Examination of stock predictions using RNN & LSTM

Talor R Braly
College of Natural Sciences and Mathematics
University of Texas at Dallas
Plano, TX, USA
Trb180000@utdallas.edu

Gary Chen
School of Engineering and Computer Science
University of Texas at Dallas
Richardson, Texas
gxc097020@utdallas.edu

Nancy Dominguez
School of Engineering and Computer Science
University of Texas at Dallas
Richardson, Texas
dominguez.nancy@utdallas.edu

Jonathan Hulsey
College of Natural Sciences and Mathematics
University of Texas at Dallas
Grand Prarie, Texas
jdh150330@utdallas.edu

***Abstract*—As they are effective in a broad range of applications, LSTMs are extensively covered in a number of technical and scientific blogs and journals. This breadth of applications led us to focus our attention here, as its examination is likely to provide utility in any of a number of future fields. The goal of this paper is to present the essentials of RNN and its refinement, LSTM, in a clear and approachable format. We present a brief look at the background and origins of the modeling technique. Then, we discuss the theory underlying the technique, explaining how raw data is transformed into useful information and predictions. Next, we will examine the results obtained from the model and discuss the expectations these imply. Finally, we will present our conclusions, in particular the benefits gained by adding LSTM to the base RNN.**

Keywords—recurrent neural network, long short-term model, prediction, modeling

I. Introduction and Background

A. *Recurrent Neural Networks*

Recurrent Neural Networks (RNNs) are specialized Artificial Neural Networks (ANNs) which are particularly well suited for dealing with sequences, especially text and time sequences. RNNs were originally based upon the 1974 work of David Rumelhart, an American psychologist, specializing in the formal study of human cognition, and with essential work in the application of back-propagation, deep learning, and Artificial Neural Networks (ANNs) [1]. This foundation lead the way to Hopfield Networks, an inter-referential network of binary nodes, developed in 1982 by John Hopfield, an American Physicist who would become the Howard A. Prior Professor of Molecular Biology, Emeritus [2]. The first true RNN emerged in 1993, when a “very deep learning” task was solved by a neural history compression system; using over 1,000 subsequent layers in an RNN unfolded in time [3].

B. *Long Short-Term Memory*

Long Short-Term Memory (LSTM) is a refinement of RNNs which allow a model to include and extrapolate from dependencies between nodes which are more remote in the sequence than those available in traditional RNN models. LSTM was developed, as a further development of the earlier RNN framework, by the joint work of Sepp Hochreiter, a German Computer Scientist, specializing in Machine Learning, Deep Learning, and Bioinformatics, who now leads the Institute for Machine Learning at the Johannes Kepler University of Linz [4], and Jürgen Schmidhuber a Computer Scientist specializing in Artificial Intelligence, Deep Learning, and ANNs, who would later become a co-director of the Dalle Molle Institute for Artificial Intelligence Research in Manno [5], in 1997 [6].

II. Theoretical Framework

A. *Recurrent Neural Networks*

i. <u>Purpose and Structure</u>

RNNs primary mechanism is to iteratively update a hidden state h_i, a vector of arbitrary dimensionality, at any step t_i, when given x_i, the input vector, to predict y_i, with x_i and y_i also having arbitrary and independent dimensionality. Unlike classic NN, which have may have distinct weights for each vertex and biases for each layer, RNNs only have three weights:

W_{xh} The weight for each connection $x_t \to h_t$

W_{hh} The weight for each connection $h_{t-1} \to h_t$

W_{hy} The weight for each connection $h_t \to y_t$

and two biases:

b_h The bias for determining each h_t

b_y The bias for determining each y_t

The whole of the RNN may thus be represented as the weights in a *matrix* and the biases in a *vector*. The hidden states may be found using:

$$h_t = [some\ activation\ function](W_{hx}x_t + W_{hh}h_{t-1} + b_h) \quad (1)$$

and the output at any given step is:

$$y_t = W_{hy}h_t + b_y \quad (2)$$

ii. <u>Training</u>

With the structure established, we now consider the training process. Taking the following definitions for our training model:

$y \rightarrow$ raw output from the RNN

$p \rightarrow$ final probabilities $=$ softmax(y)

$c \rightarrow$ the true value of the target

$L \rightarrow$the cross-entropy loss = -ln(p_c)

$W_{xh}, W_{hh}, W_{hy} \rightarrow$ weight matrices of the RNN

$b_h, b_y \rightarrow$ bias vectors of the RNN

(Note that W_{xh}, W_{hh} and b_h are only used in classic RNNs. In LSTM, W_{xh} is replaced with W_{xc}, W_{xf}, W_{xi}, W_{xo}; W_{hh} is replaced with W_{hc}, W_{hf}, W_{hi}, W_{ho}, W_{hz}; and b_h is replaced by b_c, b_f, b_i, b_o)

First, we calculate $\frac{\partial L}{\partial y}$

Since

$$L = -\ln(p_c) = -\ln\left(\text{softmax}(y_c)\right) \quad (3)$$

Then

$$\frac{\partial L}{\partial y_i} = \begin{cases} p_i & \text{if } i \neq c \\ p_i - 1 & \text{if } i = c \end{cases} \quad (4)$$

Next, the gradients for W_{hy} and b_y

$$\frac{\partial L}{\partial W_{hy}} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial W_{hy}} \quad (5)$$

$$\frac{\partial y}{\partial W_{hy}} = h_n \quad (6)$$

Where h_n is the final hidden state

$$\frac{\partial L}{\partial W_{hy}} = \frac{\partial L}{\partial y} h_n \quad (7)$$

and

$$\frac{\partial y}{\partial W_b} = 1 \quad (8)$$

$$\frac{\partial L}{\partial b_y} = \frac{\partial L}{\partial y} \quad (9)$$

Finally, we find the gradients for W_{xh}, W_{hh}, and b_h.

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y} \sum_t \frac{\partial y}{\partial h_t} * \frac{\partial h_t}{\partial W_{xh}} \quad (10)$$

As changes to W_{xh} alter each h_t, it is necessary to backpropagate across each timestep, a process called Backpropagation Through Time (BPTT). W_{xh} is used to calculate each $x_t \rightarrow h_t$, so we must backpropagate through each of those links.

For each step t we must calculate $\frac{\partial h_t}{\partial W_{xh}}$

Using tanh as the activation function

$$h_t = tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \quad (11)$$

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x) \quad (12)$$

By applying the chain rule we find

$$\frac{\partial h_t}{\partial W_{xh}} = (1 - h_t^2)x_t \quad (13)$$

and

$$\frac{\partial h_t}{\partial W_{hh}} = (1 - h_t^2)h_{t-1} \quad (14)$$

$$\frac{\partial h_t}{\partial b_h} = (1 - h_t^2) \quad (15)$$

Finally, we recursively calculate $\frac{\partial y}{\partial h_t}$

$$\frac{\partial y}{\partial h_t} = \frac{\partial y}{\partial h_{t+1}} * \frac{\partial h_{t+1}}{\partial h_t} \quad (16)$$

$$= \frac{\partial y}{\partial h_{t+1}} (1 - h_t^2) W_{hh} \quad (17)$$

We now implement BPTT, working from the final hidden state, h_n, with:

$$\frac{\partial y}{\partial h_n} = W_{hy} \quad (18)$$

Once all of the gradients have been calculated, weights and biases are found using gradient decent [7].

B. *Long Short-Term Memory*

i. Purpose and Structure

LSTM addresses, primarily, two issues with RNNs. The first is long term dependency. RNNs are ill suited to detecting and applying relations to nodes the more remote they are to one another within the sequence. A classic example might be a comparison between the following two strings:

"Before I went to *France*, I learned *French*."

and

"Before traveling to *France*, I made many preparations…, and learning *French*."

An RNN might easily predict the "French" at y_n, given "*France*" at x_{n-3}, in the first string. It is less likely that in the second string an RNN would be able to properly find and exploit the connections between "*France*" at x_{n-k} and the needed "French" at y_n.

The second issue is that of exploding or vanishing gradients. This issue was discussed at length in class, but briefly it occurs in an RNN when the gradient's absolute values exceed one (an exploding gradient) or approach zero (a vanishing gradient).

LSTM is composed of three types of gates: input, forget, and output. Each of the three gates are modeled by a sigmoid activation function, allowing for only values from zero to one, with the value representing the proportion of things being allowed through the gate (zero allows nothing through, one allows everything through).

ii. Training

We will take the following definitions for our LSTM training model:

$$i_t \rightarrow input\ gate$$
$$f_t \rightarrow forget\ gate$$
$$o_t \rightarrow output\ gate$$
$$i_t \rightarrow input\ gate$$
$$\sigma \rightarrow sigmoid\ function$$
$$w_x \rightarrow weight\ for\ the\ respective\ gate(x)\ neurons$$
$$h_{t-1} \rightarrow return\ of\ the\ previous\ lstm\ block$$
$$x_t \rightarrow input\ at\ the\ curent\ index$$
$$b_x \rightarrow biases\ for\ the\ respective\ gate(x)$$

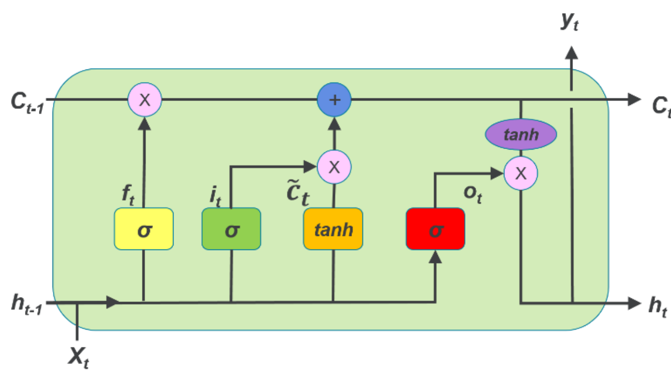

figure 1

The equations for the gates are therefore:

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \quad (19)$$
$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \quad (20)$$
$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \quad (21)$$

The input gate determines what will be stored in the cell state. The second equation, for the forget gate, determines what information will be discarded from the cell state. Finally, the output gate provides the final activation for the LSTM block at index (t).

$$\hat{c}_t \rightarrow candidate\ for\ cell\ state\ (memory)\ at\ index\ (t)$$
$$c_t \rightarrow cell\ state\ (memory)\ at\ index\ (t)$$

The equations for the candidate, cell state, and final output are therefore:

$$\hat{c}_t = \tanh(w_c[h_{t-1}, x_t] + b_c) \quad (22)$$

$$c_t = f_t * c_{t-1} + i_t * \hat{c}_t \quad (23)$$

$$h_t = o_t * \tanh(c^t) \quad (24)$$

The memory vector for any timestamp (t) is found by evaluating $\hat{c}_t$. Any specific cell evaluates what it needs to forget from the previous cell ($f_t * c_{t-1}$) and what it should consider at its current state ($i_t * \hat{c}_t$).

The cell state, once filtered, may be passed to the activation function, which determines what portion should be returned as the output at index (t). This output, h_t, is then passed to the *softmax* layer to predict the output, y_t, of the current block [8].

To backpropagate the LSTM at timestep (t) we use the following:

$$d\boldsymbol{o}_t = \tanh(\boldsymbol{c}_t)\, d\boldsymbol{h}_t \quad (25)$$

$$d\boldsymbol{c}_t = (1 - \tanh(\boldsymbol{c}_t)^2)\ \boldsymbol{o}_t d\boldsymbol{h}_t \quad (26)$$

$$d\boldsymbol{f}_t = \boldsymbol{c}_{t-1} d\boldsymbol{c}_t \quad (27)$$

$$d\boldsymbol{c}_{t-1} = \boldsymbol{f}_t * d\boldsymbol{c}_t \quad (28)$$

$$d\boldsymbol{i}_t = \boldsymbol{g}_t d\boldsymbol{c}_t \quad (29)$$

$$d\boldsymbol{g}_t = \boldsymbol{i}_t d\boldsymbol{c}_t \quad (30)$$

For backpropagation over the whole sequence we find the derivative for the input-to-gate connection weights

$$dW_{xo} = \sum_t \boldsymbol{o}_t(1 - \boldsymbol{o}_t)\boldsymbol{x}_t d\boldsymbol{o}_t \quad (31)$$

$$dW_{xi} = \sum_t \boldsymbol{i}_t(1 - \boldsymbol{i}_t)\boldsymbol{x}_t d\boldsymbol{i}_t \quad (32)$$

$$dW_{xf} = \sum_t \boldsymbol{f}_t(1 - \boldsymbol{f}_t)\boldsymbol{x}_t d\boldsymbol{f}_t \quad (33)$$

$$dW_{xc} = \sum_t (1 - \boldsymbol{g}_t^2)\boldsymbol{x}_t d\boldsymbol{g}_t \quad (34)$$

And the hidden state-to-gate connection weights

$$dW_{ho} = \sum_t \boldsymbol{o}_t(1 - \boldsymbol{o}_t) d\boldsymbol{o}_t \quad (35)$$

$$dW_{hi} = \sum_t \boldsymbol{i}_t(1 - \boldsymbol{i}_t)\boldsymbol{h}_{t-1} d\boldsymbol{i}_t \quad (36)$$

$$dW_{hf} = \sum_t \boldsymbol{f}_t(1 - \boldsymbol{f}_t)\boldsymbol{h}_{t-1} d\boldsymbol{f}_t \quad (37)$$

$$dW_{hc} = \sum_t (1 - \boldsymbol{g}_t^2)\boldsymbol{h}_{t-1} d\boldsymbol{g}_t \quad (38)$$

For the corresponding hidden states at timestep (t-1)

$$d\boldsymbol{h}_{t-1} = \boldsymbol{o}_t(1 - \boldsymbol{o}_t)W_{ho} d\boldsymbol{o}_t + \boldsymbol{i}_t(1 - \boldsymbol{i}_t)W_{hi} d\boldsymbol{i}_t + \boldsymbol{f}_t(1 - \boldsymbol{f}_t)W_{hf} d\boldsymbol{f}_t + (1 - \boldsymbol{g}_t^2)W_{hc} d\boldsymbol{g}_t \quad (39)$$

For the error backpropagation, we use the least square objective function

$$L(\mathrm{x}, \theta) = \min \sum_t \frac{1}{2}(y_t - z_t)^2 \quad (40)$$

With $\theta = \{W_{hc}, W_{hf}, W_{hi}, W_{ho}, W_{hz}, W_{xc}, W_{xf}, W_{xi}, W_{xo}\}$, and ignoring biases. For ease, moving forward, this will be written as

$$L(t) = \frac{1}{2}(y_t - z_t)^2$$

At timestep (T), we take the derivative with respect to c_t

$$\frac{\partial L(T)}{\partial \mathbf{c}_T} = \frac{\partial L(T)}{\partial \mathbf{h}_T}\frac{\partial \mathbf{h}_T}{\partial \mathbf{c}_T} \quad (41)$$

Similarly, at timestep (T - 1), we have

$$\frac{\partial L(T-1)}{\partial \mathbf{c}_{T-1}} = \frac{\partial L(T-1)}{\partial \mathbf{h}_{T-1}}\frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{c}_{T-1}} \quad (42)$$

However, the error is not only backpropagated via $L(T-1)$, but also from c_T. So, we have a final gradient of

$$\frac{\partial L(T-1)}{\partial \mathbf{c}_{T-1}} = \frac{\partial L(T-1)}{\partial \mathbf{c}_{T-1}} + \frac{\partial L(T)}{\partial \mathbf{c}_{T-1}}$$
$$= \frac{\partial L(T-1)}{\partial \mathbf{h}_{T-1}}\frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{c}_{T-1}} + \frac{\partial L(T)}{\partial \mathbf{h}_T}\frac{\partial \mathbf{h}_T}{\partial \mathbf{c}_T}\frac{\partial \mathbf{c}_T}{\partial \mathbf{c}_{T-1}} \quad (43)$$

With these results in hand, the model may be trained using standard gradient decent [9].

III. RESULTS AND ANALYSIS

A. Results

We varied our training over both the number of days in a given sequence (5, 10, 15, and 30), and over the learning rate of our model (0.01, 0.1, 0.2, and 0.5). Reserving 20% of our dataset for validation, our validation accuracies (Found in the following table) suggest that the optimal predictive framework (as an aggregate of the accuracies and RMSEs) of our model exists when we take slices of ten days, with a learning rate of 0.2.

		Sequence Length (days)			
		5	10	15	30
Learning Rate	0.01	Train Accuracy = 93.097% Train RMSE = 13.265 Test Accuracy = 94.019% Test RMSE = 28.732	Train Accuracy = 91.436% Train RMSE = 16.829 Test Accuracy = 92.009% Test RMSE = 38.743	Train Accuracy = 89.648% Train RMSE = 20.733 Test Accuracy = 81.125% Test RMSE = 57.066	Train Accuracy = 89.127% Train RMSE = 21.747 Test Accuracy = 93.679% Test RMSE = 29.967
	0.1	Train Accuracy = 98.020% Train RMSE = 3.496 Test Accuracy = 95.146% Test RMSE = 21.925	Train Accuracy = 97.479% Train RMSE = 4.574 Test Accuracy = 95.810% Test RMSE = 19.011	Train Accuracy = 96.621% Train RMSE = 6.337 Test Accuracy = 95.811% Test RMSE = 19.142	Train Accuracy = 93.429% Train RMSE = 13.239 Test Accuracy = 94.852% Test RMSE = 24.380
	0.2	Train Accuracy = 98.496% Train RMSE = 2.635 Test Accuracy = 95.169% Test RMSE = 21.960	Train Accuracy = 98.188% Train RMSE = 3.247 Test Accuracy = 95.672% Test RMSE = 19.559	Train Accuracy = 97.752% Train RMSE = 4.141 Test Accuracy = 95.616% Test RMSE = 19.821	Train Accuracy = 95.346% Train RMSE = 9.345 Test Accuracy = 94.872% Test RMSE = 23.558
	0.5	Train Accuracy = 98.940% Train RMSE = 1.847 Test Accuracy = 94.967% Test RMSE = 23.433	Train Accuracy = 98.773% Train RMSE = 2.171 Test Accuracy = 95.535% Test RMSE = 20.489	Train Accuracy = 98.608% Train RMSE = 2.526 Test Accuracy = 95.447% Test RMSE = 20.759	Train Accuracy = 96.565% Train RMSE = 7.013 Test Accuracy = 93.968% Test RMSE = 27.408

B. Analysis

This implies that when the sequence is too long, the model will find it difficult to determine what data is pertinent to its prediction. When the sequence given to the LSTM is too short, the model does not have enough history to predict future trends accurately. Therefore, a sweet spot must be found for the sequence length- one that has enough history of stock trends and just enough granularity to make accurate future predictions.

IV. Conclusions

In this project, we learned how LSTM is applied to RNN to improve time sequence predictions for stock prices and how the cell state process information between each epoch to produce more accurate predictions than a normal RNN. When adjusting the parameters of the RNN model with LSTM, we found that the sequence length of the input as well as the learning rate cannot be too low as it will cause underfitting, or too high as it will cause overfitting. What should be taken into account, is that the model tries to predict the results in a continuous manner, thus it is not reliable when predicting sequences that experienced sudden unpredictable change. Overall, given good parameters, an RNN model utilizing LSTM is a powerful tool to predict time sequence data.

V. References

[1] Nature, Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. Volume 323, Issue 6088, pp. 533-536 (1986)

[2] "John Hopfield." *Wikipedia*, Wikimedia Foundation, 23 Sept. 2019, en.wikipedia.org/wiki/John_Hopfield.

[3] Schmidhuber, Jürgen. Juergen Schmidhuber's Online Publications, Apr. 1996, people.idsia.ch/~juergen/onlinepub.html.

[4] "Sepp Hochreiter." *Wikipedia*, Wikimedia Foundation, 12 Oct. 2019, en.wikipedia.org/wiki/Sepp_Hochreiter.

[5] "Jürgen Schmidhuber." *Wikipedia*, Wikimedia Foundation, 13 Oct. 2019, en.wikipedia.org/wiki/J%C3%BCrgen_Schmidhuber.

[6] Neural Computation, Hochreiter, Sepp; Schmidhuber, Jürgen Volume 9, Issue 8, pp. 1735–1780 (1997-11-01)

[7] Zhou, Victor. "An Introduction to Recurrent Neural Networks for Beginners." *Victor Zhou*, Victor Zhou, 24 July 2019, victorzhou.com/blog/intro-to-rnns/.

[8] Thakur, Divyanshu. "LSTM and Its Equations." *Medium*, Medium, 6 July 2018, medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af.

[9] Chen, Gang. "A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation." *ArXiv.org*, 14 Jan. 2018, arxiv.org/abs/1610.02583.