COMPARISON STUDY OF MODELS FOR TIME SERIES FORECASTING

Hugo S. Oliveira

University of Porto

ABSTRACT

In this work we explore the uni-variate and multivariate models approach Multivariate Time Series (MTS) regression. Pre-trained models can be potentially used for downstream tasks such as regression and classification, forecasting and missing value imputation. We evaluate our models on several benchmark datasets for uni-variate and multivariate time series regression show that they compete with current state-of-the-art performance, even when the number of training samples is very limited, while at the same time offering computational efficiency. We show that several approximates have advantage and disadvantages, offering substantial performance benefit when compared with similar approaches.

***Index Terms*—** Forecasting, Time Series, Deep Learning

1. INTRODUCTION

MTS is a type of data that is ubiquitous in a wide variety of domains, such as science, medicine, finance, engineering, and industrial applications. With the advent of "Big data" and the abundance of MTS data becomes the new normal. However, the availability of labeled data, in particular, is still limited since extensive data labeling is often prohibitively expensive or even impractical since it may require much time and effort by domain experts to adequately labeled it.

There is a large variety of modeling approaches for univariate and multivariate time series, with Deep Neural Networks (DNN) recently replacing the state of the art methods such as forecasting, regression, and classification [1, 2, 3]. However, unlike the dominance of recent year in Computer Vision or Natural Language Processing (NLP), the reign of deep learning in time series domains is far from established. In fact, non-deep learning methods such as TS-CHIEF [4], HIVE-CCOTE [5], and ROCKET [6] currently hold the state of the art results on time series regressions and classification dataset benchmarks [2], matching or outperforming sophisticated deep learning architectures such as InceptionTime [3], and ResNet [3].

In this work, we investigate the use of simpler regression models and DNN for uni-variate and multivariate time series. In particular we give a theoretical empathise to the uni-variate models such as Auto-Regressive Itegrative Moving Average (ARIMA) [7] and DNN, in particular Long Short Term Memory (LSTM) [8] and Gated Recurrent Unit (GRU) [9] models and models variations with working examples and explore the DNN models for MTS regression/forecasting. For the task in hands we organize our paper as follow:

Section 2 presents the basics theory for regression applied for time series data and describes the basic models and their variations and evaluation techniques. Section 3 defines the methodology and experimental setup for univariate and multivariate time series forecasts. Section 4 presents the conducted experiments and intermediary results. Section 5 discusses the obtained results as an aggregated summary of the findings, and the final section presents conclusions and sketches of future work.

2. BACKGROUND IN TIME SERIES AND RELATED MODELS

Over the years, several models have been proposed for univariate and multivariate times series forecast. Considering this work's broad scope, we focus on the main characteristics and variations commonly encountered when addressing time series forecasting problems.

2.1. Regression of uni-variate time series

ARIMA, also Box-Jenkins model, corresponds to a generalization of the Auto-Regressive Moving Average (ARMA) model by including an integrative component I. These integrated components are useful when data is non-stationary, and the integrated part of ARIMAs contributes to reducing the non-stationary.

The ARIMA applies differentiation on time series one or more times to remove the non-stationary effect. The $ARIMA(p, d, q)$ represents the order of Auto Regression (AR), and Moving Average (MA), and differentiation components.

The major differences between ARMA and ARIMA models are the d component, which updates the series on which the forecasting model is built. The d component aims to detrend the signal to make it stationary, and ARMA model can then be applied to the de-trended data. For a different value of d, the series response changes:

* For $d = 0$: $x_t = x_t$
* For $d = 1$: $x_t = x_t - x_{t-1}$

* For $d = 2$: $x_t = (x_t - X_{t-1}) - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} - x_{t-2}$

As can be seen, the second difference is not two periods ago, rather it is the difference of the first different, that is, $d = 1$. Lets say that $\hat{x}_t$ represents the differences response and so ARIMA forecasting can be written as follows:

$$\hat{x}_t = \phi_1\hat{x}_{t-1} + \phi_2\hat{x}_{t-2} + \cdots + \hat{\phi_{pt-q}} + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \theta_q\epsilon_{t-q} + \epsilon_t$$

Depending on the order of pm d and q, the model behaves differently. For example, $ARIMA(1, 0, 0)$ is a first-order AR model. Similarly, $ARIMA(0, 0, 1)$ is a first-order MA model.

Lets take an example of $ARIMA(0, 1, 0)$ to see the different components of ARIMA modeling. The $ARIMA(0, 1, 0)$ represents a random walk model. The random walk model depends only on the last time instance and can be represented as

$$x_t = x_{t-1} + \epsilon_t$$

The preceding random walk equation can also be represented as lag operators:

$$(1 - L)x_t = \epsilon_t$$

Here, ϵ_t $N(0, \sigma^2)$ is the error component and follows normal distribution. Adding a constant to the preceding random walk model will cause a drift in the model, which is also stochastic as in the following equation:

$$(1 - L)x_t = \alpha + \epsilon_t$$

Here, α is a drift operator that will give a drifting effect to the time series signals.

2.2. LSTM for uni variate time series

Recurrent Neural Nets (RNN) are connoted with difficulty to be trained. Vanilla RNN, the most common, suffer from the problem of vanishing and exploding gradients that lead to inconsistent results during training. As such, the RNN has some difficulties learning long-range dependencies. In the case of time series forecasting, going too many steps back in the past would be problematic. To address this problem, [8] proposed the LSTM to address the problem of vanishing gradients [10].

To effectively train a RNN BackPropagation Thought Time (BPTT) [11], a variant of the back-propagation algorithm [12]is commonly used. The BPTT times is the technique used to commute the weights by summing over the paths connecting the loss node and every time-step. The problem of vanishing gradient in long-range RNN is due to the multiplicative terms in the BPTT gradient.

To solve the vanishing gradient, LSTM introduces an additional computations on each time-step. We can see it as a black box unit that, for time-step h_t, returns the state internal (f_t) and this is forwarded to the next time-step. But internally, these vectors are computed differently. LSTM introduces three new gates: The input (o_t), forget g_t and output c_t gates. On each time-step exist a hidden state h_t and a internal memory $\sigma W^h x_t + U^t s_{t-1}$ and are computed as:

$$f_t = \sigma(W^f x_t + U^f s_{t-1})$$

$$o = \sigma(W^o x_t + U^o s_{t-1})$$

$$g_t = \tanh(W^g x_t + U^g s_{t-1})$$

$$c_t = f_t \odot c_{t-1} + h_t \odot g_t$$

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

$$h_t = f_t$$

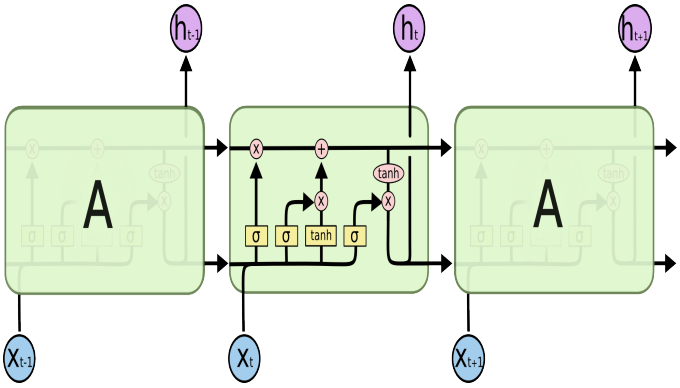

Fig. 1: LSTM main architecture

2.3. GRU for uni variate time series

GRU proposed by [9] are derived from LSTM and are a simpler form that only has two internal states, namely, the update gate z_t and the reset gate r_t., Fig. 2. The computations of the update and reset gates are

$$z_t = \sigma(W^z x_t + U^z s_{t-1})$$

$$r_t = \sigma(W^z x_t + U^z s_{t-1})$$

The state of s_t of the time-step t is computed using the input x_t state s_{t-1} from previous time-step, the update, and the reset gates as:

$$s_t = z_t \odot s_{t-1} + (1 - z_t) \odot \tanh\left((W^h x_t + U^h(r_t \odot s_{t-1}))\right)$$

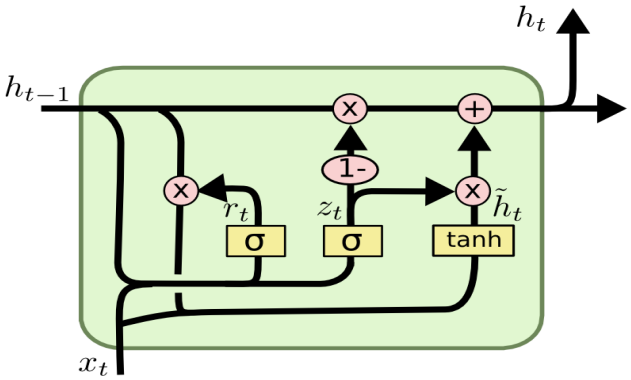

Fig. 2: GRU cell main architecture

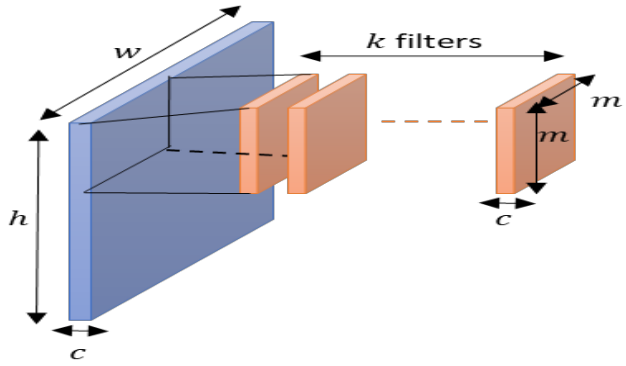

Fig. 3: CNN main architecture

The update obtained by the Sigmoid function determines how much of the previous steps memory is to be retained in the current time-step. The reset gate controls how to combine the last memory with the recent steps input.

Concerning the differences between LSTM an GRU, GRU has fewer parameters/trainable weights when compared with LSTM. One rule is to use GRU when we have less training data since it requires less trainable weights. Contrary, LSTM shows better performance when we have large data-sets such as model languages.

2.4. Convolutional Neural Networks (CNN) of uni-variate time series

The 2D CNN have been widely used to address the problem of image classification, speech recognition, and time series problems with great success, [13]. They were considering the fact that images have a rectangular dimension of w and h width and height, respectively. 2D CNN enable to connect neurons to only patches of the images (activation's regions/maps) instead of all pixels inputs, reducing parameter space.

Considering the Fig. 3, a filter with dimension is applied to a local image patch. The third dimension of the filter is the same as the number of color channels of the image. The weight of each neuron in the filter is multiplied by the corresponding pixel value in the image. The final feature out of this local patch is computed by adding these individual values and the added bias weight and passing the sum through an activation function such as Rectified Linear Unit (ReLu) [14]. ReLu has some characteristics that make it useful for CNN. The first derivative of ReLu is either 0 or 1, making it appropriate for gradient calculations. Concerning the convolution layers, a CNN contains several layers of convolutional layers, generally with many filters to learn different features representations.

To cover all the image, the filter is moved with a horizontal stride and a vertical stride along with the image (Fig. 4). This is the convolutions operation. The features resulting from convolutions over the entire image forms a rectangular feature map.

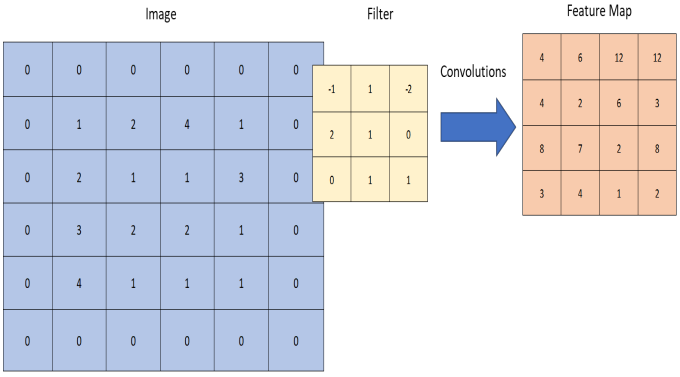

Fig. 4: Convolution filtering

The first convolution corresponds to $1 \times -1 + 2 \times 1 + 2 \times 2 + 1 \times 1 = 6$ Then the ReLu activation's acts as an identity function. This convolution process creates the feature map.

The stacking of several convolution layers enables the generation of better features from the original images. These features are passed to a fully-connected dense layer that generates a SoftMax output over the set of objects classes in the final layers.

In practical terms, we don´t rely only on convolutions to downsample an original image or the intermediate feature maps. If we rely solely on the convolutions layers to perform this task, the number of trained parameters would be huge.

In CNN we make use of another approach to down-sample the input image. The first the convolution layers; they are used primarily for extract features such as corners, edges, shapes, and so on, having an inherent down-sampling also.

Second, to achieve a more effective down-sampling, polling layers filters are used in a local patch of a feature map to compute a single feature. This filter is convoluted over the entire feature map. Polling layers (Fig. 5)do not have trainable weights and are simple arithmetic functions such as maximum or average to generate their output feature maps. Besides, polling layers enable the CNN to be invariant to image rotation, achieving better accuracy.

A common CNN is formed by a set of stacked layers, with

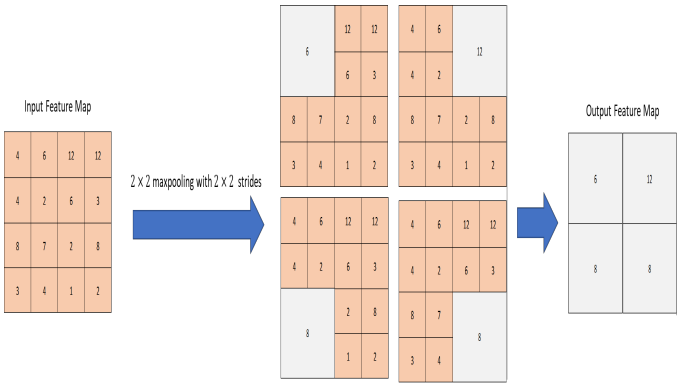

Fig. 5: Pooling space reduction

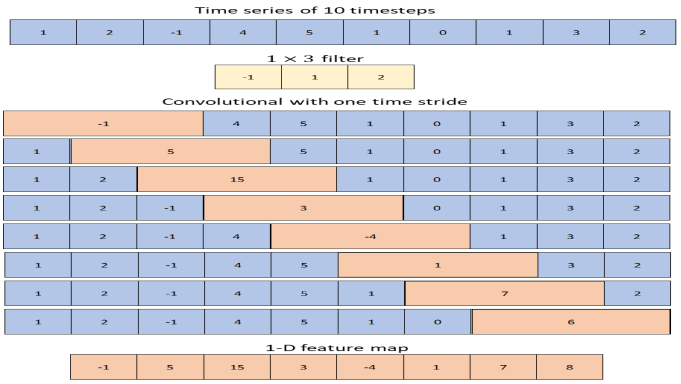

Fig. 7: 1D CNN convolution filters

the image showing two blocks of convolution layers followed by a block of fully-connected dense layers. The last layer is a SoftMax layer that provides the predicted probabilities across the classes, with the higher probability class being the predicted one.

Considering a filter $-1, 1, 2$ convoluted with the times series of 10 steps as represented on the image, the filter moved with a stride of 1. The original time series is not zero-padded, resulting in a feature map with two units shorter than the original time series length. A polling layer can be stacked with the convolution layer to down-sample the feature map even further.

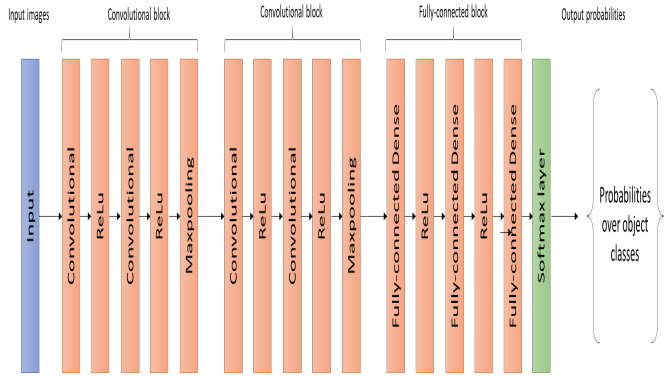

Fig. 6: Common CNN layers architecture

The approach of using a 1×3 convolution filter corresponds to training several local autoregressive models of order, Fig. 7. These local models generate features over the 1D convolution layer, and it creates moving averages over the feature map generated by the preceding convolution layer. Several 1D convolutions and pooling layers, when stacked with each other, give a powerful way of extracting features from the original time series. Thus, using CNN proves to be useful when dealing with complex, nonlinear time series such as audio waves, speech, and so on. CNN have been successfully applied in classifying audio waves also using 2D spectrograms or 1D convolutions in an end to end fashion.

2.5. DNN model variations

Variations of RNN and CNN architectures are very common and useful for improving model generalization by variations to the training process and architecture. As example Bidirectional LSTM, GRU or Vannila RNN are an extension of traditional RNN that can improve model performance on sequence classification problems. In problems where all timesteps of the input sequence are available, Bidirectional RNNs train two instead of one RNNs on the input sequence. The first on the input sequence as is and the second on a reversed copy of the input sequence Fig. 8. This bidirectional can provide additional context to the network and faster and even fuller learning on the problem.

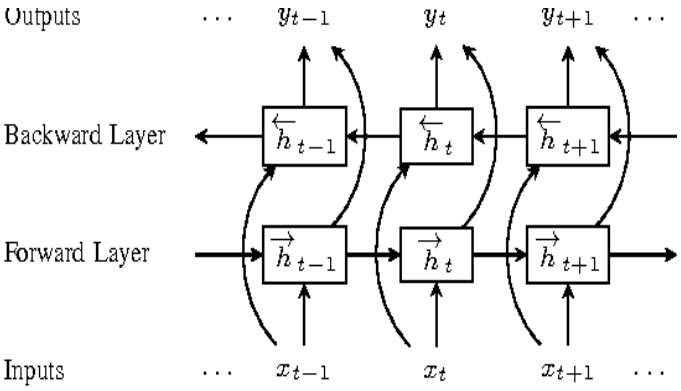

Fig. 8: Bidirectional LSTM example

Other DNN model variations such as Dual-Path RNN, DPRNN, a simple yet effective method for organizing RNN layers in a deep structure to model too long sequences. DPRNN splits the long sequential input into smaller chunks and applies intra- and inter-chunk operations iterative, Fig.10 where the input length can be made proportional to the square root of the original sequence length in each process.

Other variations are the employ of Sequence 2 Sequence (Seq2Seq) approaches, that consist of an encoder-decoder based machine model that maps an input of sequence to an output of sequence with a tag and attention value. The idea is

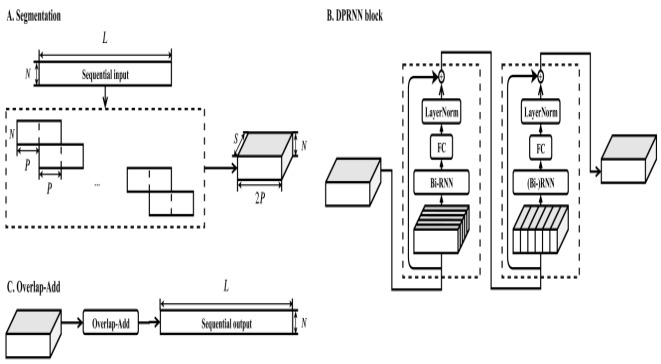

Fig. 9: Dual-Path LSTM example

to use two RNN that will work together with a unique token, and predict the next sequence from the previous sequence, Fig. **??**

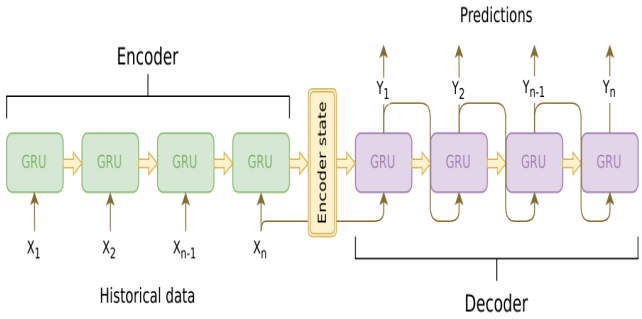

Fig. 10: Sequence 2 Sequence GRU example

Sequence-to-sequence (Seq2seq) models have played an essential role in the recent success of various natural language processing methods, such as machine translation, text summarization, and speech recognition. However, current Seq2Seq models have trouble preserving global latent information from a a long sequence of words. Variational Autoencoder (VAE), Fig. 11 alleviates this problem by learning a continuous semantic space of the input. The encoder path is similar. It is merely a network that takes in an input and produces a much smaller representation (the encoding), which contains enough information for the next part of the network to process it into the desired output format.

Typically, the encoder is trained together with the other parts of the network, optimized via back-propagation, to produce encodings specifically useful for the task at hand. This fact enables to capture of more meaningful full latent feature representations in a smaller space.

In a resume, the general idea of autoencoders is pretty simple. It consists of setting an encoder and a decoder as neural networks and learning the best encoding-decoding scheme using an iterative optimization process. Thus, intuitively, the overall autoencoder architecture (encoder+decoder) creates a bottleneck for data, ensuring that only the central structured part of the information can go through and be reconstructed.

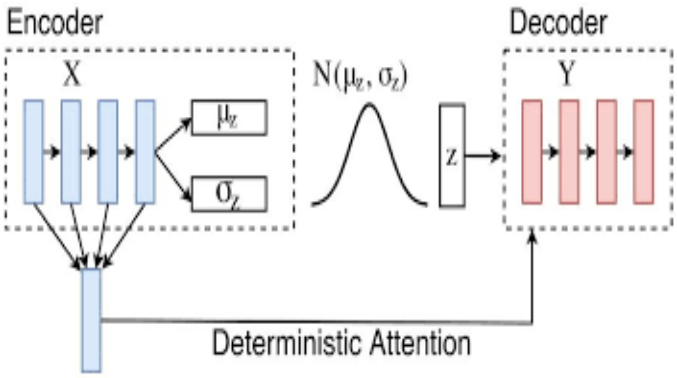

Fig. 11: Sequence-to-sequence (Seq2seq) with VAE

Attention layers are common in combination with sequence-to-sequence (seq2seq) models. Attention was initially introduced to address the central issue surrounding seq2seq models, also commonly defined as the Encoder-Decoder model, and to great success.

The standard seq2seq model is unable to accurately process longer input sequences since only the last hidden state of the encoder RNN is used as the context vector for decoding. Attention Mechanism enables to address this limitations by retaining and utilize the hidden states of from the input sequence for the decoding process. This is achived by the creation of a simple unique mapping for each time step of the decoder output towards the encoder hidden states. Each output that the decoder mak, it can access the entire input sequence and select an specific element of the sequence to produce the output. Two genres of attention mechanisms are commonly employed, the Additive Attention by Dzmitry Bahdanau [15], Fig. 12.

It aims to improve the sequence-to-sequence model in machine translation by aligning the decoder with the relevant input sentences and implementing Attention. The entire step-by-step process of applying Attention corresponds to:

1. Producing the Encoder Hidden States - Encoder produces hidden states of each element in the input sequence

2. Alignment Scores calculation from the the previous decoder hidden state and each of the encoder's hidden states.

3. Softmaxing Alignment Scores - the alignment scores for each encoder hidden state are used to represent a single vector and subsequently softmax

4. Calculating the Context Vector - the encoder hidden states and their respective alignment scores are multiplied to form the context vector

5. Decoding the Output - the context vector iscombined with the previous decoder output and used to supply the Decoder RNN for that time step along with the decoder earlier hidden state to produce a new output
6. The steps 2-5 are then repeated for each time step of the decoder until a token is generated or output is past the specified maximum length

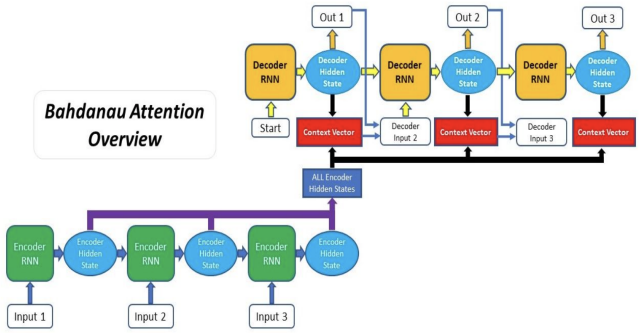

Fig. 12: Additive Attention scheme

The second genre of Attention was proposed by Thang Luong [16]. It commnluy known as Multiplicative Attention and has its bases on the Attention mechanism initialy proposed by Bahdanau. The main differences between the two attention methods are:

- The way that the alignment score is calculated
- The position at which the Attention mechanism is being introduced in the decoder

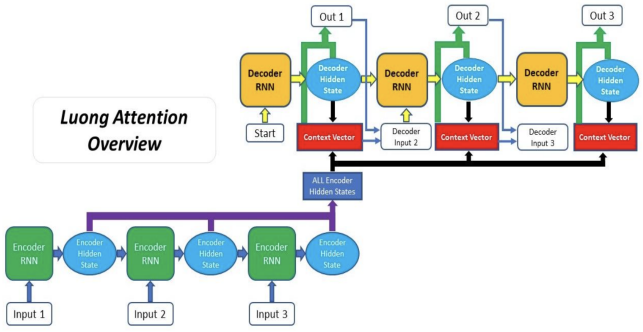

Fig. 13: Multiplicative Attention scheme

Three types of alignment scoring functions are proposed in Luong's paper along with Bahdanau's one type. Also, the Attention Decoder's general structure is different for Luong Attention, as the context vector is only utilized after the RNN produced the output for that time step. In greater detail the differences through the Luong Attention process are:

1. Producing the Encoder Hidden States - Encoder produces hidden states of each element in the input sequence
2. Decoder RNN - the previous decoder hidden state and output are passed through the Decoder RNN that generates a new hidden state for the particular time step
3. Calculating Alignment Scores - using the new decoder hidden state and the encoder hidden states, alignment scores are calculated
4. Softmaxing Alignment Scores - the alignment scores for each encoder hidden state are combined together and can be represented by a single vector and subsequently softmax
5. Calculating the Context Vector - the encoder hidden states and their respective alignment scores are multiplied to form the context vector
6. Producing the Final Output - the produced context vector is then combined with decoder hidden state previously generated and passed through a fully connected layer to produce a new output

The steps 2-6 are repeated in each time step until an token is produced by the decoder or output is past the specified maximum.

A final model modification directed for CNN is the Dilated Convolutional Neural Networks (ID-CNN), which have better capacity than traditional CNNs for large context and structured prediction. Unlike RNN whose sequential processing on sentences of length N requires $O(N)$ time even in the face of parallelism, ID-CNN's permit fixed-depth convolutions to run in parallel across entire inputs.

Dilated Convolutions are a type of convolution that "inflate" the kernel by inserting holes between the kernel elements, Fig.14. An additional parameter (dilation rate) indicates how much the kernel is widened. There are usually spaces inserted between kernel elements.

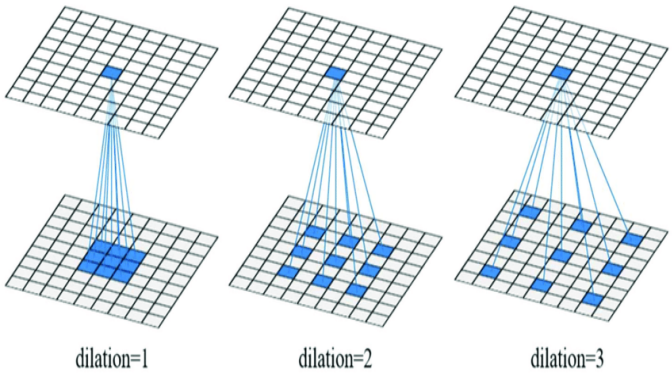

Fig. 14: Dilated 2D CNN

2.6. Time Series evaluation criteria

Akaike Information Criterion (AIC), Bayesian information criterion (BIC) and Structural Risk Minimization (SRM) where methods proposed for model selection.

AIC enables to evaluate how well models fits the data set. The AIC score favors models that achieve a high goodness-of-fit score and penalizes them if they become overly complicated. Models with lower AIC score are expected to obtain a better balance between the ability to fit the data set and simulatiosly avoid over-fitting the data set.

The formula for the AIC score:

$$AIC = 2k - 2ln(\hat{L}) \tag{1}$$

where $\hat{L}$ corresponds to the maximized value of the likelihood function of the model, and k is the number of free parameters to be estimated.

BIC, on the other hand, is a criterion for model selection among a finite set of models. It is based on the likelihood function, and it is closely related to AIC. When fitting models, the likelihood can be increased by adding parameters, but it may result in overfitting. The BIC resolves the problem of overfitting caused by the addition of extra parameters. It introduces a penalty term for the number of parameters in the model. The penalty term is larger in BIC than in AIC.

BIC is also widely used for model identification in time series. It can, however, be applied quite widely to any set of maximum likelihood-based models. Mathematically BIC can be defined as:

$$BIC = ln(N)k - 2ln(\hat{L}) \tag{2}$$

where $\hat{L}$ and k also corresponds to the maximized value of the likelihood function of the model and k is the number of free parameters to be estimated, respectively. The N corresponds to the number of data points.

Structural Risk Minimization (SRM) is an inductive principle of use in machine learning. Commonly in machine learning, a generalized model must be selected from a finite data set, with the consequent problem of overfitting – the model becoming too strongly tailored to the particularities of the training set and generalizing poorly to new data. The SRM principle addresses this problem by balancing the model's complexity against its success at fitting the training data.

$$R_{sm} = (F)\frac{1}{N}\sum_{i=1}^{N} L(h(y_i, f(x_i)) + \lambda J(F) \tag{3}$$

which is called SRM. $J(f)$ is the complexity of the model, usually can be the bound of vector space. h correspond to the model and $\lambda \geq 0$ is the coefficient choosing the strength of the penalty term.

2.7. Regression Metrics

Standard metrics for regression include Mean Square Error (MSE) and Mean Absolute Error (MAE) as defined in Equations 4 and 5, where y_i is a true value and $\hat{y}_i$ its prediction.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{4}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{5}$$

Relative error metrics are unit less which means that their scores can be compared across different domains and are calculated by comparing the scores of the model under evaluation against the scores of some baseline model. The relative score is expected to be a value between $0, 1]$, with values nearer (or even above) 1 representing performances as bad as the baseline model, which is usually chosen as something too naive. The most common baseline model is the constant model consisting of predicting for all test cases the average target variable value calculated in the training data. Normalized Mean Squared Error (NMSE) (Equation 6) and Normalized Mean Absolute Error (NMAE) (Equation 7)

$$NMSE = \frac{\sum_{i=1}^{n}(\hat{y} - y_i)^2}{\sum_{i=1}^{n}(\bar{y} - y_i)^2}. \tag{6}$$

$$NMAE = \frac{\sum_{i=1}^{n}|\hat{y} - y_i|^2}{\sum_{i=1}^{n}|\bar{y} - y_i|^2}. \tag{7}$$

Mean Average Percentage Error(MAPE) (Equation 8)

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{y}_i - y_i}{y_i}. \tag{8}$$

The correlation between the predictions and the true values $(\rho_{\hat{y},y})$ is given by

$$\rho_{\hat{y},y} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{9}$$

3. METHODOLOGY AND CONDUCTED EXPERIMENTS

In this work, we explore several models for time series forecasting, that range from mathematical approach, up to DNN models. The approach starts by individual models and inclusion of variants for performance evaluation.

We gonna explore the models from the same datasets, and its variations, and briefly discuss each of the models in term of performance on the respective datasets. Finally we evaluate an model in a different dataset to access the generalization capabilities for multi variate time series.

For the task in hands, we consider the use of uni-variate and multivariate times series datasets to access the performance of the models and establish a relation of between them. For univatiate analysis, we explore the 1, 2 and 3 datasets.

[1]

[2]

[3]

for training purposes the train dataset derived from starting timestamp until last 30 days, and the test dataset contains the last 30 days. for forescating and repeated 10 times. Numerical features are normalized for training and the inverse transformation is performed to obtain the original numeric scale.

3.1. LSTM for uni-variate time series

RNN has a fancier recurrence relation than the vanilla RNN. LSTM has two states, one is being the usual hidden state $h(t)$ simillar as in the vanilla RNN and another one is called the cell state $c(t)$. Cell state corresponds to a an internal vector that is not exposed to the outside world. LSTM have been used in a wide variety of sequence problems, such as Natural Language Processing, speech recognition. We evaluate the model performance using different LSTM models with the 1 dataset that contains MTS. Same datasets and scale transformation are considered for the experiments.

3.1.1. LSTM

Using the same dataset, we evaluate the model performance using RNN models with the 1 Data Data Set that contains MTS.

On Fig. 15 shows a box-plot that exhibits the present of spikes in the data (Fig. 16, meaning that sudden changes in the mean value are present.

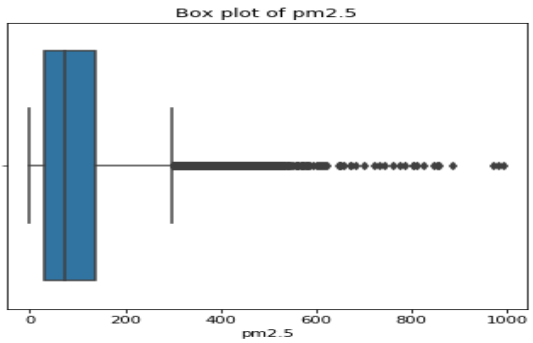

Fig. 15: Box plot of data

[1]Beijing PM2.5 Data Data Set
[2]IBM common stock closing prices Data Data Set
[3]Google Stock Price

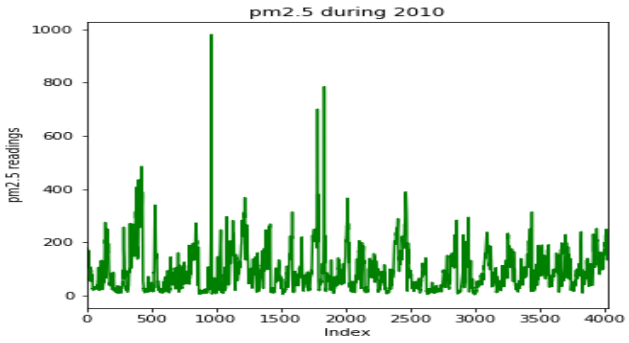

Fig. 16: Time Series during 2010

Concerning the modeling, a good practice is to normalize the values between $[0-1]$ interval, Fig 17 enabling to have lower gradients and reduce the training time of the LSTM model.

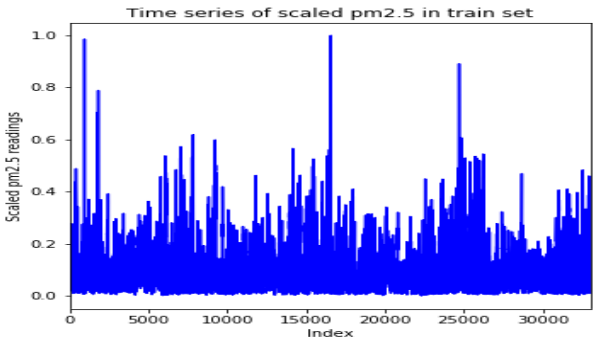

Fig. 17: Time series of scaled in validation set

The proposed LSTM architecture is defined with two LSTM layers and a Dropout layer [17] and dense layer according to Fig. 18. The Dropout layer acts as a regularization technique to avoid over-fitting of the network, by randomly skipping weight connections during training to improve model generalization.

For optimization we use the Adam [18] using a learning rate $\eta = 0.001$ and decay factor of $1e^{-5}$. The batch size was set to 16, and model was trained during 20 epochs using the MAE loss. Fig. 19 shows the loss evolution of the training and validation set.

The final predicted values vs the Ground Truth (GT) are shown on the Fig. 20, with the model exhibiting a MAE of 11.583.

3.1.2. LSTM Bidirectional

LSTM in its core, preserves information from inputs that has already passed through it using the hidden state. While unidirectional LSTM only preserves information of the past because the only inputs it has seen are from the past, bidirec-

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 7, 1)              0
_________________________________________________________________
lstm_1 (LSTM)                (None, 7, 64)             16896
_________________________________________________________________
lstm_2 (LSTM)                (None, 32)                12416
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 29,345
Trainable params: 29,345
Non-trainable params: 0
_________________________________________________________________
```

Fig. 18: LSTM model architecture

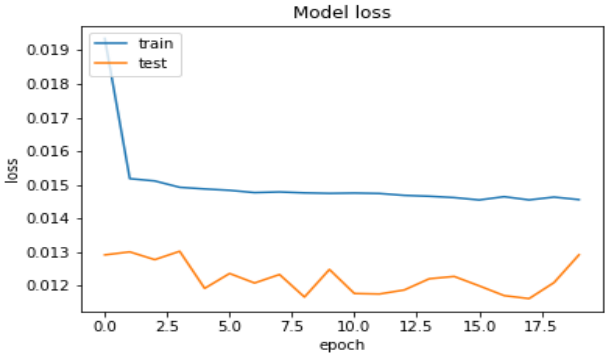

Fig. 19: LSTM model loss

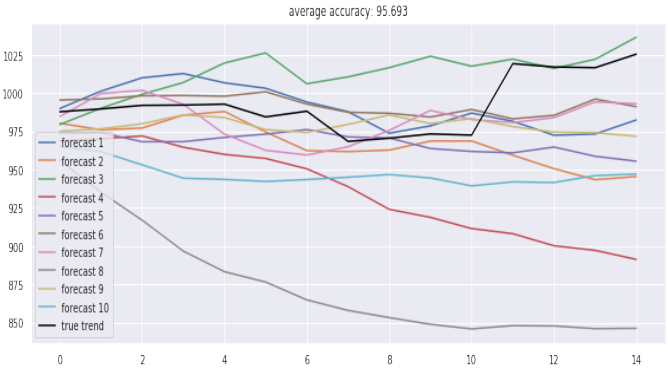

Fig. 20: LSTM Actual vs Predicted

tional will run the inputs in two ways, one from past to future and one from future to past. The main difference of this approach from the unidirectional version is that in the LSTM that runs backwards preserves information from the future and by the use of two hidden states combined is able in any point in time to preserve information from both past and future. On fig. **??** awe show the performance bidirectional LSTM in the dataset. 10 random experiments were performed and the aggregated mean accuracy was calculated.

We see a little downgrade in the performance when compared with LSTM, however for longer sequences is possible that the model is able to generalize better.

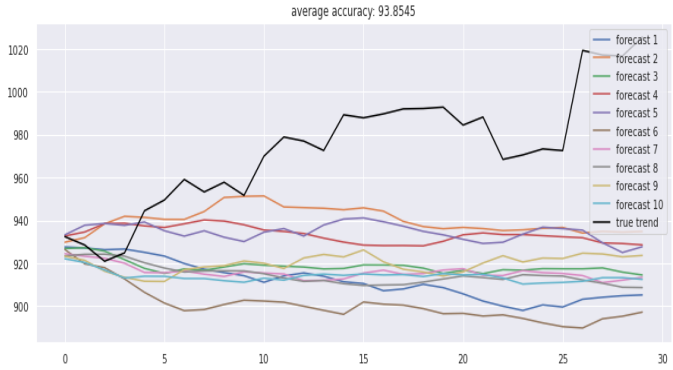

Fig. 21: LSTM Bidirectional Training Actual vs Predicted

3.1.3. LSTM Sequence 2 Sequence

A sequence to sequence model maps a fixed-length input with a fixed-length output where the length of the input and output may differ. Its commonly composed by a encoder Encoder formed by a stack of several recurrent units LSTM cells where each accepts a single element of the input sequence, collects information for that element and propagates it forward. Then is followed by a decoder, also a stack of several recurrent units where each predicts an output y_t at a time step t. Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state. This make suitable for time series forecasting of any length, enabling to obtain competitive results. Fig. 22 shows the obtained results on the test set of the 10 random experiments.

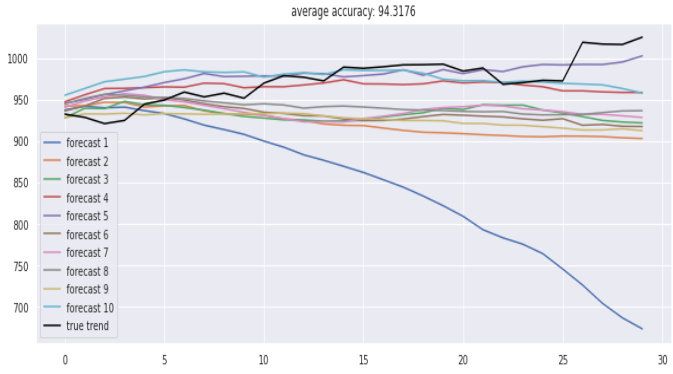

Fig. 22: LSTM Sequence 2 Sequence Actual vs Predicted

Is possible to observe that the results are compare with simple LSTM models, meaning that the extra complexity of the encoder-decoder configuration does not add significant improvements to model performance.

3.1.4. LSTM Sequence 2 Sequence Bidirectional

We reuse the last model, but now but making a bidirectional. Bidirectionally enables to capture past and future information for forecasting. Fig. 23 shows the obtained results on the test set of the 10 random experiments.

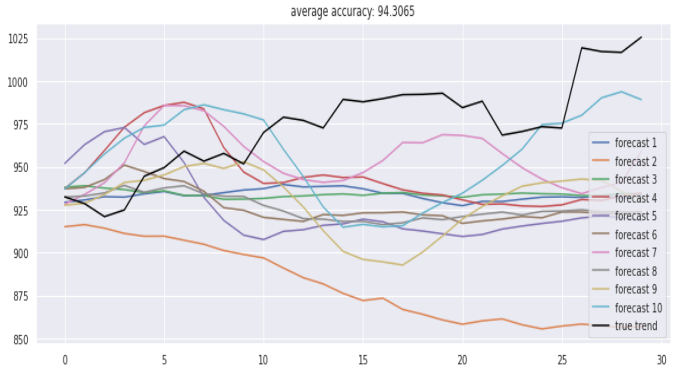

Fig. 23: LSTM Sequence 2 Sequence Bidirectional Actual vs Predicted

Is possible to observe that the results are compare with simple LSTM models, meaning that the extra complexity of the encoder-decoder configuration does not add significant improvements to model performance.

3.1.5. LSTM Sequence 2 Sequence VAE

We reuse the last model, but now but introducing a bottleneck. The bottleneck is an inheritance to a simple encoder-decoder models. However, it encompasses several problems, such as overfitting. A variational autoencoder can be defined as an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable the generative process. Just as a standard autoencoder, a variational autoencoder is an architecture composed of both an encoder and a decoder, and that is trained to minimize the reconstruction error between the encoded-decoded data and the initial data. However, in order to introduce some regularisation of the latent space, a slight modification of the encoding-decoding process is made by instead of encoding an input as a single point, it encodes as a distribution over the latent space. Fig. 24 shows the obtained results on the test set of the 10 random experiments.

Is possible to observe that the results are compare with simple LSTM models, meaning that the extra complexity of the encoder-decoder configuration does not add significant improvements to model performance. However a slight improvement was observed when compared with traditional encoder-Decoder models, suggesting that the bottleneck in fact avoid some degree of overfitting.

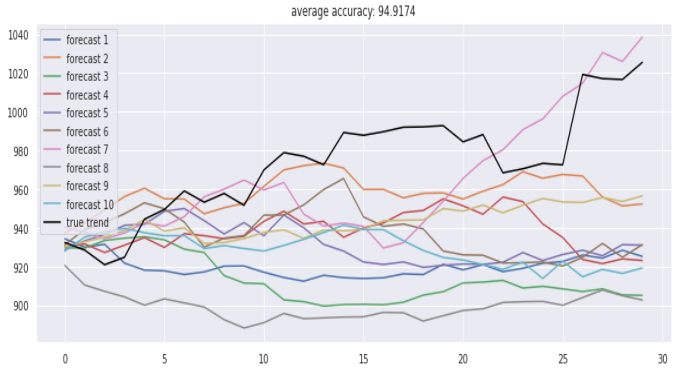

Fig. 24: LSTM Sequence 2 Sequence Variational Auto Encoder (VAE) Actual vs Predicted

3.2. GRU for uni-variate time series

Using the same dataset, we evaluate the model performance using RNN models with the 1. Same consideration were made and dataset transformations such as feature normalization for training and inverse transformation.

3.2.1. GRU

The proposed GRU architecture is defined with two GRU layers and a Dropout layer [17] and dense layer according to Fig. 25. The Dropout layer acts as a regularization technique to avoid over-fitting of the network, by randomly skipping weight connections during training to improve model generalization.

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 7, 1)              0
_________________________________________________________________
gru_1 (GRU)                  (None, 7, 64)             12672
_________________________________________________________________
gru_2 (GRU)                  (None, 32)                9312
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 22,017
Trainable params: 22,017
Non-trainable params: 0
_________________________________________________________________
```

Fig. 25: GRU model architecture

For optimization we use the Adam [18] using a learning rate $\eta = 0.001$ and decay factor of $1e^{-5}$. The batch size was set to 16, and model was trained during 20 epochs using the MAE loss. Fig. 19 shows the loss evolution of the training and validation set.

The final predicted values vs the GT are shown on the Fig. 27, with the model exhibiting a MAE of 11.6807.

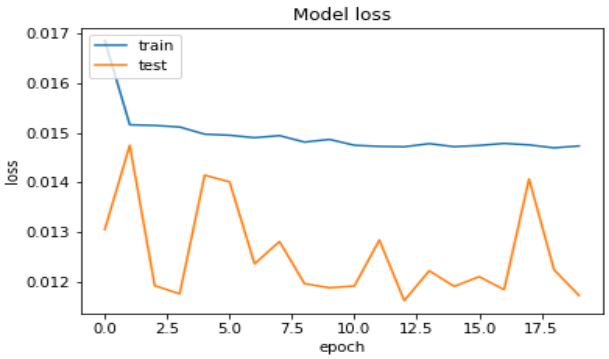

Fig. 26: LSTM model loss

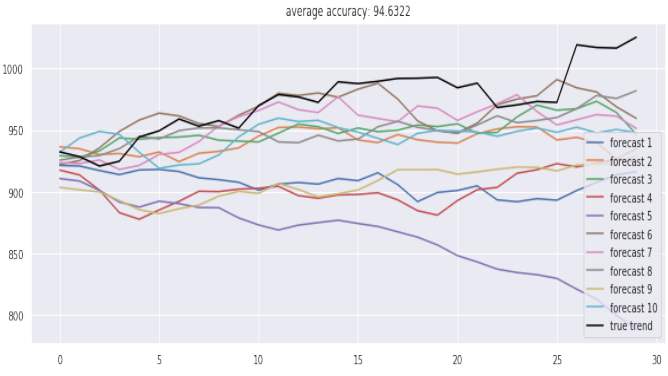

Fig. 27: GRU Actual vs Predicted

3.2.2. GRU Bidirectional

Similar to LSTM in its core, the GRU bidirectional also share common characteristics. The main difference is that GRU has fewer control gates when compared with LSTM. Fig. 28 shows the obtained results on the test set of the 10 random experiments.

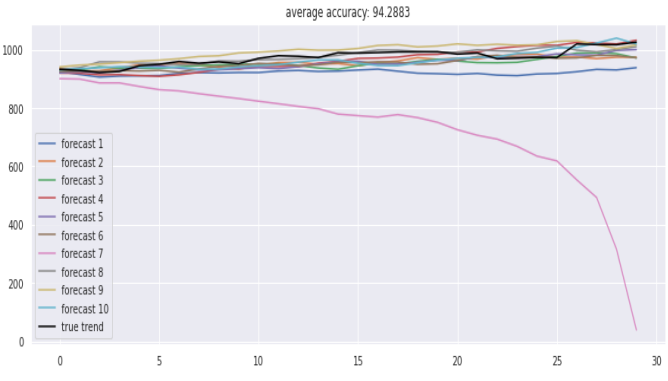

Fig. 28: GRU Bidirectional Training Actual vs Predicted

Is possible to observe that the results are lower when compare with simple LSTM models, meaning that the extra complexity of bidirectional in GRU configuration does not add significant improvements to model performance. In fact some degree of overfitting was observed.

3.2.3. GRU Sequence 2 Sequence

Similar conducted experiment of Encoder-Decoder, but now using GRU models. Fig. 29 shows the obtained results on the test set of the 10 random experiments.

ng

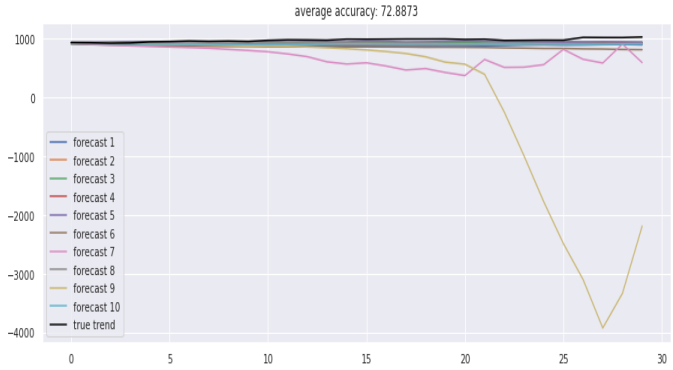

Fig. 29: GRU Sequence 2 Sequence Actual vs Predicted

Is possible to observe that the results are lower when compare with simple LSTM models, meaning that the extra complexity of Encoder-Encoder in GRU configuration does not add significant improvements to model performance. In fact large degree of overfitting is observed.

3.2.4. GRU Sequence 2 Sequence VAE

Ion this experiment, we explore the performance of Encoder-Decoder combined with a VAE to encapsulate the relevant information. Fig. 30 shows the obtained results on the test set of the 10 random experiments.

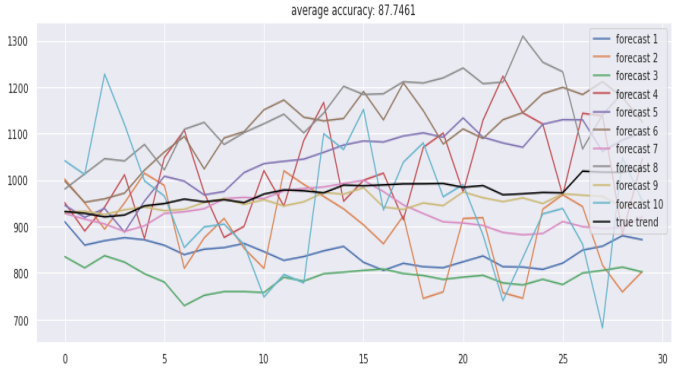

Fig. 30: GRU Sequence 2 Sequence VAE Actual vs Predicted

Is possible to observe that the results are even lower than expected, with simple LSTM models, meaning that the ex-

tra complexity of bidirectional in GRU configuration does not add significant improvements to model performance. In fact large degree of overfitting is observed.

3.3. CNN for Univariate Time Series

Using the same dataset, we evaluate the model performance using RNN models with the 1 Data Data Set that contains MTS. Same consideration were made and dataset transformations.

3.3.1. CNN

The proposed CNN architecture is defined with two CNN layers and followed by a Average Polling layer, an Dropout layer [17] and dense layer for final regression predicted value , according to Fig. 31. The Dropout layer acts as a regularization technique to avoid over-fitting of the network, by randomly skipping weight connections during training to improve model generalization. In addiction, the initial signal is zero padding to enable overcome the convolution problems with left and side limits of the signal, maintaining the original length.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 7, 1)              0
_________________________________________________________________
zero_padding1d_1 (ZeroPaddin (None, 9, 1)              0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 7, 64)             256
_________________________________________________________________
average_pooling1d_1 (Average (None, 5, 64)             0
_________________________________________________________________
flatten_1 (Flatten)          (None, 320)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 320)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 321
=================================================================
Total params: 577
Trainable params: 577
Non-trainable params: 0
_________________________________________________________________
```

Fig. 31: CNN model architecture

For optimization we use the Adam [18] using a learning rate $\eta = 0.001$ and decay factor of $1e^{-5}$. The batch size was set to 16, and model was trained during 20 epochs using the MAE loss. Fig. 19 shows the loss evolution of the training and validation set.

The final predicted values vs the GT are shown on the Fig. 33, with the model exhibiting a MAE of 13.0001.

3.3.2. CNN Sequence 2 Sequence

We evaluate the performing of CNN in a Encoder-Decoder configuration. Fig. 34 shows the obtained results on the test set of the 10 random experiments.

Is possible to observe that the results are even lower than expected than simple LSTM models, meaning that the extra complexity of Encoder-Decodder in CNN configuration does not add significant improvements to model performance. In fact large degree of overfitting is observed. Meaning that most of information was already captures in the convolution layers, and the encoder was not able to explore correctly this information.

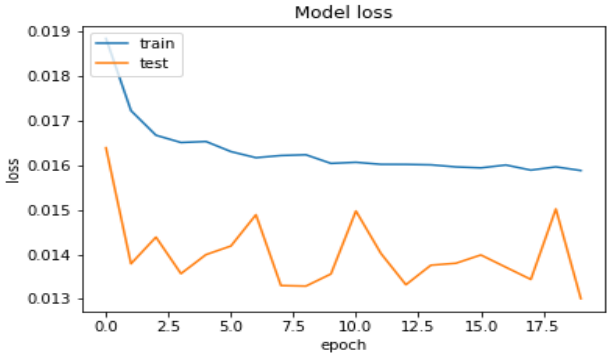

Fig. 32: CNN model loss

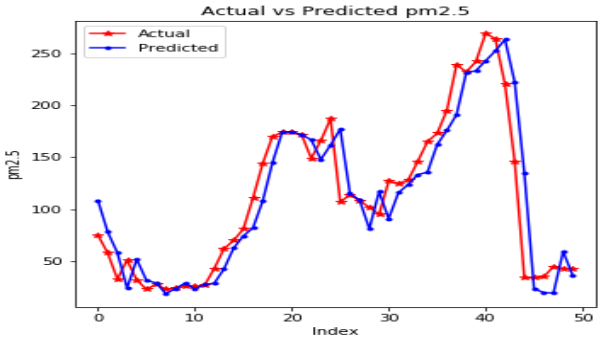

Fig. 33: 1D CNN Actual vs Predicted

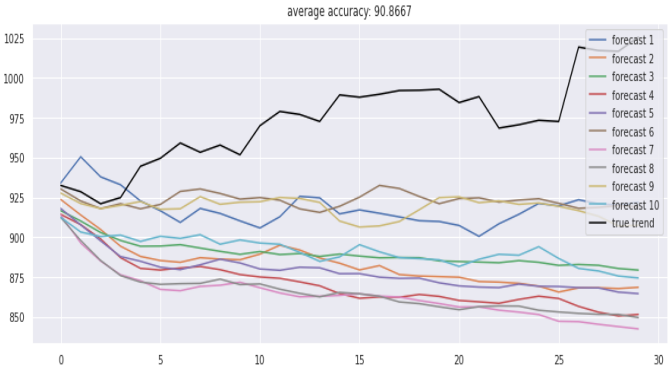

Fig. 34: CNN Sequence 2 Sequence Actual vs Predicted

3.3.3. CNN Dilated Sequence 2 Sequence

We evaluate the performing of CNN in a Encoder-Decoder configuration using dilated convolutions. Fig. 35 shows the obtained results on the test set of the 10 random experiments.

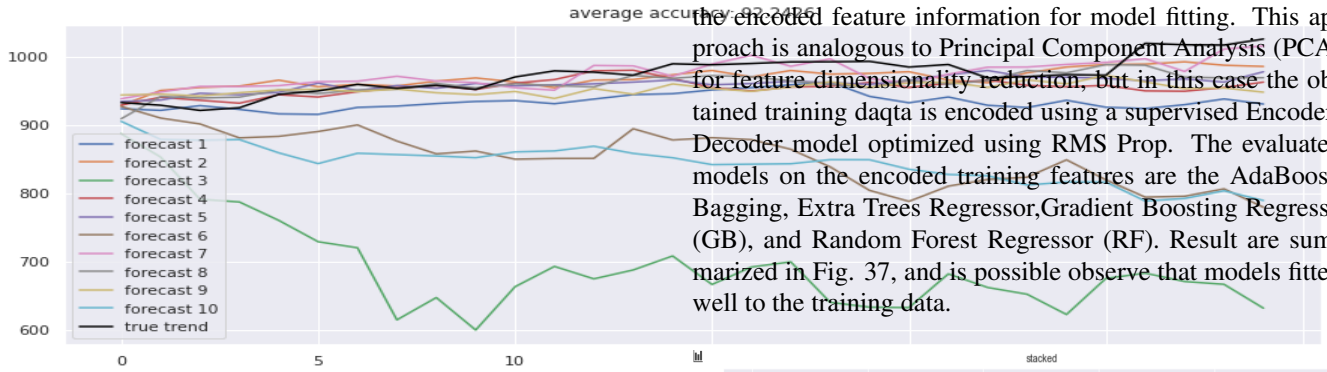

Fig. 35: CNN Sequence 2 Sequence Actual vs Predicted

Results are more robust, meaning that the increase of space time-space resolution in CNN configuration adds significant improvements to model performance. Most of information was captured in the convolution layers, and was able to be used correctly for forecasting.

3.4. Deep Feed-forward Auto-Encoder Neural Network to reduce dimension + Deep Recurrent Neural Network + ARIMA + Extreme Boosting Gradient Regressor (XGB)

On this experiment, we want to access the performance of several models to determine the complexity vs performance trade-off. For the experiments we employ 4 different models, ARIMA, RNN and stacked RNN and XGB. The result on Fig. 36 show that ARIMA suppressed the more advanced models, meaning that for more simpler problems, ARIMA presents good results without major complexity.

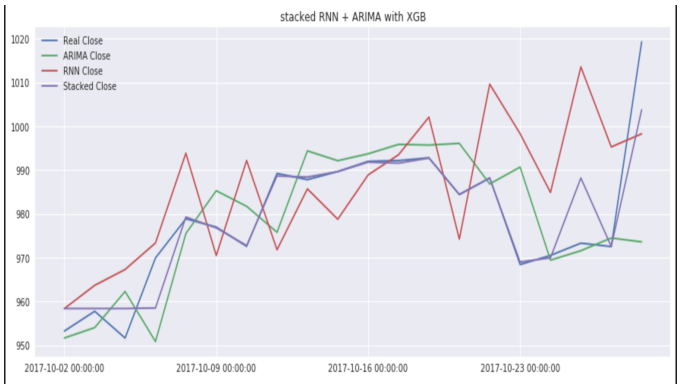

Fig. 36: Side to side forecast comparisons of the several models

3.5. Stacked Ensemble with XGB

On this experiment, we explore the use of encoder feature combined with several regressors models. The encoder is responsible to encapsulate the relevant information of the training set on a latent reduced space, enabling for models to reuse the encoded feature information for model fitting. This approach is analogous to Principal Component Analysis (PCA) for feature dimensionality reduction, but in this case the obtained training daqta is encoded using a supervised Encoder-Decoder model optimized using RMS Prop. The evaluated models on the encoded training features are the AdaBoost, Bagging, Extra Trees Regressor,Gradient Boosting Regresso (GB), and Random Forest Regressor (RF). Result are summarized in Fig. 37, and is possible observe that models fitted well to the training data.

Fig. 37: Side to side forecast comparisons of the several models

When evaluating the test data, the models where able to forecast correctly the 30 lags ahead and then converged to the mean, Fig. 38.

Fig. 38: Side to side forecast comparisons of the several models

3.6. LSTM for Multi Variate Time Series

As our final goal, we evaluate the performance of MTS multi variate time series. As summary of our experiments, we now present a experiments in a different dataset, containing multiple variable as predictors and a continuous target variable, the stock value. An exploratory analysis is performed to conduct

to the final task. First a simple Peak analysis was performed, Fig. 39, to identify abnormal behaviours.

Fig. 39: MTS peaks and trends

We observe to major frontiers, an lower values at the beginning and a higher value towards the end. This is consistent with a constant increasing value company and no seasonality at a first glance is observed. Considering the features values, we asses theirs rate of change along the period of time in consideration, Fig. 40.

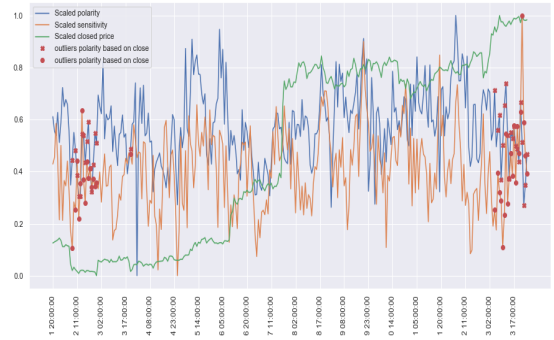

Fig. 40: MTS feature scaled plots

Is not possible to correlate immediately an feature with the response, since they don't exhibit similar trend distribution. To support out initial guess, we plot the person correlation, Fig. 41.

We see that only the open, high and low exhibit a correlation with the close value. This is correlation is due to the fact that the opening, high and low summarizes the close value in an short range of time, commonly one day period. when analysing over a 42 hour period, in intervals of 7 hours, Fig. 42

we observe that during a period of approximately one week, of trading, the close price are very correlated with each other, meaning that during a week period the exist similar dynamic in action. To evaluate our model, we start by performing a simple side-by-side comparisons with Moving

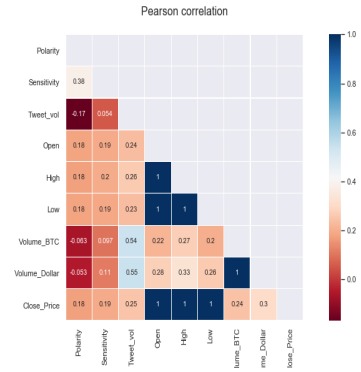

Fig. 41: MTS Feature Pearson Correlation

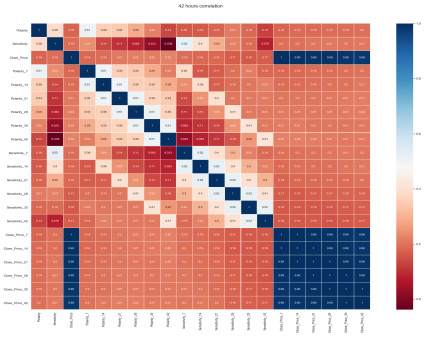

Fig. 42: MTS 42 hours Feature Pearson Correlation

Average Models with different memory lengths, Fig. 43

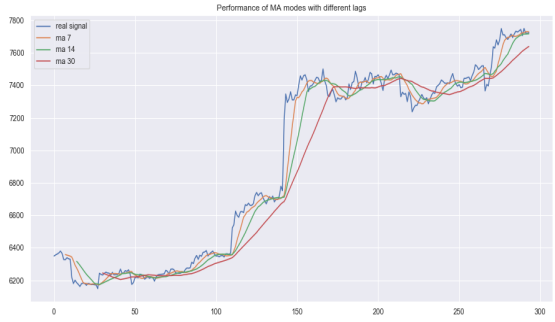

Fig. 43: MTS Simple Moving Average evaluation with different lags

We observe that longer memory models, don't follow well the trend, and shorter memory models, while close to the trend, they are very sensitive to the ongoing trend dynamics. For forecast we then employ a LSTM model that makes use of all features at once, and forecast the next 30 lags, Fig 44

We observe that the models was able to capture well the trend dynamics and forecast correctly this time series without major error.

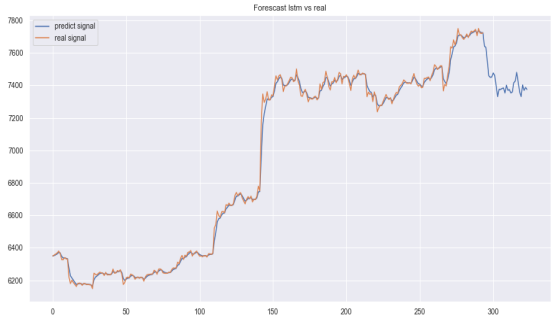

Fig. 44: MTS Forecasts vs GT

4. OVERALL MODEL EVALUATION

Considering the unvaried time series experiments, we observe that CNN, GRU and LSTM presented similar performance values, however the CNN with dilated convolutions showed an competitive performance presented far less parameters than the other models, while achieving similar performance.

Concerning the multivariate analysis, LSTM showed good performances when compared with simpler models such as Moving Average. However is some cases, ARIMA showed some good performances namely when the time series is not too long. Contrary, RNN exhibit a better performance when the time series is longer, meaning that LSTM and GRU cells are able to capture long term memory dependencies of the time series.

5. CONCLUSIONS

With the recent advancement on developing sophisticated based techniques and in particular DNN algorithms. The major question is then how accurate and powerful these newly introduced approaches are when compared with common methods. In this work we compare several ARIMA models with and LSTM, as representative techniques when forecasting time series data.

Several techniques were implemented and applied on a set of financial, ans climate data and the results indicated that LSTM was superior to ARIMA. More specifically, the LSTM based algorithm improved the prediction by 85% on average compared to ARIMA. Furthermore, the number of epochs does not improve when is changed.

The work described in this paper advocates the benefits of applying DNN algorithms and techniques to the economics, climate and financial data.

There are several other prediction problems in finance and economics that can be formulated using DNN. We plan to investigate the improvement achieved through DNN by applying these techniques to some other problems and datasets with various numbers of features.

Overall, the use of DNN for time series forecasting showed promising results, and alleviate the need to verify an wide number of pre-conditions such as stationary, and non trending, enabling models to be applied to wide genre of time series.

6. REFERENCES

[1] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau, "Gru-ode-bayes: Continuous modeling of sporadically-observed time series," in *Advances in Neural Information Processing Systems*, 2019, pp. 7379–7390.

[2] Chang Wei Tan, Christoph Bergmeir, Francois Petitjean, and Geoffrey I Webb, "Monash university, uea, ucr time series regression archive," *arXiv preprint arXiv:2006.10996*, 2020.

[3] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean, "Inceptiontime: Finding alexnet for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936–1962, Sep 2020.

[4] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I Webb, "Ts-chief: A scalable and accurate forest algorithm for time series classification," *Data Mining and Knowledge Discovery*, pp. 1–34, 2020.

[5] Jason Lines, Sarah Taylor, and Anthony Bagnall, "Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles," *ACM Transactions on Knowledge Discovery from Data*, vol. 12, no. 5, 2018.

[6] Angus Dempster, François Petitjean, and Geoffrey I Webb, "Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1454–1495, 2020.

[7] Javier Contreras, Rosario Espinola, Francisco J Nogales, and Antonio J Conejo, "Arima models to predict next-day electricity prices," *IEEE transactions on power systems*, vol. 18, no. 3, pp. 1014–1020, 2003.

[8] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[10] Sepp Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[11] Paul J Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, "Learning internal representations by error propagation," Tech. Rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[13] Yann LeCun, Yoshua Bengio, et al., "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, pp. 1995, 1995.

[14] Abien Fred Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[16] Minh-Thang Luong, Hieu Pham, and Christopher D Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.