Dual-stage Attention-based recurrent Neural Network for Time Series Prediction.

Summary

In this work we address the Nonlinear autoregression exogenous (NARX) problem, where we try to predict the current value of time series based upon its previous value as well as the current and the past values of multiple driving series. The approach for this predictive modeling exercise is based on Dual stage attention-based recurrent neural network (DA-RNN) technique. The technique is developed by **Qin, et al. 2017** at the University of California, San Diego. This Machine Learning technique is widely used complex dynamic system analysis. In this work we present the concept and the application of the workflow to time series prediction in the stock market.

- **DA-RNN:**

The Dual stage attention-based recurrent neural network (DA-RNN) is an NARX (nonlinear autoregressive exogenous) model designed to:

- Capture the long-term temporal dependencies;
- Select the relevant driving series to make prediction.

The technique consists of two major stages:

Stage 1: it introduces an input attention mechanism to adaptively extract relevant driving series at each time step by referring to the previous encoder hidden state.

Stage 2: it uses temporal attention mechanism to select relevant encoder hidden states across all time steps.

- **Problem formulation:**

Given the previous values of the target series $(y_1, y_2, y_3, \dots, y_{t-1})$ with $y_{t-1} \in \mathrm{IR}$

As well as the current and the past value of n driving (exogenous) series $\mathbb{X} = (X^1, X^2, \dots, X^n)^T = (X_1, X_2, \dots, X_T) \in \mathrm{IR}^{n+T}$ where T is the length of window size.

$X^k = (x_1^k, x_2^k, \dots, x_T^k) \in \mathrm{IR}^T$ represent a driving series of length T

$X_t = (x_t^1, x_t^2, \dots, x_t^n)^T \in \mathrm{IR}^n$ represent a vector of n exogeneous (driving) input series at time t.

The NARX model aims to learn a non-linear mapping to the current value of the target series

$$\hat{y}_T = F(y_1, y_2, \dots, y_{T-1}, X_1, X_2, \dots, X_T)$$

Where F is the mapping nonlinear function to learn.

- ***LSTM/GRU for solving the problem:***

Recurrent Neural Network is a type of deep neural network specially designed for sequence modeling; however, it suffers from vanishing/exploding gradient and consequently it does not capture the long term dependencies. LSTM / GRU have overcome these limitations.

LSTM/GRU units Encoder-Decoder neural networks are popular for sequence modeling and have shown some success in machine translation. The idea is based on encoding the source sentence as fixed -length vector and use the decoder to generate a translation. However, the performance of these encoder decoder networks deteriorates rapidly as the length of the sequence increases. These has provided the bases to build attention-based encoder-decoder network that employs an attention mechanism to select part of the hidden states across all the time steps.

For time series prediction, we use two-stages attention based Recurrent Neural Network DA-RNN. In this workflow, the first attention mechanism aims at extracting the relevant driving series at each time step by referring to the previous encoder hidden state. The second attention mechanism is a temporal attention that aims at selecting relevant encoder hidden states across all time steps. These attention mechanisms can select the most relevant input features as well as capture the long term temporal dependencies of time series appropriately.

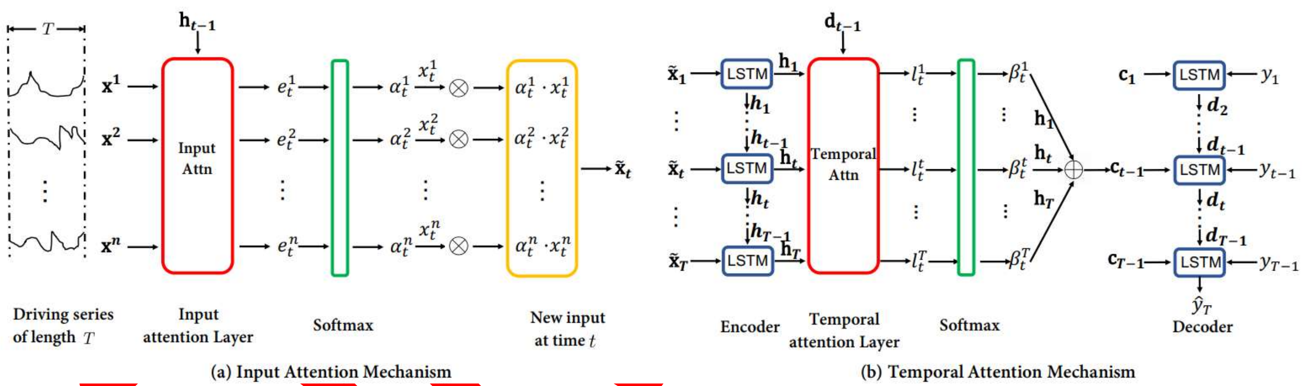

Figure 1:Illustration of dual-stage attention-based recurrent neural network. (a) The input attention mechanism computes the attention weights α_t^k for multiple driving series $\{x^1, x^2, x^3, ..., x^n\}$ conditioned on the previous hidden state h_{t-1} in the encoder and then feeds the newly computed $\hat{x}_t = (\alpha_t^1 x_t^1, \alpha_t^2 x_t^2, ..., \alpha_t^n x_t^n)^T$ into the encoder LSTM unit. (b) the temporal attention system computes the attention weights β_t^t based on the previous decoder hidden state d_{t-1} and represents the input information as a weighted sum of the encoder hidden states across all the time steps. The generated context vector c_t is then used as an input to the decoder LSTM unit. The output $\hat{y}_T$ of the last decoder LSTM unit is the predicted result.

- ***Model***

1. ***Encoder with input attention:***

The encoder is an RNN that encodes the input sequences. In this application given an input sequence $\mathbb{X} = (X_1, X_2, ..., X_t, ..., X_T)$ with $X_t \in \mathrm{IR}^n$ where n is the number of driving series, the encoder is implemented to learn the relationship f_1 between X_t *and* the hidden state h_t at time step t.

$$h_t = f_1(h_{t-1}, X_t)$$

Where $h_t \in \mathrm{IR}^m$ is the hidden state of the encoder at time t, m is the size of the hidden state. In this work the f_1 is a non-linear activation function LSTM unit to learn the long-term dependencies between the input series.

Each LSTM cell has a memory cell with the state s_t at time t equal to:

$$s_t = f_t \odot s_{t-1} + i_t \odot tanh(W_s[h_{t-1}; x_t] + b_s)$$

The cell state is controlled by there sigmoid gates:

- Forget gate : $f_t = \sigma(W_f[h_{t-1}; x_t] + b_f)$
- Input gate: $i_t = \sigma(W_i[h_{t-1}; x_t] + b_i)$
- Output gate: $o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$

The hidden state at time t is represented as follow:

$$h_t = o_t \odot tanh(s_t)$$

Here $[h_{t-1}; x_t] \in \mathrm{IR}^{m+n}$ is a concatenation of the previous hidden state h_{t-1} and the current input x_t

$W_f, W_i, W_o, W_s \in \mathrm{IR}^{m\times(m+n)}$ and $b_f, b_i, b_o, b_s \in \mathrm{IR}^m$ are parameters to learn.

The input attention mechanism at the level of the encoder aims at selecting the relevant driving series.

Given the k-th input driving series $X^k = (x_1^k, x_2^k, \dots, x_T^k) \in \mathrm{IR}^T$ we contract a deterministic attention model, a multilayer perceptron by using to previous hidden state h_{t-1} and the cell state s_{t-1} in the encoder LSTM unit with:

$$e_t^k = v_e^T tanh(W_e[h_{t-1}; s_{t-1}] + U_e X^k)$$

and

$$\alpha_t^k = \frac{exp(e_t^k)}{\sum_{i=1}^n exp(e_t^i)}$$

Where $v_e \in \mathrm{IR}^T$, $W_e \in \mathrm{IR}^{T\times 2m}$ and $U_e \in \mathrm{IR}^{T\times T}$ are parameters to learn.

α_t^k is the attention weight that measures the importance of the k-th time series at time t.

To ensure all the attentions weights sum up to 1, a softmax function is applied to e_t^k.

Using these attention weights, we can extract the driving series:

$$\tilde{x}_t = (\alpha_t^1 x_t^1, \alpha_t^2 x_t^2, \dots, \alpha_t^n x_t^n)^T$$

and the hidden state at time t:

$$h_t = f_1(h_{t-1}, \tilde{x}_t)$$

2. *Decoder with Temporal attention:*

The decoder is a LSTM based RNN that aims at decoding the encoded input information. To mitigate the rapid deterioration of encoder-decoder performance due to the length of sequences a temporal attention mechanism is used to decode adaptively select relevant encoder hidden states across all time steps.

The attention weight of each encoder hidden state at time t is calculated based upon the previous decoder hidden state $d_{t-1} \in \mathrm{IR}^p$ and the cell state of the LSTM unit $s'_{t-1} \in \mathrm{IR}^p$ with

$$l_t^i = v_d^T tanh(W_d[d_{t-1}; s'_{t-1}] + U_d h_i), \qquad 1 \leq i \leq T$$

and

$$\beta_t^i = \frac{exp(l_t^i)}{\sum_{j=1}^T exp(l_t^j)}$$

$[d_{t-1}; s'_{t-1}] \in \mathrm{IR}^{2p}$ is a concatenation of the previous hidden state and cell state of the LSTM unit.

$v_d \in \mathrm{IR}^m, W_d \in \mathrm{IR}^{m\times 2p}$ and $U_d \in \mathrm{IR}^{m\times m}$ are parameters to learn.

The attention weight β_t^i represents the importance of the i-th encoder hidden state for the prediction.

Each encoder hidden state h_i is mapped to a temporal component of the input, the attention mechanism computes the context vector c_t as a weighted sum of all the encoder hidden states $\{h_1, h_2, \ldots, h_T\}$:

$$c_t = \sum_{i=1}^{T} \beta_t^i h_i$$

At each time step the context vector is different. The weighted summed context vectors are combined with the target series $(y_1, y_2, \ldots, y_{T-1})$:

$$\tilde{y}_{t-1} = \widetilde{W}^T[y_{t-1}; c_{t-1}] + \tilde{b}$$

$[y_{t-1}; c_{t-1}] \in \mathrm{IR}^{m+1}$ is a concatenation of the decoder input y_{t-1} and the computed context vector c_{t-1}. $\widetilde{W} \in \mathrm{IR}^{m+1}$ and the $\tilde{b} \in IR$ map the concatenation to the size the decoder input. $\tilde{y}_{t-1}$ can be used for the update of the decoder hidden state at time t:

$$d_t = f_2(d_{t-1}, \tilde{y}_{t-1})$$

f_2 non-linear function as LSTM unit. The hidden state of the decoder is updated as follow:

$$f'_t = \sigma(W'_f[d_{t-1}; \tilde{y}_{t-1}] + b'_f)$$

$$i'_t = \sigma(W'_i[d_{t-1}; \tilde{y}_{t-1}] + b'_i)$$

$$o'_t = \sigma(W'_o[d_{t-1}; \tilde{y}_{t-1}] + b'_o)$$

$$s'_t = f'_t \odot s'_{t-1} + i'_t \odot tanh(W'_s[d_{t-1}; \tilde{y}_{t-1}] + b'_s)$$

$$d_t = o'_t \odot tanh(s'_t)$$

$[d_{t-1};\ \tilde{y}_{t-1}] \in \mathrm{IR}^{p+1}$ is a concatenation of the previous hidden state d_{t-1} and the decoder input $\tilde{y}_{t-1}$

$W'_f, W'_i, W'_o, W'_s \in \mathrm{IR}^{p\times(p+1)}$ and $b'_f, b'_i, b'_o, b'_s \in \mathrm{IR}^{p}$ are parameters to learn.

Here we aim to approximate the function

$$\hat{y}_T = F(y_1, y_2, \dots, y_{T-1}, X_1, X_2, \dots, X_T) = v_y^T(W_y[d_T; c_T] + b_w) + b_v$$

$[d_T; c_T] \in \mathrm{IR}^{p+m}$ is a concatenation of the decoder hidden state and the context vector.

$W_y \in \mathrm{IR}^{p\times(p+m)}$ and $b_w \in \mathrm{IR}^{p}$ map the concatenation to the size of the decoder hidden states. The linear function with weights $v_y \in \mathrm{IR}^{p}$ and bias $b_v \in IR$ provides the final prediction $\hat{y}_T$.

- ***Dataset and setup:***

To evaluate the performance of the DA-RNN we apply the workflow to a small dataset as a proof of concept. The purpose of the workflow here is to predict the change in SPY based on a selected set of stocks. Here we purposely look for stocks that have very low correlation with SPY to show the robustness of the workflow. The application of this technique is not limited to Stock market, it is also widely used for weather forecast and complex dynamic system analysis.

The dataset consists of about 1000 data points collected between January 2016 and July 2019. The target series is The SPDR S&P 500 ETF Trust that is used to track the S&P 500 stock market index. After careful analysis that of all the SP500 we only consider 229 possible driving time series for this analysis. **In order to challenge the model and its effectiveness in mapping the most appropriate nonlinear model to predict SPY we remove all the time series that have a correlation with SPY of 0.9 and higher from the list of driving time series.** For the purpose of this analysis 30% of the data is used for testing. Figure 2 shows the training data and the testing data. The test data is also used as the validation data.

This small dataset is collected from the public domain. Several procedure of data pre-processing is applied to the data to run the DA-RNN. Data preparation and pre-processing is not the presented here, however the full workflow includes a set of automated subroutines that facilitated data collection, preparation, and pre-processing as well as automated application of the DA-RNN for time series prediction. To learn more, please get contact yesser.nasser@icloud.com.
The Application of the workflow to a much larger dataset is not shared in this work since the dataset **is not available in the public domain**.

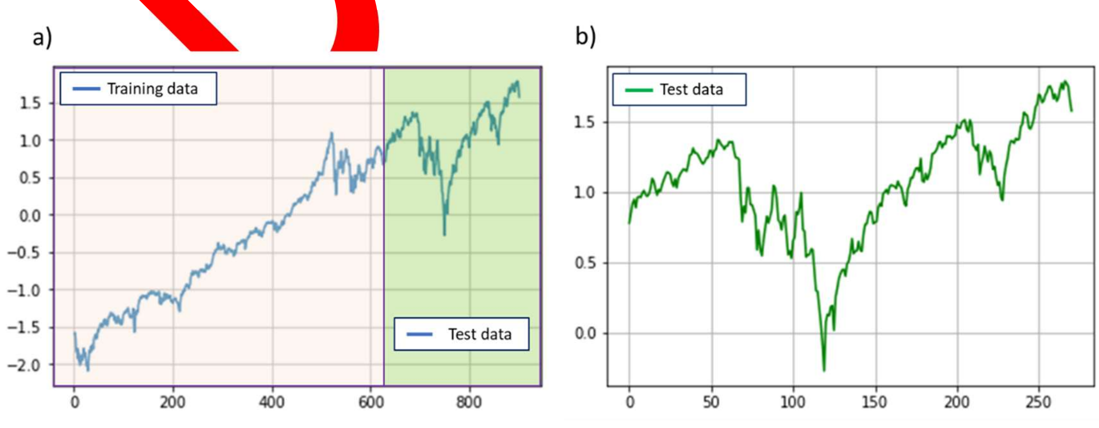

*Figure 2: **a)** The full dataset used for the modeling exercise, 30% of the data is used for testing. **b)** the test dataset.*

- ***Training procedure:***

70% of the data is used for training procedure. Given the size of the data we use minibatches of 32. Furthermore, we use the Adam optimizer with a learning rate of 0.002. To improve the model convergence to a minimum we schedule a learning rate reduction of 10% after each 200 iterations. For N number of training samples, the cost function is quantified using the mean squared error.

$$Cost(\,y_T, \hat{y}_T) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_T^i, y_T^i)^2$$

Parameters:

Three main parameters to consider in this workflow:

- Number of time steps in the window $T \in \{4,6,10,12\}$
- Size of the hidden states for the encoder $m \in \{16,32,64,128\}$
- Size of the hidden states for the decoder $p \in \{16,32,64,128\}$

- ***Training results:***

For this application we run several scenarios in order to evaluate the impact of these 3 main parameters. Window size of 6, and 64 hidden states for each, the encoder and decoder seem to lead to the best results. Here, we run our model for 70 epochs.

For visual illustration of the training procedure, in Figure 3 we plot the true data as well as the predictions from training and testing for different epochs. In Figure 3. A shows a large misfit between the true data and the initial prediction at Epoch 0 (for each epoch we ran 20 iterations). However, by epoch 10 (200 iterations) the model is showing a significant improvement with training prediction. The prediction with the testing (which is also used as validation data) is improving but with less accuracy. After multiple epochs (Figure 3.E and Figure 3.F) the model was able to generate better predictions for the test/validation data.

The evaluation of the loss during the training is showing in Figure 4. Here we show the loss for each iteration (Figure 4.a) as well as the loss for each epoch (Figure 4.b). The loss function with respect to epochs is smoother compared to the one for each iteration. Here we average the loss function of each 20 iterations to generate a loss for each Epoch.

Figure 5 shows a visual comparison between the prediction at epoch 1 and the prediction at epoch 70. Clearly the model have improved significantly with more training.

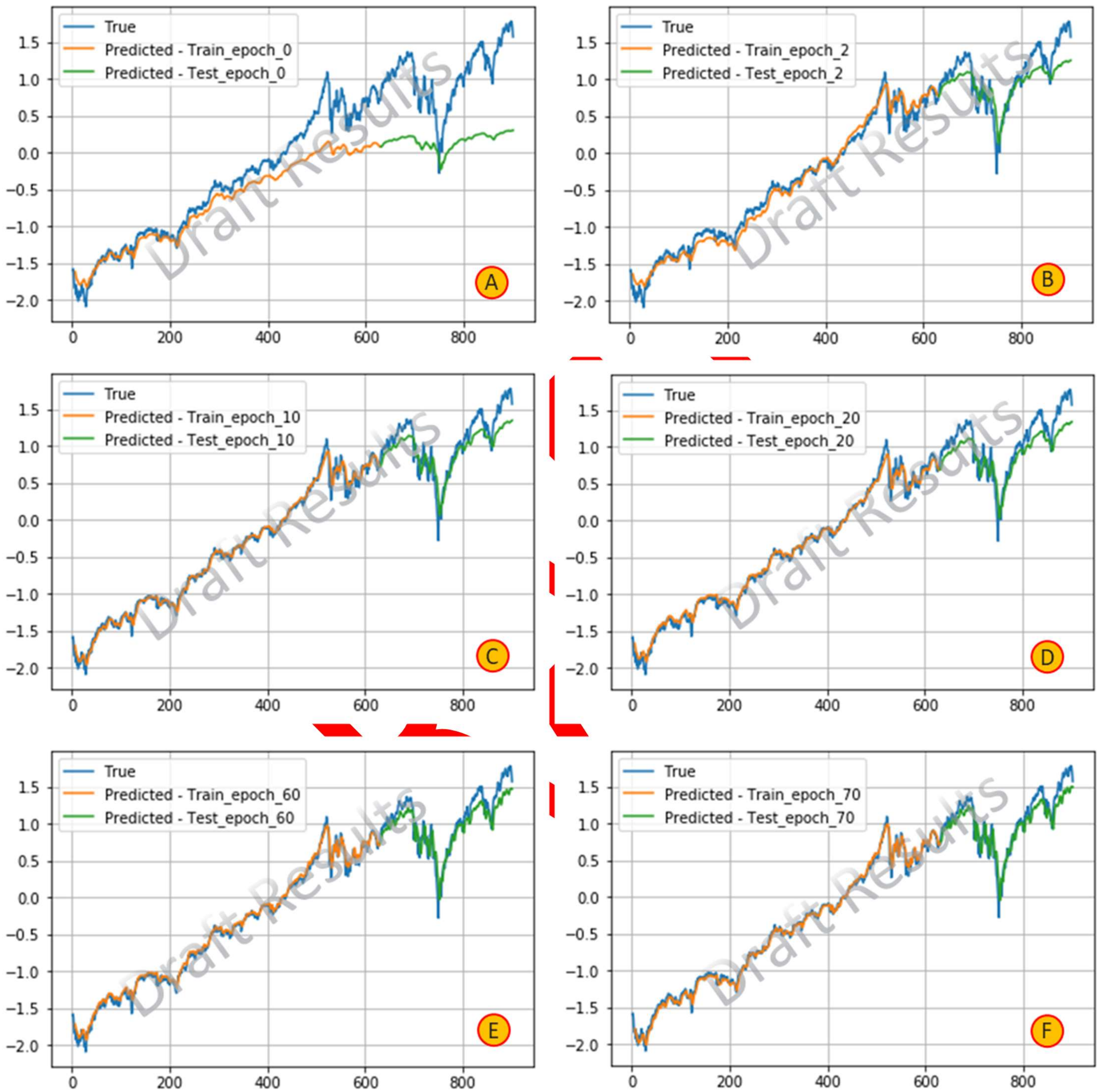

Figure 3: Summary of the training results, True data is shown in blue, orange shows the training prediction, green shows the test prediction after: ***A)*** *0 Epochs,* ***B)*** *2 Epochs,* ***C)*** *10 Epochs,* ***D)*** *20 Epochs,* ***E)*** *60 Epochs, and* ***F)*** *70 Epochs.*

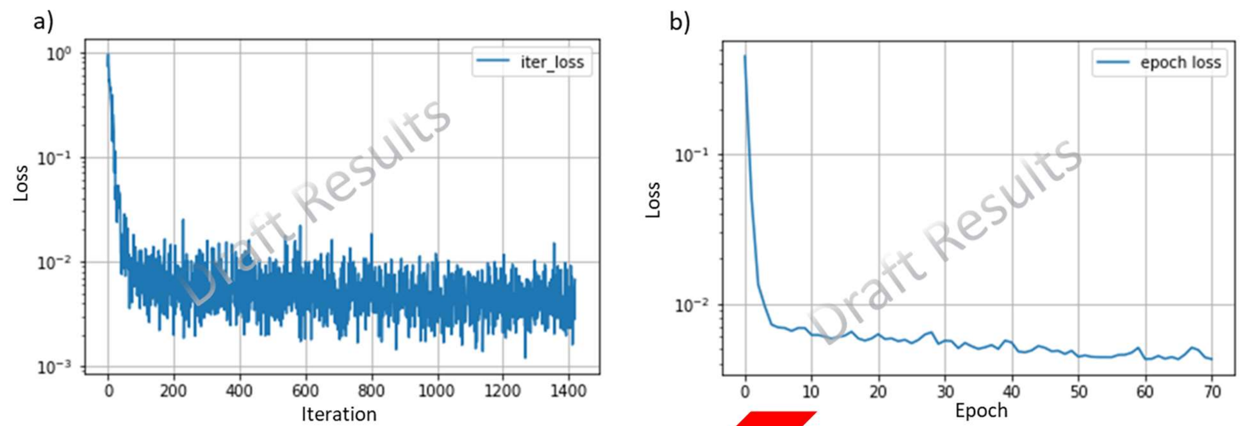

Figure 4: Loss with a) iterations and b) Epochs

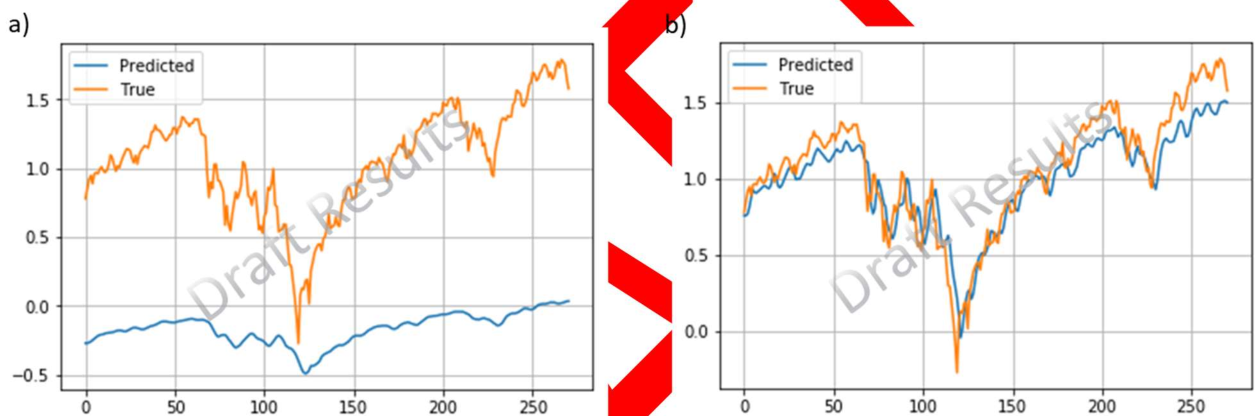

Figure 5: True data in blue and Prediction in orange after a) 1 Epoch and b) 70 Epochs

- ***Application to larger dataset:***

In this section we briefly discuss the application of the DA-RNN workflow to large data set to predict SP500. The dataset consists of 41000 data points and 87 driving time series to predict SP500. Figure 6 shows the summary of the results in term of training and test predictions as well as the loss for each iteration and epochs. In the case of large dataset, there is enough training data to reach a good match between the true data the training prediction; only 40 Epochs to achieve good prediction. However, the computation time is considerably large given the number of iterations for each of the data points.

Based on this analysis, the workflow is proven to be effective for both cases the small dataset (1000 data points with 229 driving time series) and the large data set (41000 data points and 87 driving series).

Here We only address the prediction of 1 time series at a time, however the workflow is built to hand the prediction of multiple time series given multiple and different driving sequences at the same time. (To learn more about the application please get in touch at: yesser.nasser@icloud.com)

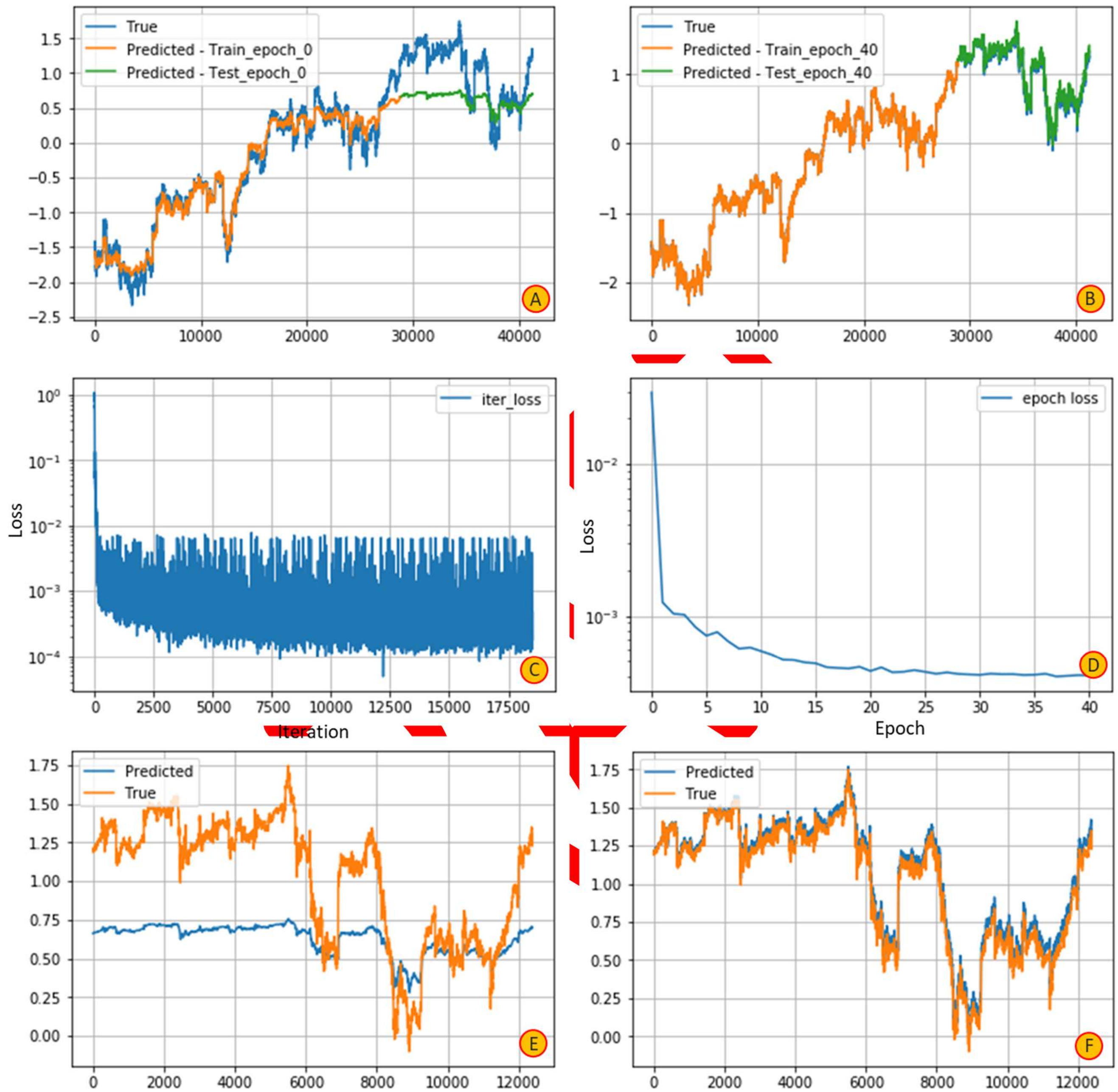

Figure 6: Summary results for the prediction of SP500. A) True data and prediction at Epoch 0, B) True data and prediction at Epoch 40, C) Loss for each iteration, D) Loss per Epoch, E) prediction after 1 Epoch, F) prediction after 40 Epochs.