Time Series modelling using DA-RNN

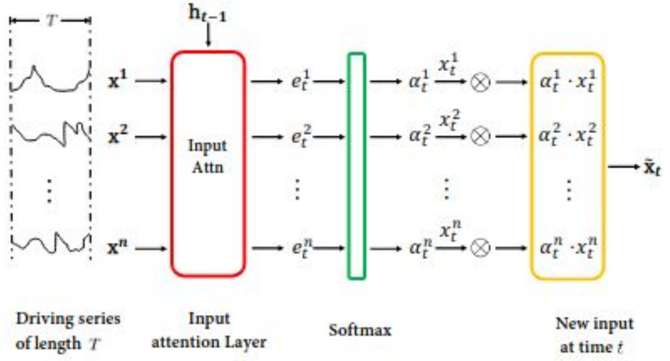
Bhushan D Deo
Saankhya Mondal

Background

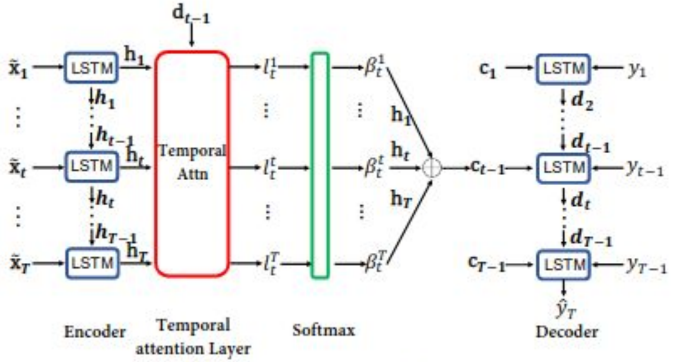
- The aim is to predict future time step time series (stock price) value based on the past history and available exogenous (driving) time series data.
- The simple LSTM based encoder-decoder architecture is insufficient.
- The authors of the Dual Attention RNN paper propose attention mechanisms in both the Encoder and the Decoder models to better capture longterm dependency.
- We have a set of exogenous(driving) time series and the target time series.
- Based on the past history of the target and the exogenous time series we want to do k-step prediction.

Proposed Approach

- The fundamental model is the standard Encoder-Decoder model.
- The baseline paper incorporates an attention mechanism both into the Encoder and the Decoder inspired from behavioural mechanics.It justifies this by achieving excellent results on two datasets at hand.



(a) Input Attention Mechanism



(b) Temporal Attention Mechanism

Technical Details

- The input to the encoder is the exogenous time series $x_t^i \quad i \in \{1, 2, \ldots, N\} \; t \in \{1, 2, \ldots, T\}$
- The encoder input is a weighted stack of the N time series at hand. These weights are computed as a similarity measure between the previous time step encoder hidden state and the input
- The decoder's input context vector is a function of ALL the encoder hidden states.
- This function is an attention mechanism where the weights are computed as a similarity measure between previous time step decoder hidden state and the encoder hidden state.

$$e_t^k = \mathbf{v}_e^\top \tanh(\mathbf{W}_e[\mathbf{h}_{t-1}; \mathbf{s}_{t-1}] + \mathbf{U}_e \mathbf{x}^k)$$

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

$$\tilde{\mathbf{x}}_t = \left(\alpha_t^1 x_t^1, \alpha_t^2 x_t^2, \cdots, \alpha_t^n x_t^n\right)^\top$$

$$l_t^i = \mathbf{v}_d^\top \tanh(\mathbf{W}_d[\mathbf{d}_{t-1}; \mathbf{s}'_{t-1}] + \mathbf{U}_d \mathbf{h}_i), \quad 1 \le i \le T$$

and

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

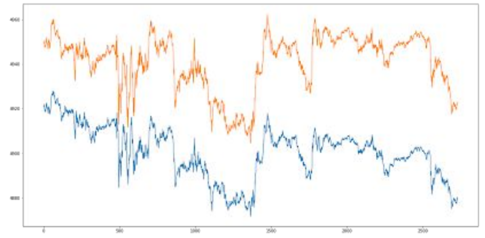
$$\mathbf{c}_t = \sum_{i=1}^T \beta_t^i \mathbf{h}_i.$$

Contributions (Novelty)

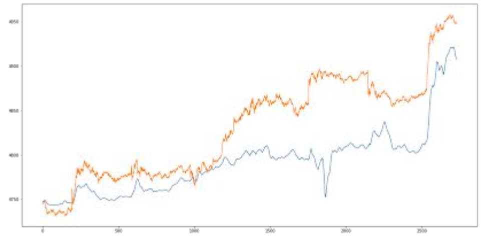
- Firstly we test the efficacy of attention scheme in the Encoder. We simply replace $\tilde{\mathbf{x}}_t = \left(\alpha_t^1 x_t^1, \alpha_t^2 x_t^2, \cdots, \alpha_t^n x_t^n\right)^\top$ with $\tilde{\mathbf{x}}_t = \left(x_t^1, x_t^2, \cdots, x_t^n\right)^\top$, that is simply feeding the input. We observe an inherent bias in the predictions.
- We substitute dot product attention instead of the formulation used by the authors.
- $e_t^k = \mathbf{v}_e^\top \tanh\left(\mathbf{W}_e \left[\mathbf{h}_{t-1}; \mathbf{s}_{t-1}\right] + \mathbf{U}_e \mathbf{x}^k\right) \Longrightarrow e_t^k = \left(\mathbf{W}_e \left[\mathbf{h}_{t-1}; \mathbf{s}_{t-1}\right]\right)^T \mathbf{U}_e \mathbf{x}^k$
- We also tried using all previous hidden states of the decoder for predicting the future target values:
- $\hat{y}_T = \mathbf{v}_y^\top \left(\mathbf{W}_y \left[\mathbf{d}_T; \mathbf{c}_T\right] + \mathbf{b}_w\right) + b_v \Longrightarrow \hat{y}_T = \mathbf{v}_y^\top \left(\mathbf{W}_y \left[\mathbf{d}_T^{net}; \mathbf{c}_T\right] + \mathbf{b}_w\right) + b_v$
- We also used the Transformer model for prediction. Autoregressive mask was removed in the Decoder architecture, encoder-decoder embedding dimensions and num_heads were customized. Also we used a modified version of time2vec encodings for positional embeddings.

Our Dot Product attention and All Previous Hidden states models resulted in better starting points and hence better and faster learnability. Our Transformer model was found to be overfitting on the dataset and needs further analysis.

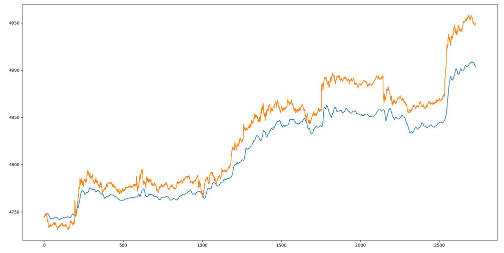
Results & Conclusion



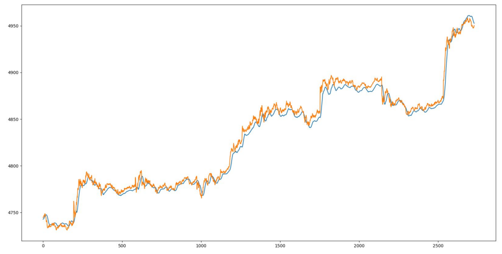
(a)



(b)



(c)



(d)

(a) Bias Gap without attention

(b) Large init. Error Baseline

(c) SDPA Good initial start

(d) PDHS Fast Convergence , good start

Method	MSE/MAE[(1)]	MSE/MAE[(2)]	MSE/MAE[(3)]	MSE/MAE[(TEST)]	Tot. Time Taken
DA-RNN	100/100	0.7/8.6	0.35/5.6	2.15/0.98 (abs)	485 seconds
DA-RNN[(1)]	100/100	4.9/23.2	1.5/10	2.01/0.85 (abs)	606 seconds
DA-RNN[(2)]	100/100	50/80	20/48	2.4/1.06(abs)	400 seconds*
DA-RNN[(3)]	100/100	45.6/62.3	33.81/58.8	1600/40(abs)	450 seconds
Transformer	100/100	13.95/30.8	3.7/16	34.55/5.02(abs)	400 seconds*