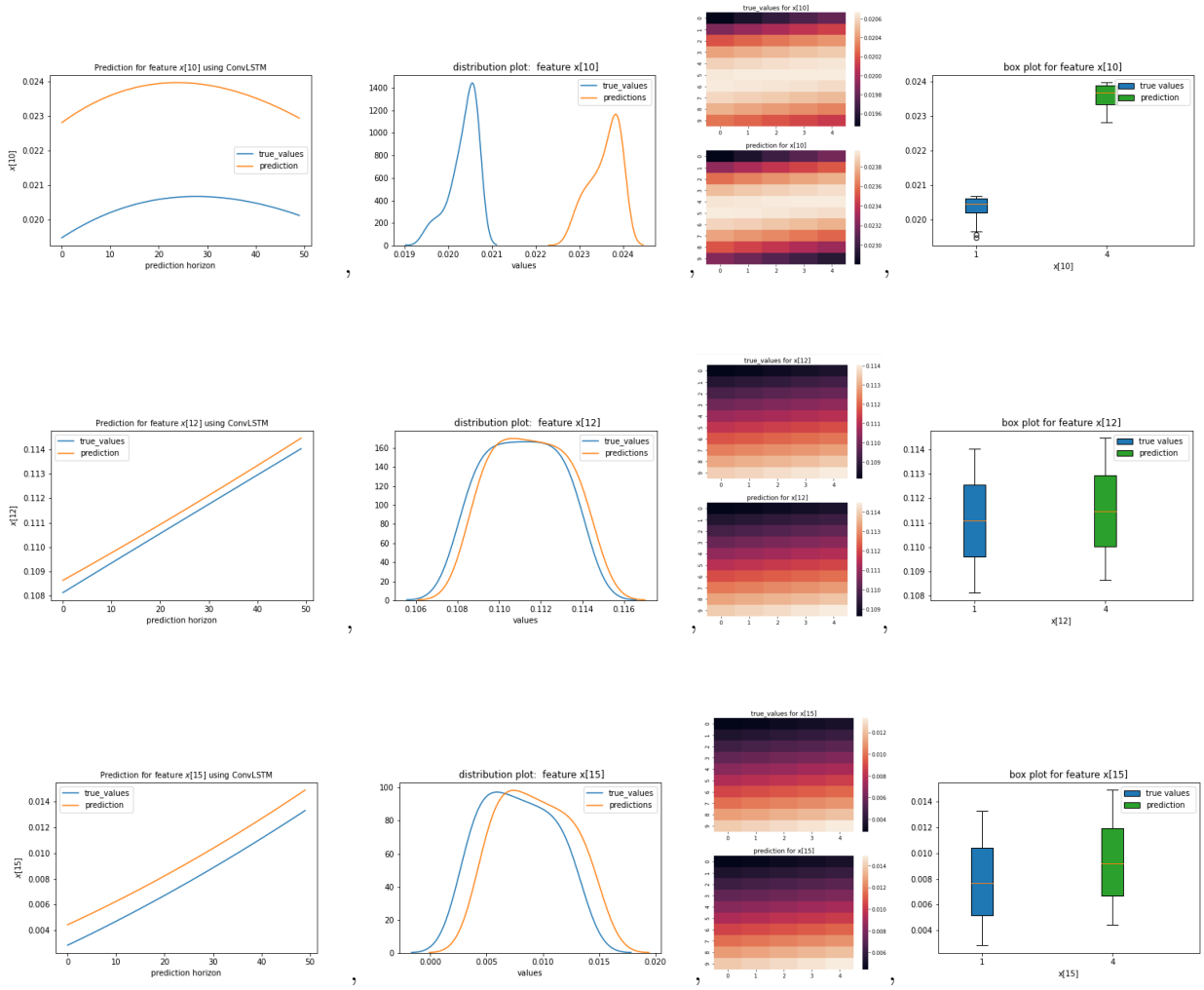

box plot for feature x[8]
true values
prediction
x[8]


Prediction for feature x[9] using ConvLSTM
true_values
prediction
prediction horizon


distribution plot: feature x[9]
true_values
predictions
values


true_values for x[9]
prediction for x[9]


box plot for feature x[9]
true values
prediction
x[9]



3 Conclusion

LSTM neural network is powerful in learning long-term steps but still it involves many parameters and hyperparameters which makes it a difficult task to acquire accurate predictions for long steps. We can produce more accurate predictions by feeding the LSTM with a sequence of data one at a time for each time step. For this to be done, one can create images of original data and use a Convolutional-LSTM network where linear combination of the inputs is replaced by convolution operations. Results in the second section of this project shows that, with this change in data we can produce at least 50 steps (number of columns in each image) close predictions for original data on data generated from Burgers-Hopf equation, whereas a deep layer LSTM is not able to produce accurate short-term predictions.

References

[1] Stamen Iankov Dolaptchiev, Ilya Timofeyev, Ulrich Achatz, *Subgrid-scale closure for the inviscid Burgers-Hopf equation.* Comm. Math Sci, 2013

[2] Stamen Iankov Dolaptchiev, Ilya Timofeyev, Ulrich Achatz, *Stochastic closure for local averages in the finite-difference discretization of the forced Burgers equation.* Theor. Comput. Fluid Dyn, 2013

[3] Soon Hoe Lim, Ludovico Theo Giorgini, Woosok Moon, J.S. Wettlaufer, *Predicting Rare Events in Multiscale Dynamical Systems using Machine Learning* . arXiv preprint arXiv:1908.03771, 2019.

[4] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting.* . arXiv:1506.04214v2 [cs.CV] 19 Sep 2015.

[5] S.L. Brunton and J.N. Kutz. *Data-driven Science and Engineering: Machine Learning, Dynamical Systems, and Control* . Cambridge University Press, 2019.