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 values as well as the current and the past values of multiple driving series. The approach for this predictive modeling exercise is a 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 in complex dynamic system analysis. Here, we present the mathematical formulation and the application of the workflow to time series prediction of ETF Trust that is used to track the S&P 500 stock market index.

- **DA-RNN:**

The Dual stage attention-based recurrent neural network (DA-RNN) is an NARX (nonlinear autoregressive exogenous) workflow 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 to make final predictions.

- **Problem formulation:**

Given the previous values of the target series $(y_1, y_2, \dots, y_{t-1})$ 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 nonlinear function to learn.

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

Recurrent Neural Network is a special Deep Neural Network designed for sequence modeling; however, it suffers from vanishing/exploding gradient, consequently it does not capture the long-term dependencies. To overcome this limitation, LSTM (Long Short-Term Memory)/ GRU (Gated Recurrent Unit) are widely used to predict sequences.

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. This has raised the need 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. The second attention mechanism is a temporal attention that aims at selecting relevant encoder hidden states across all time steps to make final predictions. These attention mechanisms can select the most relevant input features as well as capture the long-term temporal dependencies of time series appropriately.

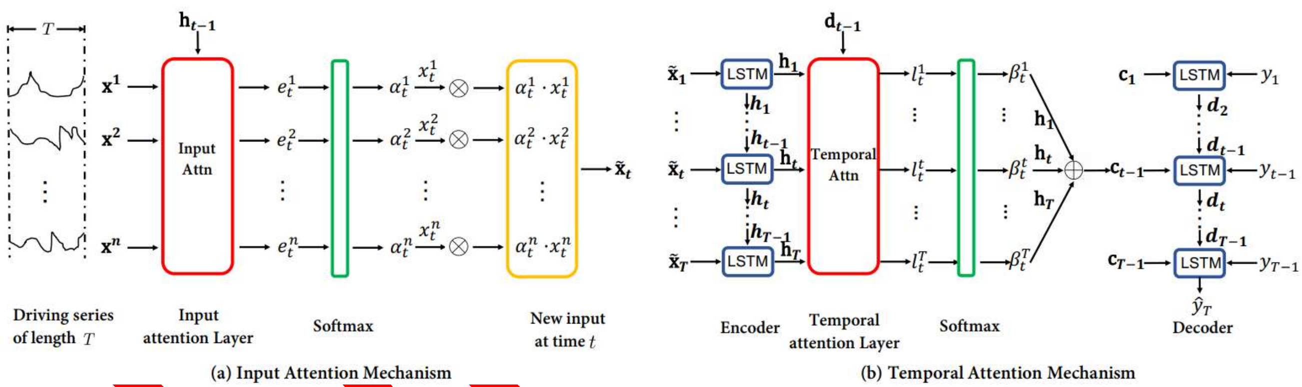

Figure 1:Illustration of dual-stage attention-based recurrent neural network. (a) The input attention mechanism computes the attention weights α_t^k for multiple driving series $\{x^1, x^2, x^3, \ldots, x^n\}$ conditioned on the previous hidden state h_{t-1} in the encoder and then feeds the newly computed $\hat{x}_t = (\alpha_t^1 x_t^1, \alpha_t^2 x_t^2, \ldots, \alpha_t^n x_t^n)^T$ 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 $\mathbb{X} = (X_1, X_2, \ldots, X_t, \ldots, X_T)$ with $X_t \in \mathrm{IR}^n$ where n is the number of driving series, the encoder is applied to learn the relationship f_1 between X_t *and* the hidden state h_{t-1} 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 states. here f_1 is a non-linear activation function LSTM unit designed to learn the long-term dependencies between the input series.

Each state cell 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 these 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:

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

where $[h_{t-1}; x_t] \in \mathrm{IR}^{m+n}$ 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 construct a deterministic attention model (multilayer perceptron) by referring to the 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:

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

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 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 $\{h_1, h_2, \dots, h_T\}$:

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

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

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

$[y_{t-1}; c_{t-1}] \in \mathrm{IR}^{m+1}$ 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 of the decoder input, then $\tilde{y}_{t-1}$ is used to update the decoder hidden state at time t:

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

f_2 is a non-linear function 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 F to obtain an estimate of the current output $\hat{y}_T$ with respect of all inputs as well as previous outputs.

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

$[d_T; c_T]\ \in \mathrm{IR}^{p+m}$ 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. 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 low correlation with target series 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 of all the SP500 we only consider 29 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 exclude all the time series that have a correlation with the SPY of 0.9 and higher from the list of driving time series.** During the training process we used 30% of the data for testing. Figure 2 shows the training data and the testing data.

This small dataset is collected from the public domain. Several procedure of data pre-processing is applied to the data prior to running the DA-RNN workflow. Data preparation and pre-processing steps are not presented here, however the full workflow includes a set of automated subroutines that facilitated data collection, preparation, and pre-processing. To learn more, please get contact yesser.nasser@icloud.com. The Application of the workflow to larger dataset is briefly presented at the end of the work. We should point out that an up-to-date large dataset of this kind **is not available for free in the public domain**. **Therefore, we are currently working on building an UpToDate larger dataset (min 10 months).**

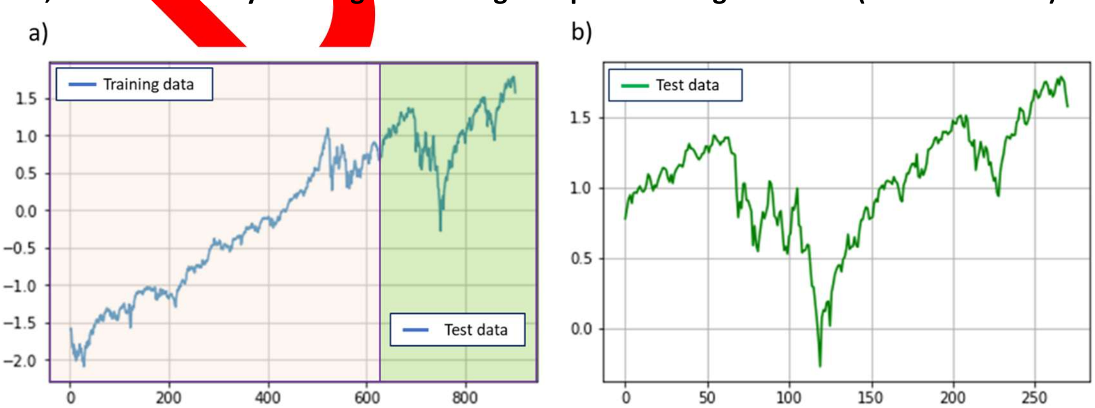

*Figure 2: **a)** The full dataset used for the modeling exercise, 30% of the data is used for testing. **b)** the test dataset.*

- ***Training procedure:***

70% of the data is used for the 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, we apply a 10% reduction to the learning rate after every 200 iterations. During this process, for N number of training samples we quantify the cost function as following:

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

- ***Parameters:***

In this workflow there are three main parameters to consider:

- 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 $p \in \{16,32,64,128\}$

The range of values for T, m, and p presented above are used to run a sensitivity analysis on the model predictions. We use these scenarios to optimize the window size T and the size of hidden states for encoder (m) / decoder (p) to achieve the best results.

- ***Training results:***

For this application we run several scenarios in order to evaluate the impact of these 3 main parameters. A window size of 6, and 64 hidden states for each, the encoder and decoder seem to lead to the best results after 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. Figure 3. A) shows a large misfit between the true data and the initial prediction at Epoch 1 (for each epoch we ran 20 iterations). However, by epoch 10 (200 iterations) the model is showing a significant improvement with training predictions. The prediction on the test data (which is also the 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. The loss for each epoch is generated by averaging the loss function of each 20 consecutive iterations.

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

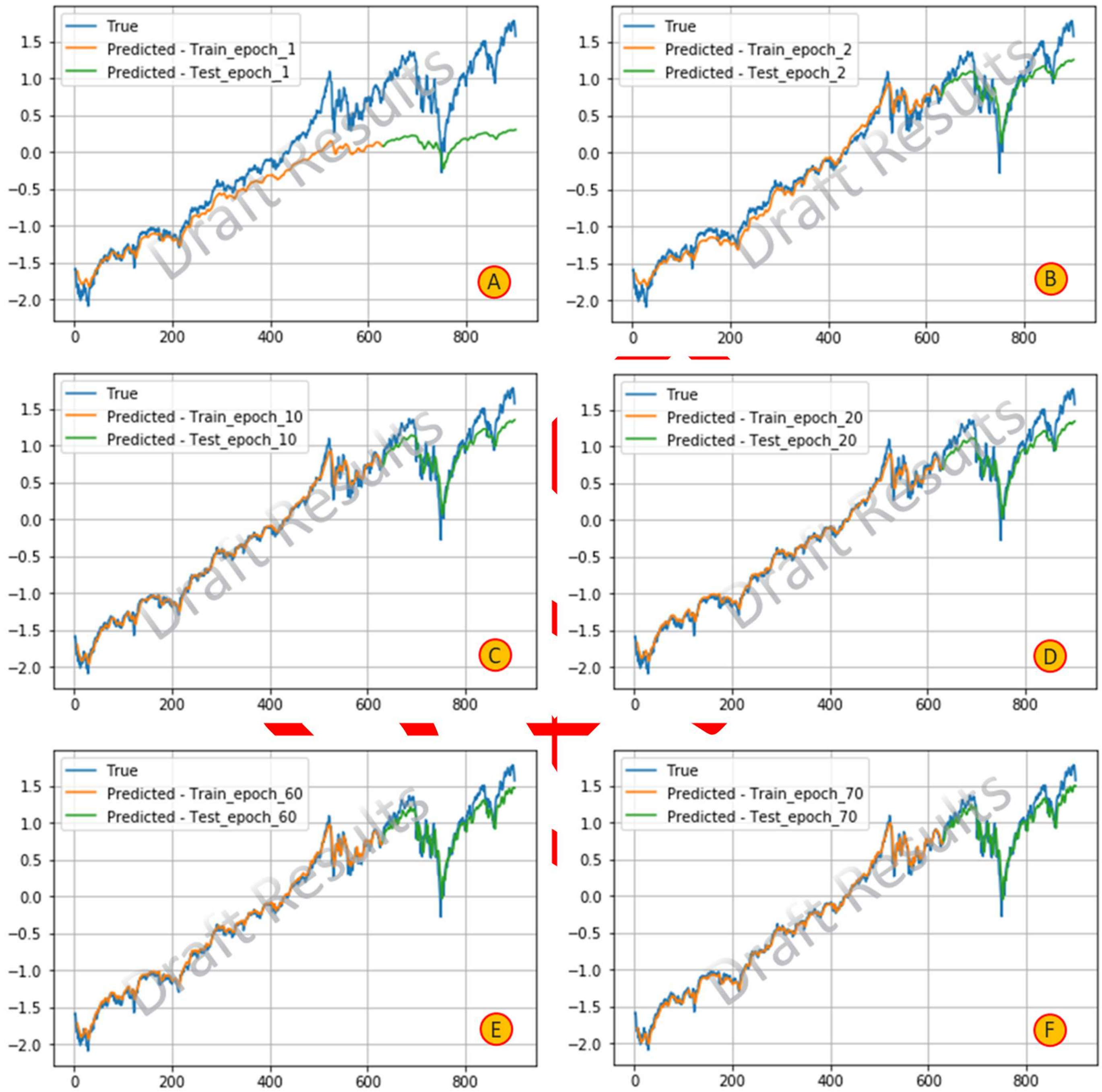

*Figure 3: Summary of the training results. True data, training predictions, and test predictions are shown in blue, orange, and green respectively. Prediction results after **A)** 1 Epoch, **B)** 2 Epochs, **C)** 10 Epochs, **D)** 20 Epochs, **E)** 60 Epochs, and **F)** 70 Epochs.*

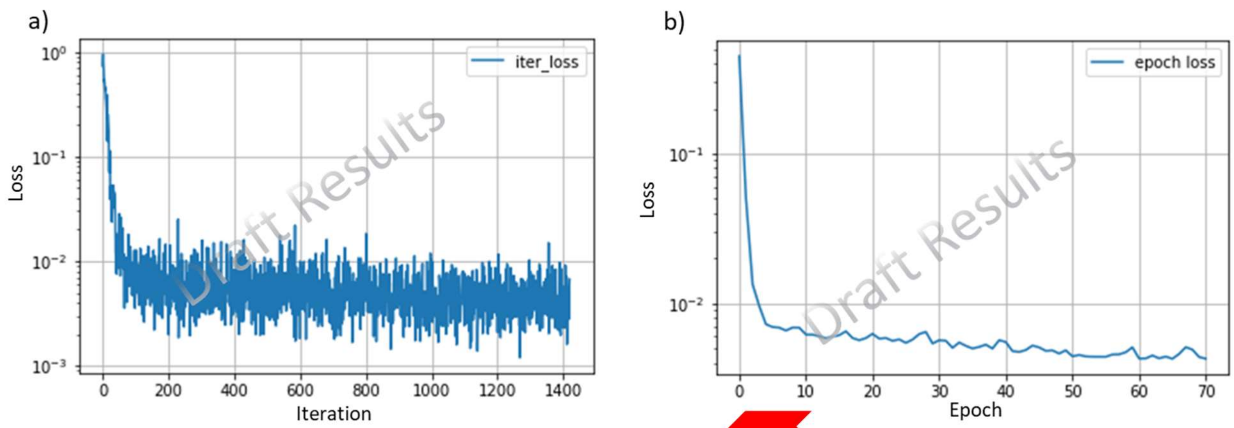

Figure 4: Loss with a) iterations and b) Epochs

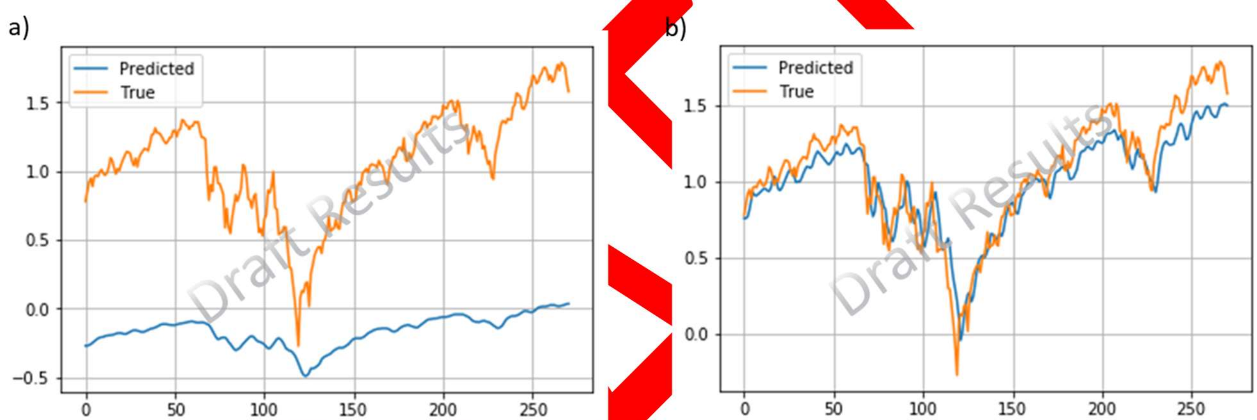

Figure 5: True data in blue and Prediction in orange after a) 1 Epoch and b) 70 Epochs

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

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

The analysis of the model predictions suggests that the workflow is proven to be effective for both cases, the small dataset (1000 data points with 29 driving time series) and the large dataset (41000 data points and 87 driving series). Here, the attention mechanisms are very crucial in selecting the most important input driving time series and the appropriate time window to achieve an accurate prediction of the target time series.

In this work 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 set of driving sequences at the same time. (To learn more about the application please get in touch at: yesser.nasser@icloud.com)

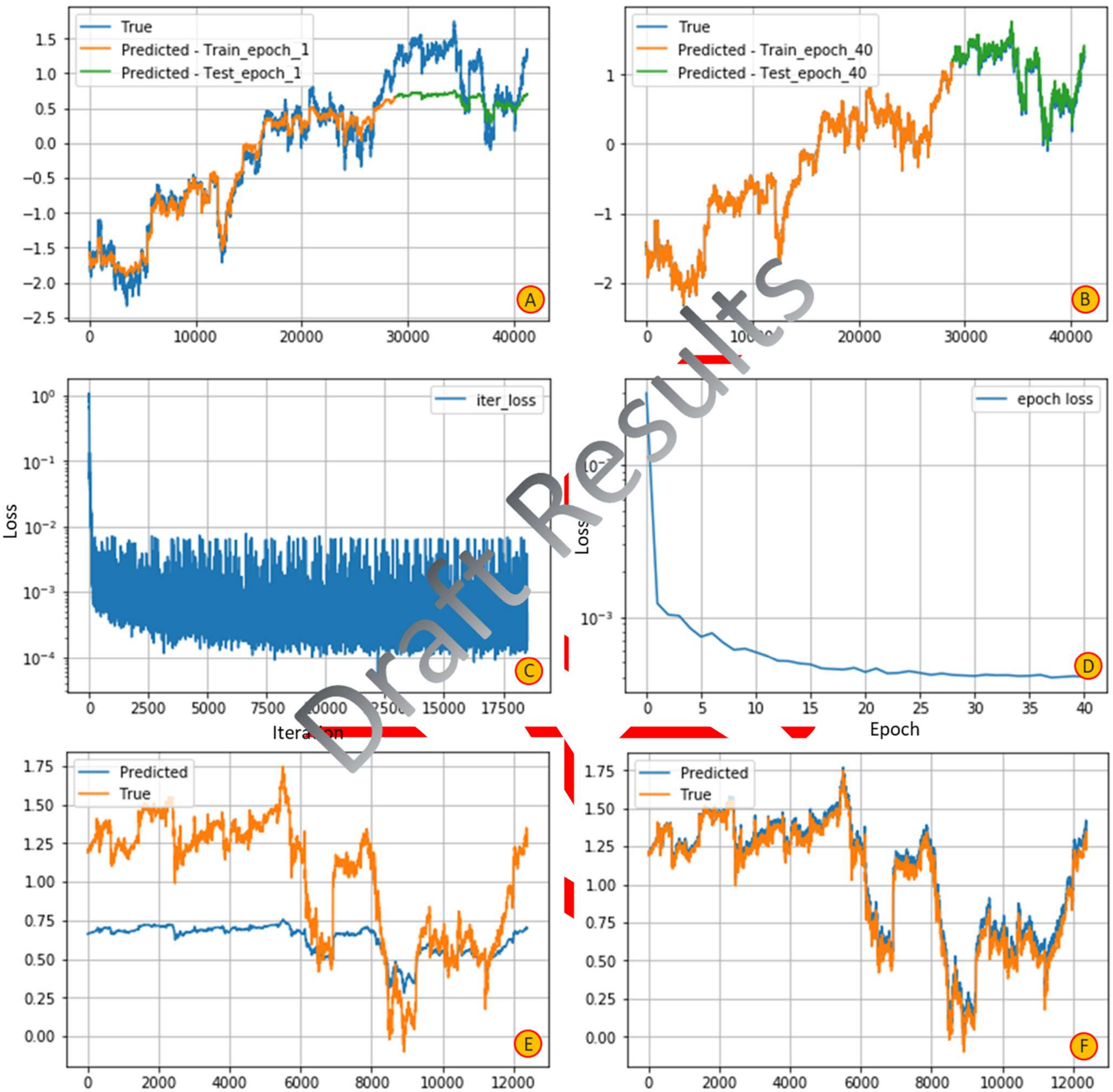

Figure 6: Summary of the results for the SP500 prediction. A) True data and prediction at Epoch 1, B) True data and prediction at Epoch 40, C) Loss for each iteration, D) Loss per Epoch, E) prediction after 0 Epoch, F) prediction after 40 Epochs