Deep Learning Approach for Prognoses of Long-Term Options Behavior

A. V. Bogdanov[1,2(✉)], S. A. Bogdanov[1,2], V. P. Rukovchuk[1,2], and D. S. Khmel[1,2]

[1] Plekhanov Russian University of Economics, Stremyanny lane, 36, Moscow 117997, Russia
bogdanov@csa.ru
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany

Abstract. Since the inception of asset pricing models, starting as far back as beginning of XX century, and moreover after the fundamental work of Black and Scholes (1973), there has been considerable interest in analytical research of stock exchange equities behavior. Still up to nowadays it remains a critical task for participants engaged in the field of "financial mathematics". The reason of such an undying interest is that to adequately assess investments risks the stock exchange actors (brokers, investors, traders, et. al) need still more and more accurate prediction results obtained as fast as possible. It is a matter of fact proven by numerous researchers that assets derivatives behave differently being observed in small, medium, and long-term frames.

Algorithms for predicting the dynamics of stock options and other assets derivatives for both small times (where one plays on market fluctuations), and medium ones (where trade is stressed at the beginning and closing moments) are well developed, and trading robots are actively used for these purposes. Analysis of the dynamics of assets for very long time-frames (of order of several months and years) is still beyond the scope of analysts as it is expensively prohibited, although this issue is extremely important for hedging the investments portfolios.

The present paper focuses on construction of an effective and resource-intensive model for predicting the behavior of financial instruments, trends and price movements based upon the principles of deep learning. The forecasts obtained by the model showed an almost acceptable compliance with the true prices of the S&P500.

Keywords: Deep learning · S&P500 · Recurrent neural networks · LSTM · TensorFlow

1 Introduction

Many scientists investigating the forecasting of stock and financial markets are trying to introduce machine learning. So, Robert P. Schumaker and Xinchun Chen from the University of Arizona conducted a textual analysis of the stock market forecast using the latest financial news [1]. Their study examines the method of machine learning on financial news using several different textual representations: Bag of Words [2], nominal groups (IG) and named entities. They examined 9,216 financial news articles

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11620, pp. 631–640, 2019.
https://doi.org/10.1007/978-3-030-24296-1_51

and 10,259,042 stock quotes covering the S & P 500 for five weeks. The resulting model was used to estimate a discrete stock price twenty minutes after the publication of a news article. The forecast accuracy was 57%, and profit growth was 2%.

Raj Thakur, Sanchita Badkas, Minakumari Pade and Pallavi Hude. Patila Technical College [3] built a hybrid system on neural networks that tries to predict the price of opening the stock market for several specific companies based on the company's past activity and information from various news sources, combining elements of fundamental and technical analysis and confirming the viability of this approach.

Felix Min Fai Wong, Zhenmin Liu, Mung Chang from Princeton [4] built a system that predicts price movements only from Wall Street Journal news using Text Mining and sparse matrix factorization. Unlike many existing approaches, the new model can simultaneously use correlations: (a) between stock prices, (b) among news articles, and (c) between stock prices and news articles. Thus, this model is able to predict more than 500 stocks daily with an accuracy of 55.7%.

There are now many open source tools and libraries for creating deep learning models. Most frameworks present a model in the form of a computational graph, static or dynamic:

- The frameworks that declare calculation graphs statically, such as TensorFlow, allow you to first define the architecture of the graph, and then you can run it many times, running some amount of data through it. This makes it easy to distribute tasks for execution on different machines. In addition to this, frameworks can optimize the graph for you in advance. They also allow users to serialize a schedule after it has been created and execute it without requiring the code that built it.
- Frameworks that declare calculation graphs dynamically, such as PyTorch, allow you to implicitly define a computational graph. Dynamic graph structures are usually less invasive and deeply integrated with the programming language used. This leads to a cleaner and more easily debugged code and allows the use of conditional operators and cycles to build more complex graph structures, such as recurrent neural networks. The construction and execution of the graph are intertwined, which leaves little time for its optimization.

Stating ultimately, the recurrent neural network (RNN) architecture [5], built on the elements of long-time short-term memory (LSTM) [6], was chosen for providing learning procedure. Since the historical process of price movements is a time series, we need to regularly refer to it and to take into account the long-term context for which implementation, in fact, RNN with LSTM is intended.

To implement the selected model, the choice fell on TensorFlow. The popularity of this framework is well deserved: we get support from a large number of developers, extensive documentation choices, open source code, distributed execution of our models, various architectural optimizations, visualization at runtime (which makes it easier to debug the model), and portability to the extent that we can run the model on smartphones with Android.

Next, we will discuss how to build a RNN model with LSTM cells for predicting the S & P500 index.

2 Data Preparation

In this paper, we used data from the S & P 500 from January 3, 1950 (the earliest date that Yahoo! Finance can track) until April 22, 2019. The dataset provides several price points per day. For simplicity, we will use only daily closing prices for forecasting.

A recurrent neural network (RNN) is a type of artificial neural network with recurrent transitions in hidden layers, which allows it to use the previous state of hidden neurons to obtain a new result with new input. RNN can handle sequential data. Long Short-Term Memory (LSTM) is a specially designed cell that helps the RNN better remember the long-term context. Stock prices are time series with lengths N, defined as $p_0, p_1, \ldots, p_{N-1}$, where p_i – is a closing price at date i, $0 \leq i < N$. Imagine that we have a fixed-size sliding window w (hereinafter referred to as *input_size*), and we move it to the right by w so that there is no overlap between the data in all sliding windows (Fig. 1).

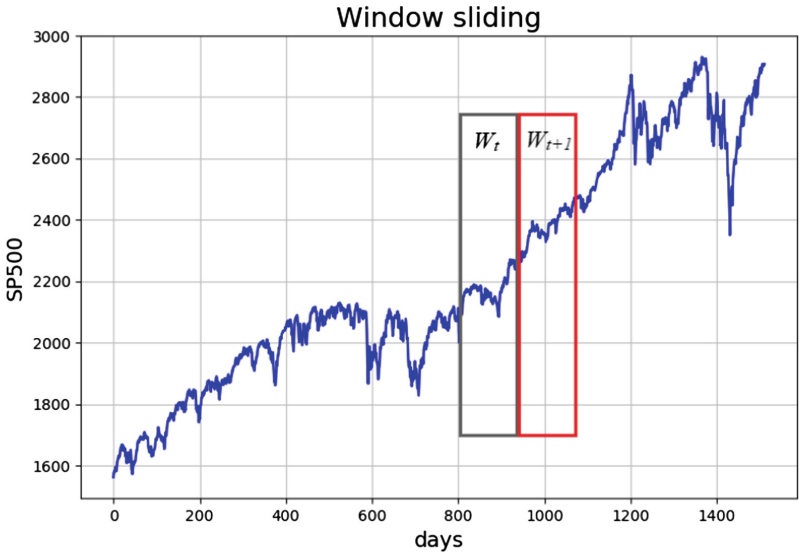

Fig. 1. S & P 500 prices in time. We use data in one sliding window to make a prediction for the next, while there is no overlap between two consecutive windows.

The RNN model we are building has LSTM cells as major hidden elements. We use values from the very beginning of training in the first sliding window W_0 till window at the moment t:

$$\begin{aligned} W_0 &= (p_0, p_1, \ldots, p_{w-1}) \\ W_1 &= (p_w, p_{w+1}, \ldots, p_{2w-1}) \\ &\ldots \\ W_t &= \left(p_{tw}, p_{tw+1}, \ldots, p_{(t+1)w-1}\right) \end{aligned}$$

to predict prices in the next window W_{t+1}:

$$W_{t+1} = \left(p_{(t+1)w}, p_{(t+1)w+1}, \ldots, p_{(t+2)w-1}\right)$$

Essentially, we are trying to find an approximation function:

$$f(W_0, W_1, \ldots, W_1) \approx \sim W_{t+1}$$

The sequence of prices is first divided into small non-overlapping windows. Each one contains *input_size* numbers, each of which is considered as one independent input element. Then any consecutive input elements *num_steps* grouped into one input for learning, forming a "non-twisted" version of the RNN for training in TensorFlow. The corresponding label is an input element immediately after them

E.g. if *input_size* = *3* and *num_steps* = *2*, then the first few examples of training will look like this:

$$\begin{aligned}
\text{Input}_1 &= [[p_0, p_1, p_2], [p_3, p_4, p_5]],\ \mathit{Label}_1 = [p_6, p_7, p_8] \\
\text{Input}_2 &= [[p_3, p_4, p_5], [p_6, p_7, p_8]],\ \mathit{Label}_2 = [p_9, p_{10}, p_{11}] \\
\text{Input}_3 &= [[p_6, p_7, p_8], [p_9, p_{10}, p_{11}]],\ \mathit{Label}_3 = [p_{12}, p_{13}, p_{14}]
\end{aligned}$$

Since we always want to predict future values, we take the last 10% of the data as test values.

3 Data Normalization

The S & P500 index increases with time, which leads to the fact that most of the values in the test set are out of tolerance, and thus the model must predict values that it has never seen before. It is not surprising that sooner or later the model ceases to behave adequately (Fig. 2).

To solve the problem with the scale, it is necessary to normalize prices in each sliding window. Now the task is to predict the relative levels of change instead of absolute values. In normalized sliding window W'_t at the moment t all values are divided by the last unknown price - the last price in W_{t-1}. Thus, the deviation from the fixed last value of the previous window is taken as the normalization factor:

$$W'_t\left(\frac{p_{tw}}{p_{tw-1}} - 1, \frac{p_{tw+1}}{p_{tw-1}} - 1, \ldots, \frac{p_{(t+1)w-1}}{p_{tw-1}} - 1\right)$$

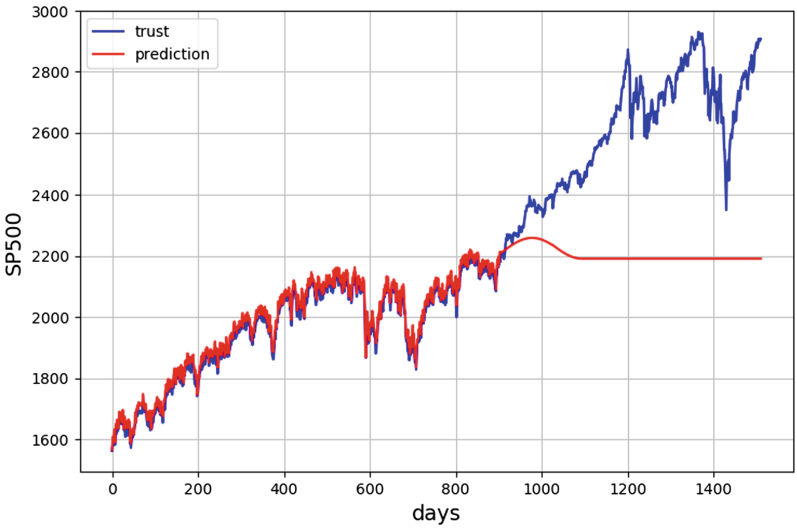

Fig. 2. An example when RNN tries to predict values that lie outside the training data.

4 Building the Model

Definitions to be used in the model (Table 1)

Table 1. Model Definitions in TensorFlow

Константа	Описание
lstm_size	The number of elements in one LSTM layer
num_layers	The number of LSTM-layers
keep_prob	The percentage of items to save in the exception operation
init_learning_rate	Initial learning rate
learning_rate_decay	Rate of learning attenuation in finite epochs of learning
init_epoch	The number of epochs using constant init_learning_rate
max_epoch	Total number of learning epochs
input_size	Sliding window size/one tutorial data point
batch_size	Number of data points for use in one mini-batch
embedding_size	Controls the size of each embedding vector
stock_count	The number of unique stocks in the data set

LSTM-model has *num_layers* of combined LSTM layers, and each layer contains *lstm_size* of LSTM-elements. Then, an exception mask is applied to the output of each LSTM element with the probability of saving *keep_prob*. The goal of the exception is to eliminate the potential strong dependence on a single dimension in order to prevent overtraining [7].

The learning requires *max_epoch* of epochs overall, an epoch is one complete pass through all training data. For one epoch, the training data is divided into mini-batch size *batch_size*. The learning rate if fixed as an *init_learning_rate* for the first epochs *init_epoch*, and then gets decreased by *learn_rate_decay* for each consecutive one.

It is expected that the model will study the price sequence in time of different companies. Because of the different patterns, I would like the model to clearly speak with what stocks it works. Investing is profitable compared to one-hot encoding [8], because:

- Given that *N* shares are included in the training data, one-hot encoding will result in additional sparse *N* (or *N-1*) parameters. As each share is mapped to a much smaller embedding vector of length *k*, $k \ll N$, we get a much more concise view and a smaller data set.
- Since investment vectors are learning variables, similar actions can be associated with similar investments and help predict each other, such as «GOOG» and «GOOGL».

In a recurrent neural network at one time step t, the input vector contains *input_size* (marked as *W*) daily price values of i-share, ($p_{i,tw}$, $p_{i,tw+1}$, …, $p_{i,(t+1)w-1}$). The share is uniquely displayed on the length vector *embedding_size* (marked as *k*), ($e_{i,0}$, $e_{i,1}$, …, $e_{i,k}$). As shown in Fig. 3, the price vector is concatenated with the attachment vector, and then fed to the LSTM cell.

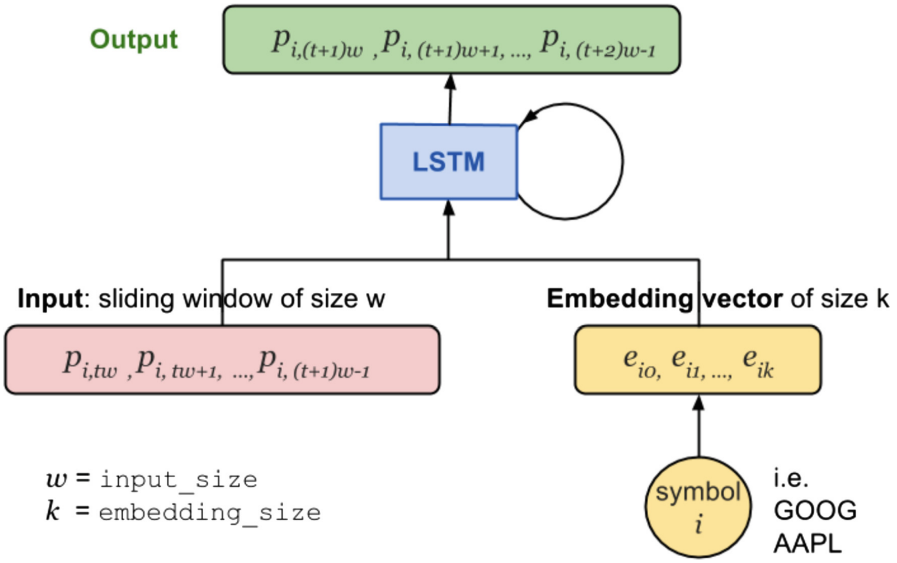

Fig. 3. RNN model for predicting stock prices with an investment.

Another alternative is to concatenate the attachment vectors with the last state of the LSTM cell and obtain new weights W and offset b in the output layer. However, in this way, an LSTM element cannot distinguish the prices of one stock from another, and its power will be largely held back.

5 Experiments

For the first one, the following configuration was used (Table 2):

Table 2. Experiment 1.

Parameter	Value
num_layers	1
keep_prob	0.8
batch_size	64
init_learning_rate	0.001
learning_rate_decay	0.99
init_epoch	5
max_epoch	50
num_steps	30

In general, predicting stock prices is not an easy task. Especially after normalization, price trends look very noisy (Figs. 4a, 4b, 4c).

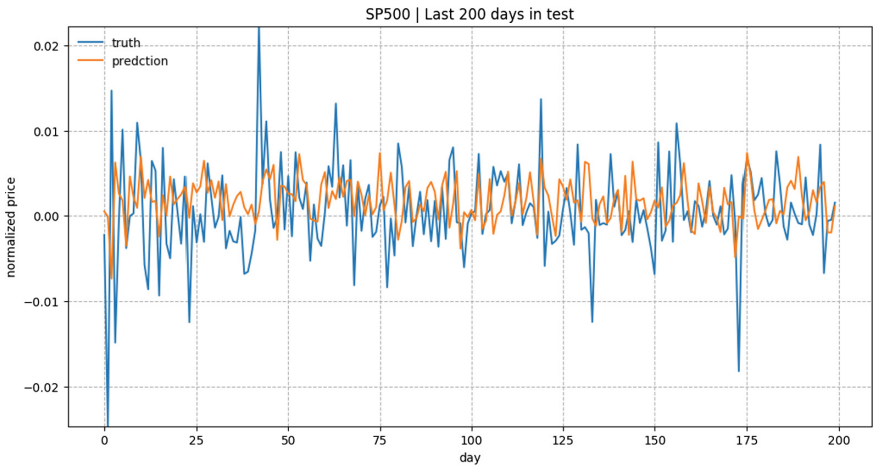

Fig. 4a. The results of the forecast for the last 200 days in the test data. The model learns with *input_size* = 1 and *lstm_size* = 32.

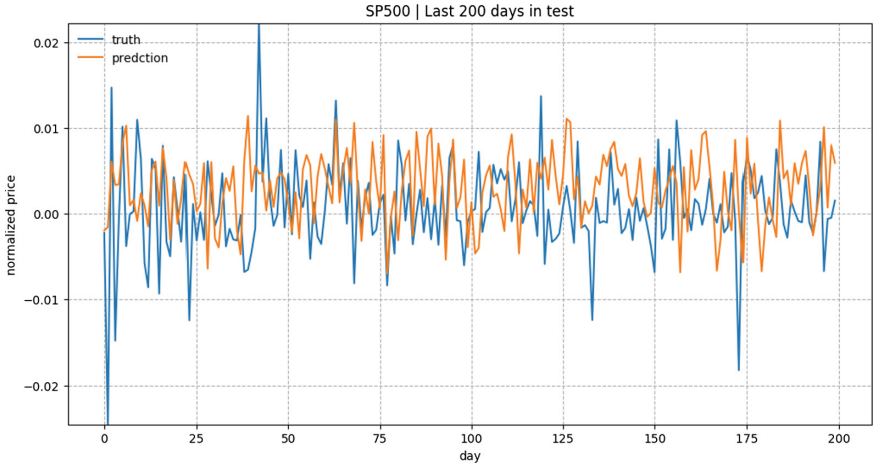

Fig. 4b. The results of the forecast for the last 200 days in the test data. The model learns with *input_size* = 1 and *lstm_size* = 128.

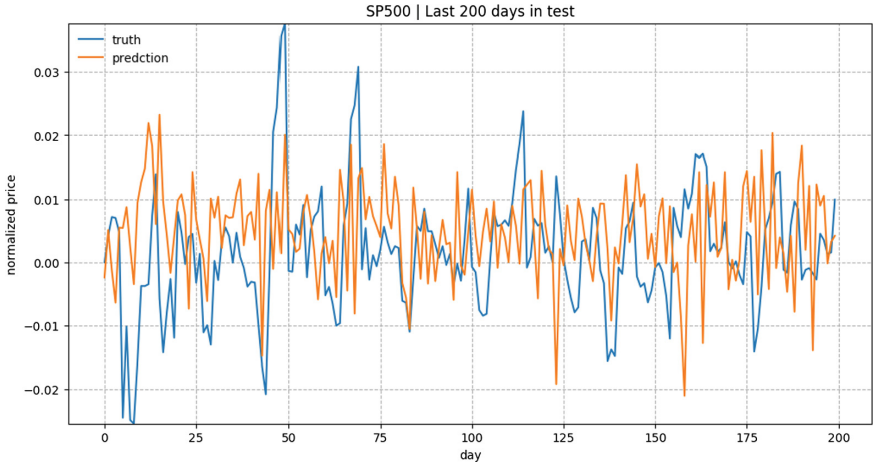

Fig. 4c. The results of the forecast for the last 200 days in the test data. The model learns with *input_size* = 5, *lstm_size* = 128 and *max_epoch* = 75 (instead of 50).

For the second experiment, the following configuration was used (Table 3):

For a brief overview of the quality of prediction in Fig. 5 presents the forecast for the test data "IBM". General trends coincide between true values and forecasts. Considering how the forecasting task is designed, the model relies on all historical data points to predict only the next 5 days (*input_size*). At low value of *input_size* model, there's no need to worry about a long-term growth curve. But as soon as we increase *input_size*, prediction will get way more complicated.

Table 3. Experiment 2.

Parameter	Value
num_layers	1
keep_prob	0.8
batch_size	64
init_learning_rate	0.05
learning_rate_decay	0.99
init_epoch	5
max_epoch	150
num_steps	30
stock_count	100
input_size	5
embed_size	3
lstm_size	256

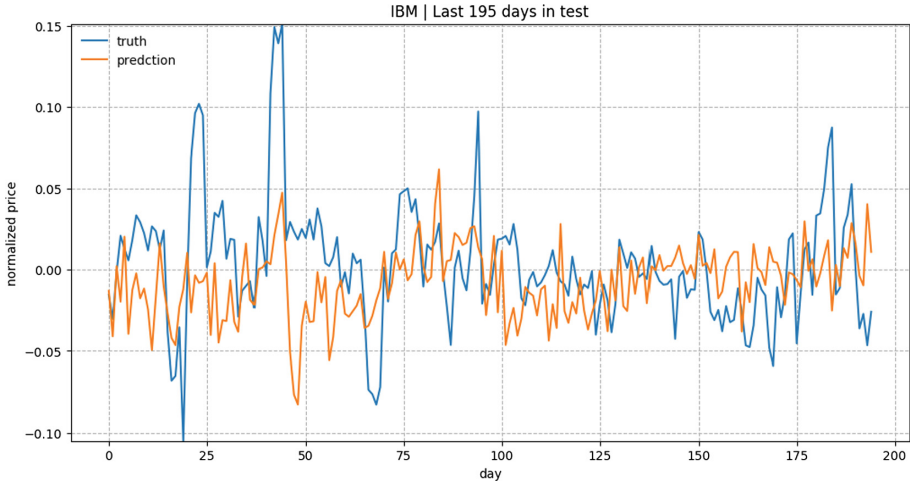

Fig. 5. The true and predicted prices of IBM stocks on a test dataset. Prices are normalized in sliding windows. Y-axis values are multiplied by 2 for a better comparison between true and predicted trends.

Predictive decrease and smooth quite often as you go through the training. Therefore, it is necessary to multiply the absolute values by a constant in order to make the trend more noticeable as in Fig. 5, because we need to know whether the direction of price movement is predicted correctly: up or down. However, there must be a reason for reducing the prediction value. Theoretically, instead of using simple MSE as a loss function, we can choose another method so that if the direction is incorrectly predicted, the penalty will be stronger.

The loss function quickly decreases at the very beginning, but it suffers from an accidental explosion (a sudden peak occurs and then immediately returns). Most likely this is due to its shape. An updated and more adequate loss function can solve the problem.

6 Conclusion

In this paper, we analyzed the various methods of in-depth training, libraries and tools that are in the public domain for the task of forecasting price movements in financial markets. Search, preprocessing and normalization of historical price data S & P500 for training and model testing.

In the process, a software package was obtained for predicting the direction of price movement that implements the recurrent neural network architecture (RNN) built on the elements of long short-term memory (LSTM). The choice was made in favor of just such an architecture, since the price movement history is a time series and it is necessary to regularly refer to it and take into account the long-term context, and the RNN with LSTM is intended for this.

There are many ways to improve the result: design of layers and neurons, different initialization and activation schemes, combination with other models (for example, with intelligent text analysis), etc. It is worth mentioning the presence of a whole class of microprocessors and coprocessors (AI accelerators) that can apply to accelerate the work of existing models due to the highest level of parallelism.

References

1. Schumaker, R.P., Chen, H.: Textual Analysis of Stock Market Prediction Using Breaking Financial News: The AZFinText System. Artificial Intelligence Lab, Department of Management Information Systems, The University of Arizona (2006)
2. Obogashchenie modeli Bag of words semanticheskimi svyazyami dlya povysheniya kachestva klassifikacii tekstov predmetnoj oblasti [Enrichment of the Bag of words model with semantic links to improve the quality of text classification in the subject domain]. http://swsys.ru/index.php?page=article&id=4153. Accessed 04 Apr 2019
3. Thakur, R., Badkas, S., Pade, M., Khude, P.: Stock market prediction using Neural Networks and sentiment analysis of News Articles. D.Y. Patil College of Engineering, Akurdi, Pune (2017). http://dx.doi.org/10.21474/IJAR01/3749. Accessed 04 Apr 2019
4. Ming, F., Wong, F., Liu, Z., Chiang, M.: Stock market prediction from WSJ: text mining via sparse matrix factorization. Princeton University. arXiv:1406.7330v1 [cs.LG] 27 Jun 2014
5. Vvedenie v rekurrentnye nejroseti: RNN, dvunapravlennaya RNN, LSTM, GRU [Introduction to recurrent neural networks: RNN, bidirectional RNN, LSTM, GRU]. https://trainmydata.com/article/vviedieniie-v-riekurrientnyie-nieirosieti-rnn-dvunapravliennaia-rnn-lstm-gru. Accessed 04 Apr 2019
6. Understanding LSTM Networks http://colah.github.io/posts/2015–08-Understanding-LSTMs. Accessed 04 Apr 2019
7. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, L., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15** (2014). https://www.cs.toronto.edu/∼hinton/absps/JMLRdropout.pdf. Accessed 04 Apr 2019
8. How to One Hot Encode Sequence Data in Python. https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python. Accessed 04 Apr 2019. Author, F.: Article title. Journal **2**(5), 99–110 (2016)