Recurrent neural networks approach to the financial forecast of Google assets

Luca Di Persio and Oleksandr Honchar

***Abstract*—**A huge quantity of learning tasks have to deal with sequential data, where either input or out-put data can have sequential nature. This is the case,e.g., of time series forecasting, speech recognition,video analysis, music generation, etc., since they all require algorithms able to model sequences. Duringrecent years, recurrent neural networks (RNNs) architectures have been successfully used in one as well as for multidimensional sequence learning tasks, quickly constituting the state of the art option for extracting patterns from temporal data. Concerning financial applications, one of from the most important examples of sequential data analysis problems is related to the forecasting the dynamic in time of structured financial products. To this end, we compare different RNNs architectures. In particular we consider the basic multi-layer RNN, long-short term memory (LSTM) and gated recurrent unit (GRU) performances on forecasting Google stock price movements. The latter will be done on different time horizons, mainly to explain associated hidden dynamics. In particular, we show that our approach allows to deal with long sequences, as in the case of LSTM. Moreover the obtained performances turn out to be of high level even on different time horizons. Indeed, we are able to obtain up to 72% of accuracy.

***Keywords*—** Artificial neural networks, Deep Learning, Financial forecasting, Gated recurrent unit, Long short-term memory, Multi-layer neural network, Recurrent neural network, Stock markets analysis, Time series analysis.

I. INTRODUCTION

Neural networks (NNs) have been widely recognized as very powerful machine learning models, achieving state-of the art results in a huge range of different machine learning tasks. In perspective of artificial intelligence algorithms NNs are known as connectionist models, since they consist of basic connected units, the artificial neurons, which are jointly combined in layers, that can learn hierarchical representations. During last years, also thank to the exponential growth of computational power, the area of artificial intelligence has gone through a relevant development. The latter is witnessed by the born of the so called deep learning applications. Basically, deep learning models are neural networks with very large size of representation hierarchy, an example being given by multilayer perceptrons (MLPs).

However, such models still suffer of some serious limitations. In fact, when working with sequential data, we can not process related time series at every time step, and saving some entire state of the sequence. This is why, in such scenario, the RNNs option can help a lot. Indeed, RNNs are still connectionist type models, but they pass input data inside the network across time steps, hence processing one element at a time. Different choices to representt emporal data can be given using Hidden Markov Models (HMMs), which are often implemented to model time series as the realization of probabilistically dependent sequence of unknown states. In this context, the usual algorithmic tool is the Viterbi dynamic programming algorithm, that performs efficient inference scales with quadratic time. Since the implementation of RNNs depends only on one single input in a time, this allows to speed up the task when compared with the HMMs approach. It is worth to mention that other, more classical methods can be used to model and forecast time series, as in the case of, e.g., ARMA, ARIMA, GARCH, etc., or using stochastic filter, such the Kalman filter, and switching models approach, as in, e.g., [3, 4]. Nevertheless, black box methods, like the NNs ones, are appealing because they require no prior assumptions on the stochastic nature of the underlying dynamics, see, e.g., [5], and references therein. . The same type of limitations also characterize stochastic filter tools, as for the Kalman case, especially when we aim at studying financial data. Indeed, the Kalman filter does not have enough features to capture rapid movements of stock prices, particularly in case of financial turbu lence, high volatility regimes and complex, interconnected financial networks, see, e.g., [2], and references therein.

Machine learning models in general, and NNs in particular, have been successfully applied in finance, both for forecasting and hedging purposes. For example, portfolio optimization problem [15], where neural networks, genetic algorithms, reinforcement learning, were applied obtaining very promising results. Such type of models can be also applied within the risk management scenario, where risky assets, see [14], can be classified in supervised way by mean of classical machine learning algorithms as random forests, or by using complex classifiers, as deep Nns. In the present paper we consider the forecasting problem of stock price prediction. Concerning the

L. Di Persio - Department of Computer Science - University of Verona - (corresponding author - e-mail: luca.dipersio@univr.it)

O. Honchar - Department of Computer Science - University of Verona (e-mail: oleksandr.honchar@univr.it).

latter we would like to underline that different approaches have been already proposed. Even taking into account only the NNS ones, we can applications belonging to the MLPs methods, see, e.g., [16], convolutional neural networks (CNNs), see, e.g.,[6], Elman neural networks, see, e.g., [10], etc. We decided to focus our attention on the analysis of last state of the art RNNs architectures, paying particlar attention to the GRU and the LSTM.

We also provide some preliminary results about hidden dynamics inside these neural networks with visualization of inner layers activations. In particular, we show on which fluctuations of input time series RNNs are reacting. Our analysis is based on Google stock prices data. Google (now Alphabet Inc.) is one of the most fast growing company in the world, being active on different technology markets, such as web search, advertisements, artificial intelligence, self-driving cars. It is a stable member of S&P Dow Jones Indices, therefore, and there is a great financial interest concerning the forecast of its stock performances. The fact that, due to stable situation of high technologies market, the associated time series dataset are not biased is a relevant feature of Alphabet's financial time series, particularly from the RRNs point of view.

II. RNN ARCHITECTURES - A: RNN

Typically a RNN approach is based on learning from sequences, where the sequence is noting but a list of pairs (x_t,y_t), where x_t, resp. y_t, indicates an input, resp. the corresponding output, at a given time step t. For different types of problems we can have a constant output value y_t=t, for for the whole sequence, or we can choose between a list of desired outputs for every single x_t. To model sequence, at every time step we consider some hidden state. The latter allows the RNN to understands the current state of a sequence, remembers the context and processes it forward to future values. To every new input x_t, a new hidden state, let us indicate it with h_t, is added according to h_(t-1). In the context of so called regular fully-connected neural networks, at every time step the RNN is just a feed-forward neural network with one hidden layer with an input x_t and an output y_t. Taking into account that we are now considering a couple of inputs, x_t and h_(t-1) ,there are three weight matrices, namely W_(hx),for weights from input to hidden layer, W_(hh)from hidden to hidden, and W_(yh) for the output's weights. The resulting basic equations for RNN are the following:

$$s_t = tanh(W_{hx}x_t + W_{hh}s_{t-1} + b_t)$$
$$o_t = softmax(W_{yh}s_t),$$

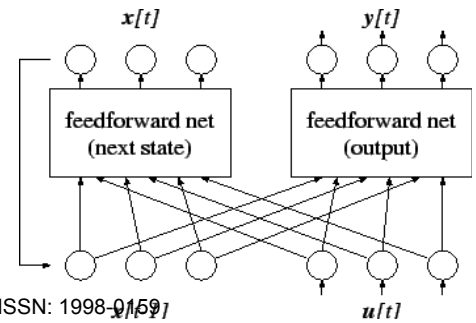


Figure 1: Recurrent neural network diagram

The training procedure for RNNs is usually represented by the so called backpropagation through time (BPTT) algorithm. The latter is derived analogously as the basic backpropagation one. Since the weight update procedure is typically performed by an iterative numerical optimization algorithm, which uses n-th order partial derivative, e.g. first order in case of the stochastic gradient descent, we need all the partial derivatives of the error metric with respect to the weights. The loss function can be represented by a negative log probability, namely

$$\frac{-1}{N}\sum_{i=1}^{N} \ln p_{target_t}.$$

To realize the BPTT algorithm, we first have to initialize all the weight matrices with random values. Then the following steps are repeated until convergence:

- U Unfold RNN for N time steps to get basic feed forward neural network
- Set inputs to this network to zero vectors
- Perform forward and backward propagation as in a feed-forward network for single training example
- Average gradients in every layer to update weight matrices on every time step the same way
- Repeat steps above for every training example in dataset

III. RNN ARCHITECTURES - B: LSTM

Basic RNNs perform particularly well in modeling short sequences. Nevertheless, they show a rather ample set of problems. This is, e.g., the case of vanishing gradients, where the gradient signal gets so small that learning becomes very slow for long-term dependencies in the data. On the other hand, if the values in the weight matrix become large, this can lead to a situation where the gradient signal is so large that the learning scheme diverges. The latter is often called exploding gradients. In order to overcome problems with long sequences an interesting approach for long-short term memory has been developed by Schmidhuber in [11], see the scheme of one LSTM cell on figure 2.

Comparing to RNNs, LSTM's single time step cell has a more complex structure then just hidden state, input and output. Inside these cells, often called memory blocks, there are three adaptive and multiplicative gating units, i.e. the input gate, the forget gate and the output gate. Both input and output gates have the same role as in the RNNs input and outputs cases, with corresponding weights. The new instance, namely the forget gate, play the role of learning how to remember or to

forget its previous state. This latter feature allows to catch more complex temporal patterns. The forward propagation equations characterizing the LSTM gates, read as follows:

$$i_t=\sigma\left(W_i x_t+U_i h_{t-1}+b_i\right)$$
$$c_i n_t=tanh\left(W_c x_t+U_c h_{t-1}+b_{c_i n}\right)$$
$$f_t=\sigma\left(W_f x_t+U_f h_{t-1}+b_f\right)$$
$$o_t=\sigma\left(W_o x_t+U_o h_{t-1}+b_o\right);$$

and for the forget state update, we have:

$$c_t=f_t\cdot c_{t-1}+i_t\cdot c_i n_t$$
$$h_t=o_t\cdot tanh\left(c_t\right),$$

where x_t represents the input to the memory cell, while W_i ,W_f ,W_c ,W_o ,U_i ,U_f ,U_c ,U_o are the weight matrices, and b_i ,b_f ,b_c ,b_o are biases.

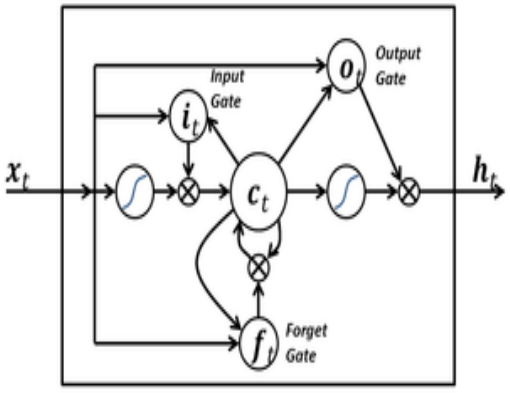


Figure 2: Long-short term memory cell diagram

IV. RNN ARCHITECTURES - C: GRU

In what follows we consider the Gated Recurrent Units (GRUs), see[1]. Basically GRUs are supposed to solve the problem affecting the RNNs architectures. In particular, they use the same gates approach defining the LSTMs, but merging the input gate with the and forget gate, and the same holds for cell state as well as for the hidden state. The result is a lighter model which is supposed to be trained faster, also performing slightly better for some tasks, see [1]. The typical GRU cell diagram can be represented as in figure 3.

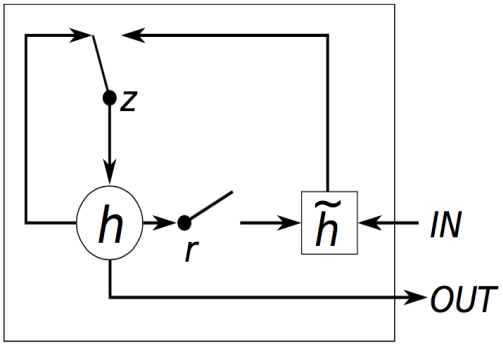


Figure 3: Gated recurrent unit network diagram

The forward propagation equations of typical GRU gates, read as follows:

$$z_t=\sigma\left(W_z x_t+U_z h_{t-1}+b_z\right)$$

$$r_t=\sigma\left(W_r x_t+U_r h_{t-1}+b_r\right.$$
$$h_t=\left(1-z_t\right)\cdot H_{t-1}+¿$$
$$+z_t\cdot\tanh\left(W_h x_t+U_h\left(r_t\cdot h_{t-1}\right)+b_h\right.$$
$$¿¿¿¿$$

where x_t ,h_t ,z_t ,r_t are, respectively, the input, the output, the update gate, and the reset gate vectors, while W,U represent the parameter matrices, and b_z,b_r,b_h are biases. We would likt to mention the comparison considered by in [12] between RNNs, LSTMs, GRUs and other variants of RNNs architectures. The final result clearly show that they all three basically produce the same performances.

V. DATA PREPROCESSING

In what follows we focus our attention on the GOOGL stock prices, exploiting daily data for the last five years, i.e. 2012-2016, see figure 4. Our goal is to forecast the movement's direction of the stock we are interested in, on the basis of historical data. In particular we consider a typical time window of 30 days of open price, high price, low price, close price and volume (OHLCV) data. The first step consists in rescaling our windows, e.g., by normalizing them as follows

$$z=\frac{x-\mu}{\sigma};\mu=\frac{1}{N}\sum_{i=1}^{N}\left(x_i\right);\sigma=\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i-\mu\right)^2},$$

or by a Min-Max type-scaling

$$X_{norm}=\frac{X-X_{min}}{X_{max}-X_{min}}.$$

To perform our analysis we have consider the [−1;1]. This is because, as we better see later, NNs with hyperbolic tangent

activation function will be used, the latter being characterized by an inner activations range equals to [−1;1]. We underline that our prediction object consists in the difference of close prices of last day with respect to a time window and objective range, namely considering the next day, the next 5 days, the 10 days, etc. The resulting differences are then binarized to better show if the close price will go up, or down, which leads to the use of binary vectors [1;0] and [0;1]. Our data set is a set of normalized time windows of 30 minutes with corresponding labels if price goes up or down. We split our dataset into a train set and a test set, with a division of 90% to 10%, respectively). Splits are done in historical order to simulate real world situation, when

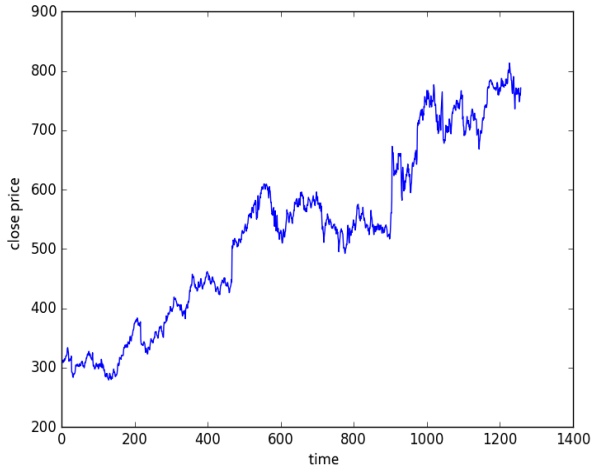


Figure 4: Close prices of GOOGL asset

we train on past data and try to predict future. To avoid overfitting, we shuffle our test and train sets after splitting. After splitting we see, that we have 45% of up labels and 55% down labels in our test dataset. It will be a good test to check, if our algorithm did not overfit - if it shows 55% of accuracy, it shouldn't mean that it predicts better then random guess, it means that it just learnt the distribution in test dataset, with other words, overfitted the dataset.

VI. Experimental Results

In this section we provide the computational results re lated to the training process. All NNs were trained using Keras, which a NNs library written in Python for deep learning. Every network was trained for 100 epochs. This high value has been chosen because experimental results show that training on less epochs causes the deep network to overfit test set and just learn the distribution. Moreover, training for longer time is necessary to better understand convergence trend. If after some time cross-entropy error will start to grow, we can choose model that has the best performance. As optimization algorithm we have used the Adam approach, see [9], with related gradients calculated with BPTT algorithm, while we have used a GPU hardware, namely Nvidia GTX 860M, to reduce computational costs.

A - Performance Analysis

For all RNNs we use the same pattern, namely a two stacked recurrent layers. In this model the output of the first layer constitutes the input of the second and so on, with one affine layer on the top with softmax function on the output to re-sample it as a probability distribution. The activation function of cells is the tanh function, while the function for the inner activation inside the cells is a sigmoid function. Moreover we have use hard approximation of the sigmoid function to speed up the whole procedure. Start weights for inputs are initialized exploiting the Glorot uniform, see [17], hence we have

$$a=\sqrt{\frac{12}{fan_{in}+fan_{out}}}$$
$$¿W\sim U[-a,a],$$

while the inner weights in cells are initialized with orthogonal initialization described by Saxe, McClelland, and Ganguli in [8]. The latter implies that They the weight matrix should be chosen as a random orthogonal matrix, namely a square matrix W such that If you modify this document for use with other

$$W^T W=I.$$

Results for prediction trend for the next day are shown on figures from 5 to 10. The results are then summarized in table on figure 12.

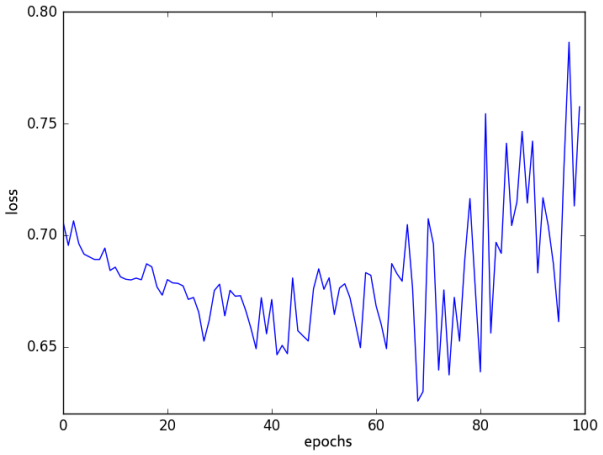


Figure 5: RNN loss within 100 epochs

Looking at the error plots we can easily see that regular RNNs tends to overfit. In fact, cross-entropy value has its minimum around the 70th epoch, starting to grow again after this. It follows, that it is better to use early stopping technique, see [13], to know when optimally stop the training scheme. In case of LSTM and GRU, loss tends to decrease, so these networks can be run more time and on more data, in case of GRU we can see that convergence is smoother then with LSTMs. Time consumption for training 100 epochs is in table 13 We also tried to increase efficiency with dropout technique for U and W weights, see [7], to LSTM and GRU. Nevertheless such

approach did not show improvements for GRU and showed small improvement, at around +2% for LSTM.

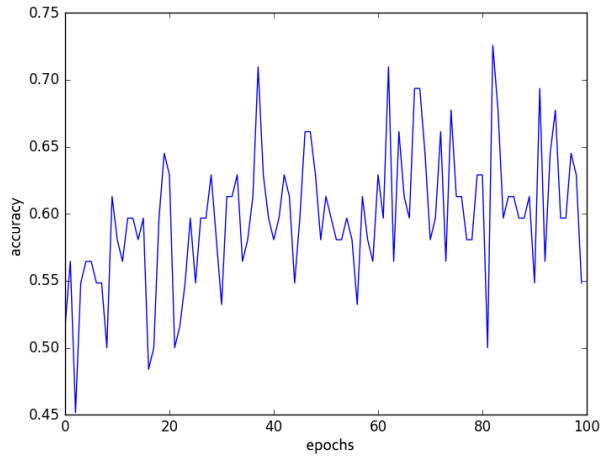


Figure 6: RNN accuracy within 100 epochs

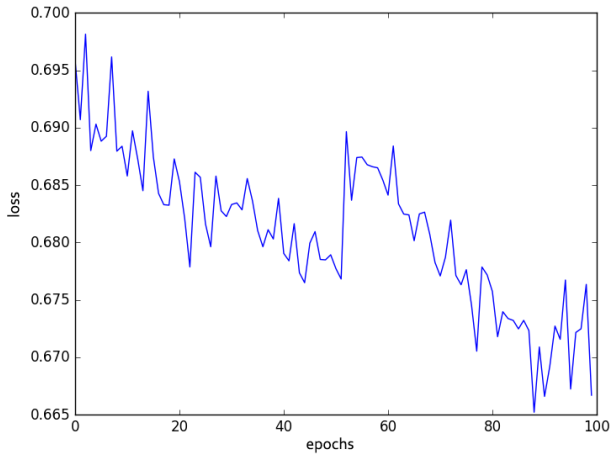


Figure 7: LSTM loss within 100 epochs

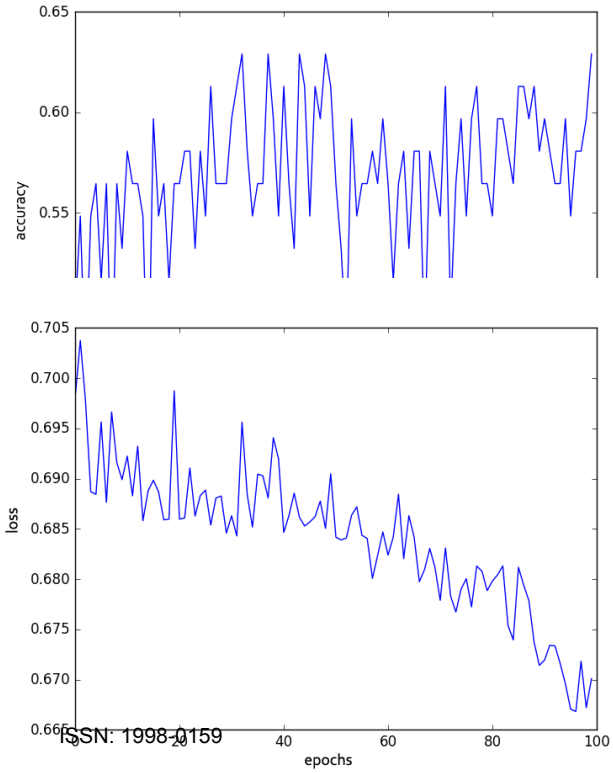


Figure 8: LSTM accuracy within 100 epochs

Figure 9: GRU loss within 100 epochs

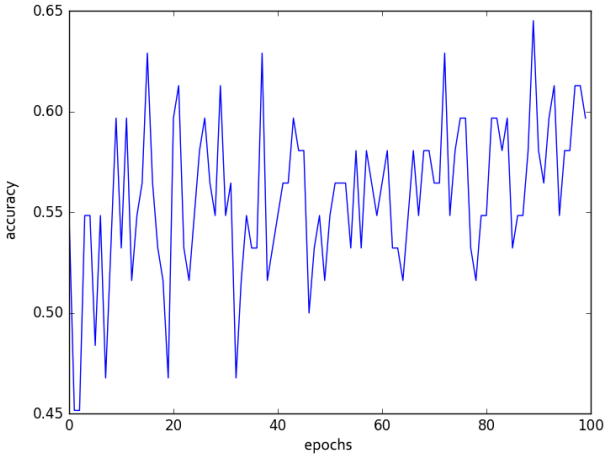


Figure 10: GRU accuracy within 100 epochs

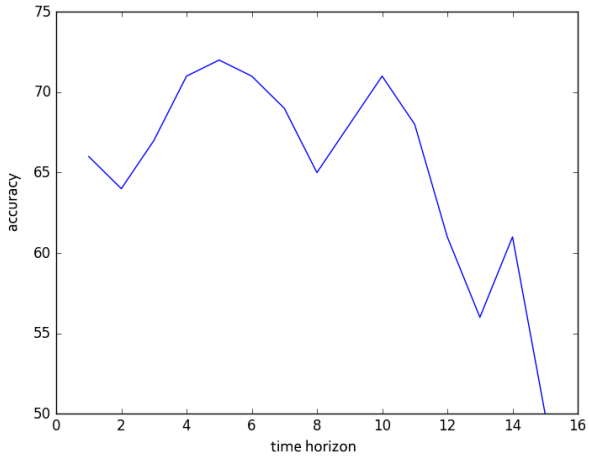


Figure 11: Prediction accuracy for different ranges: from 1 to 15 days

Architecture	Log loss	Accuracy
RNN	0.725	0.625
LSTM	0.629	0.665
GRU	0.629	0.67
LSTM + Dropout	0.645	0.681
GRU + Dropout	0.645	0.664

Figure 12: Losses and accuracy after training different architectures

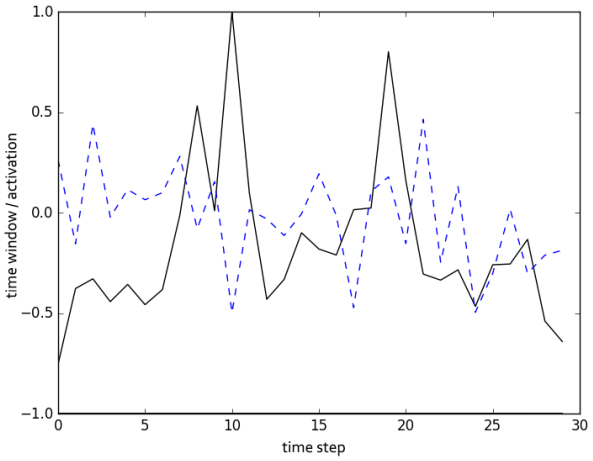
Architecture	Time (sec)
RNN	49
LSTM	189
GRU	216

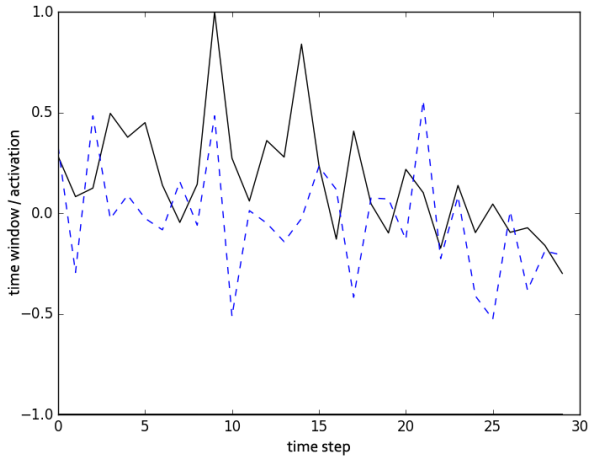
Figure 13: Time for training 100 epochs

are supposed to be totally random, while, in reality, next days trend could be influenced by some earlier fluctuations. In figure 11, results for 1-15 days of prediction have been reported. The accuracy for next day is still not bad, about 66%, but it is much better for 5 days horizon, about 72%, with a small jump to 71% of accuracy on 10-days horizon prediction. Nevertheless it is worth to mention that the latter is a test which is dataset dependent result, and it should vary for different assets. Generally, this plot is showing, that the better horizon of prediction is 1 to 5 days, which totally makes sense.

B- Hidden Dynamics Analysis

To discover hidden behavior of RRNs we have provided a visualization of activations after first recurrent layer. This idea is inspired by the LSTMVis tool, see [18], that can be used to understand hidden state dynamics in LSTM. We have the hypothesis that RNNs can early detect trend of time series movement because





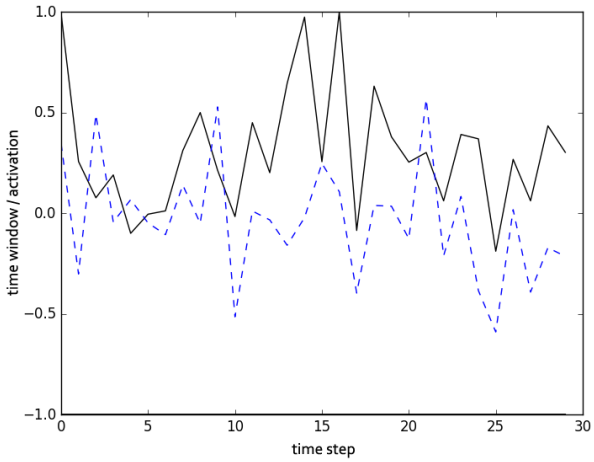


Figure 14: Activation examples on random time windows

of given task to solve. On the figure 14, the black line corresponds to some input time window, and the blue dashed line shows the activations. As we can see, RNNs can discover some useful patterns. In particular, if activation in some moment goes to −0.5, this could be a signal that price will go up in next couple of days, and vice-versa. Namely, if activation goes to 0.5 it could mean that price is going to fall in the closest future. The same holds for activations of second recurrent layer. Such an approach can be used as a powerful indicator also in more complex financial applications or, for example, as an algorithmical trading signals. We intend to deeply go through this latter topic , particularly from the machine learning point of view.

VII. Conclusions

In the present paper we have applied some of the most promising RRNs architectures, namely basic RNNs, LSTMs and GRUs, to stock market price movement forecasting. We have compared results trained on a daily basis for GOOGL stock prices with respect to the last five years, showing that the LSTMs approach is able to provide a high enough accuracy, up to 72% for 5 days prediction horizon. This means that it can be successfully applied in practice. We also show that to avoid overfitting to the dataset, RNNs have to be trained for large number of epochs, choosing final weights carefully with early stopping.

Furthermore we have also performed the analysis of RNNs hidden dynamics. The latter allows us to prove that NNs aren't not black box learning models with non interpretable inner structure. In fact, visualizations of activations clearly show, that NNs can learn useful patterns. In particular, they can detect short term ups and downs in time series. These activations can be used as indicators for further time series analysis.

In future research we plan to apply more bleeding-edge deep learning approaches to financial time series. We will mainly focus on the explanation of how neural attention mechanism, bidirectional RNNs and more complex structures that were successfully applied in NLP problems, can help in learning important parts of time series of interest. We also plan to perform more in-depth research of hidden behaviour of RNNs to use inner activations as technical indicators or feature selectors.

References

[1] Y. Bengio, D. Bahdanau, H. Schwenk et al, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, 2014

[2] C. Benazzoli and L. Di Persio Default contagion in financial networks, International Journal of Mathematics and Computers in Simulation Volume 10, 2016, Pages 112-1175

[3] L. Di Persio and M. Frigo, Maximum likelihood approach to markov switching models, WSEAS Transactions on Business and Economics Volume 12, Pages 239-242, 2015

[4] L. Di Persio and M. Frigo, Gibbs sampling approach to regime switching analysis of financial time series, Journal of Computational and Applied Mathematics Volume 300, Pages 43-55, 2016

[5] L. Di Persio and O. Honchar, Artificial neural networks architectures for stock price prediction: Comparisons and applications, International Journal of Circuits, Systems and Signal Processing, Volume 10, Pages 403-413, 2016

[6] X. Ding, Y. Zhang et al, Deep Learning for Event-Driven Stock Prediction, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 2015

[7] N.Srivastava, G. Hinton et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting *Journal of Machine Learning Research 15*, 2014

[8] A. Saxe, J.L. McClelland, S. Ganguli, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2014

[9] Diederik Kingma, Jimmy Ba Adam: A Method for Stochastic Optimization, arXiv:1412.6980, 2015

[10] Jie Wang, Jun Wang Forecasting energy market indices with recurrent neural networks: Case study of crude oil price fluctuations, 2016

[11] S.Hochreiter, J.Schmidhuber Long-short term memory, *Neural Computation*, 1997

[12] K. Greff, R. K. Srivastava et al, LSTM: A Search Space Odyssey, 2015

[13] Girosi, Federico, M. Jones, Regularization Theory and Neural Networks Architectures, *Neural Computation p.219–269* , 1995

[14] F. Butaru, Q. Chen et al, Risk and Risk Management in the credit Card Industry , 2015

[15] E. Hurwitz, T. Marwala, State of the Art Review for Applying Computational Intelligence and Machine Learning Techniques to Portfolio Optimisation *preprint*, 2009

[16] M. Naeini, H. Taremian, Stock Market Value Prediction Using Neural Networks, *International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, 2010

[17] Xavier Glorot and Yoshua Bengio, Understanding the difficulty of training deep feedforward neural networks. *International conference on artificial intelligence and statistics*, 2010

[18] H. Strobelt, S. Gehrmann, B. Huber et al, Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks, 2016