Analysis of the LSTM to Update the Forget Gate to Improve Pattern Learning

Preston Robertson

Department of Industrial and Systems Engineering, Mississippi State University, Mississippi State, MS, 39762 USA

Abstract

The Long Short-Term Memory (LSTM) model is a neural network that specializes in time-series based data. The LSTM has long stood as the best model for time-series based data sets. The goal of this paper is to analyze how the LSTM behaves given different activation functions with the end goal of updating the forget gate to allow for pattern learning. Through the trials of the activation function, a custom activation function is introduced, the Squish function. After all the activation functions are tested, the results show that proper tuning of the Squish function could possibly yield better results than the base hyperbolic tangent functions. The results also showed no correlation between exploding and vanishing gradients having a significant impact on the accuracy of the LSTM model. However, the model is hyper sensitive to the change of the recurrent activation function.

Keywords: Long Short-Term Memory, Squish, Neural Network, Activation Function

1. Introduction

The problem of analyzing time-series based data has been a goal of researchers, since normal artificial neural network learning cannot be directly applied. The architecture of a standard neural network or a convolutional neural network (CNN) can not change or update as time goes on after training the model [1]. For example, the artificial neural network or CNN will handle image data but can not have a video as the input data because the CNN is unable vectorize video data. That is where the recurrent neural network (RNN) is implemented [2]. The architecture of the recurrent neural network allows for a feed-forward network that learns from results from previous nodes. The RNN models also cannot have a video format as input data; however, the model processes each image in the video data in such a way that the model still learns as if the images were still in video formats. This

Email address: `pgr41@msstate.edu` (Preston Robertson)

change to architecture allows for new types of data to be analyzed, such as video, speech, vibrations, robot control, and sign language translation [3].

The RNN model has a common issue of vanishing gradient as the number of epochs grows [4]. An epoch is the iteration of learning in a neural network. For example, the first pass of data would be the first epoch, the second pass would be the second epoch, etc. This vanishing gradient refers to the learning slowdown of the model and occurs due to the model's infinite node nature. For example, cell 1 will have a larger impact on the results of cell 2 than cell 43's results impact on cell 44. This is due to the amount of information being transferred, with the old information eventually overshadowing the new inputs. The Long Short-Term Memory (LSTM) model is an attempt to fix this issue through adding a variable named "cell state" [2]. The LSTM was initially made to revolutionize speech detection and is even used in popular services such as Apple's Siri. The LSTM helps fix the vanishing gradient problem by giving each cell the ability to forget old inputs. Before the discussion of how the LSTM model forgets, it is important to establish how the information is translated throughout each cell. Each cell has its own output (which is the objective of the artificial neural network) and a cell state that is a recorded value transferred through each cell [5]. This how each cell forgets data. Each cell takes the previous layer's output and the current cell state and compares them. The more these values do not match, the more the cell forgets its cell state. This process of updating these first weights and biases is called the forget gate [6].

The forget gate has had significant impact on the analysis of speech data [7], which is partially the reason for the increased accuracy of speech recognition in products such as Amazon's Alexa. However, the LSTM model is not yet perfect. The forget gate has issues itself. The cell state is a single value propagating through each cell [8]. When the cell state drops to a significantly low value in the forget gate, then essentially all old information is lost. This is an issue due to the pattern nature of data sets. For example, let's say the LSTM model is analyzing stock prices. If the stock price has a significant decline then model will forget the old information and adjust to the new information. However, a researcher may notice the pattern is a seasonal issue (such as the stocks of a swimming wear company). The LSTM model will never find this pattern and will have significant error when the pattern repeats. The second issue with related to the cell state being a single value, isthe vanishing gradient problem; the third cell will have a larger impact on the cell state value than the impact of the 45th cell in the LSTM model. To address these limitations, the objective of the paper is to further understand the effects of different parts of the LSTM model with the end goal of improving the LSTM model's forget gate.

2. Literature Review

The first LSTM model was proposed in 1997 as a solution to the RNN model's difficulty handling long term dependencies in data [9]. The model has found great success in speech recognition data, DBLSTM-HMM by Alex Graves at the University of Toronto [10]. The original LSTM model initially only added the cell state to the RNN model. This model architecture was very popular in the early 2000s before the invention of the forget gate. In 2000, the forget gate was added by Gers, Schmidhuber, and Cummins [11] in an effort to correct the oversaturation that forms in the model. Due to the inability to forget data, the cell state would not properly update to represent the new information. For example if an original LSTM model predicts where Jim eats lunch; the results would most likely be as followed. Jim went to his favorite restaurant for the last 5 years, but then it closed down for the winter season. Without the ability to forget, the LSTM model will continue predict Jim will eat at the restaurant despite that not being the correct output value. This is due to the oversaturation of the previous data which happens in several data sets [12] such as predicting stocks since there are outside factors that the model can not account for. These outside factors allow for error in the neural network. This new LSTM model gains the ability to selectively forget information through the architecture below in Figure 1.

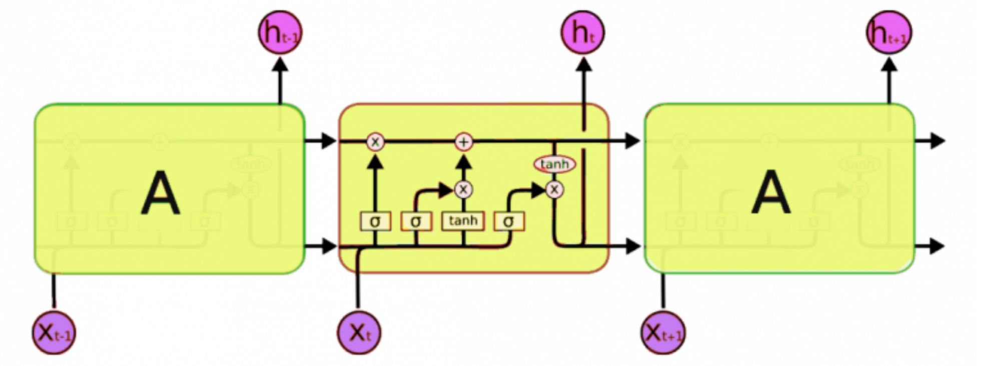

Figure 1: Architecture of LSTM Model [13]

The LSTM architecture shown above is how the overall LSTM function works. The model has an input and an output as a normal neural network would have. The input is denoted by X_t, and the output is denoted by h_t, where t represents the epoch (or time). Where X_{t-1} is the input of the previous LSTM cell. The denotation of the output gate is h_t instead of O_t due to the other function of this value as the hidden state. The hidden state is the short-term memory of the LSTM, and is how the LSTM remembers the recent output of the model. How the hidden state interacts with the LSTM model will be covered in the next section. The other line through the LSTM model is the cell state, or c_t. The cell state is how the model keeps a long term memory, and is changed through each iteration. The cell state keeps information from old iterations and is unaffected by the final output of the LSTM cell.

2.1. The Forget Gate

The forget gate is the first part of the process in the LSTM model (2). This is where the previous hidden state, h_{t-1}, and the current input, x_t, are used to determine how much of the previous cell state, c_{t-1}, is kept (1).

$$f_t = \sigma(W_f \times x_t + U_f \times h_{t-1} + b_f) \tag{1}$$

This is done by multiplying the hidden state and the input by trainable parameters known as weights. Then a bias is added on top of these parameters before going through a sigmoid activation function (2).

$$\sigma = \frac{1}{1 + e^{-x}} \tag{2}$$

This sigmoid activation function can be changed to other activation functions but the gradient of the sigmoid is desirable since it keeps values between 0 and 1. The position held by the sigmoid activation function is known as the recurrent activation function. This acts a percentage scaling value; for example, if we want to keep 60% of the cell state, then the weights and bias will be tuned to output 0.60 which will effectively only keep 60% of the cell state's information. Equation (6) shows how the forget gate mathematically accomplishes this. Figure 2 shows how the forget gate fits into the overall model referenced in Figure 1, the forget gate is referred in the graphic and mathematical models as f_t.

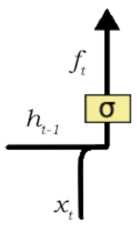

Figure 2: Architecture of the Forget Gate [13]

2.2. The Input Gate

The input gate, the second gate of the LSTM model, is where the new proposed cell state is introduced to the equation. This determines how much the current cell state should change to meet current requirements of the LSTM model. The input gate is a specific value (3) in the LSTM model but all functions in this subsection fall under what is considered the input gate.

$$i_t = \sigma(W_i \times x_t + U_i \times h_{t-1} + b_i) \tag{3}$$

The input gate value is what determines how valuable the new change is to the overall model accuracy. The input gate, much like the forget gate, defaults to using the sigmoid function as the recurrent activation function. The other value calculated in the input gate step is the suggested change to the cell state. This is determined through using the set activation function on the input values of this step (4).

$$c'_t = \tanh(W_c \times x_t + U_c \times h_{t-1} + b_c) \tag{4}$$

The default activation function is typically the hyperbolic tangent function (5). Since it is a bounded function that goes from -1 to 1 instead of the sigmoid function only going from 0 to 1. This ability to have a negative cell state has shown to benefit the LSTM model. The negative cell state effectively lets RNN change the direction of cell state. This is because the suggested cell state more shows the direction the cell state should go rather than a specific value.

$$\tanh = \frac{e^2x - 1}{e^2x + 1} \tag{5}$$

Now that the suggested changes to the cell state have been gathered through the input gate and the amount of information needed to be forgotten has also been collected it is time to combine the new information to update the cell state to its new value. We do this by multiplying the previous cell state by the forget gate and the suggested cell state by the input gate. The equation is shown below (6).

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t \tag{6}$$

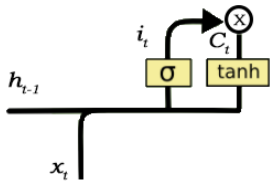

Figure 3: Architecture of the Input Gate [13]

2.3. The Output Gate

The output gate is where the output of the LSTM cell is decided. This process takes a similar approach to the methods above by combining the hidden state with the current input through a set of weights and biases then putting that value through the recurrent activation function (7). This value is recorded as the output gate.

$$o_t = \sigma(W_o \times x_t + U_o \times h_{t-1} + b_o) \tag{7}$$

The final output is then found by multiplying this o_t by the cell state that is filtered through the set activation function (8), leaving the hidden state of the current cell (this value is also referred to as the output). This cell is now complete, and the new hidden state (h_t) and cell state (c_t) are then used as the input of the next LSTM cell where the process begins anew. Figure 4 shows visually how the output gate fits into the LSTM model.

$$h_t = o_t \cdot \tanh(c_t) \tag{8}$$

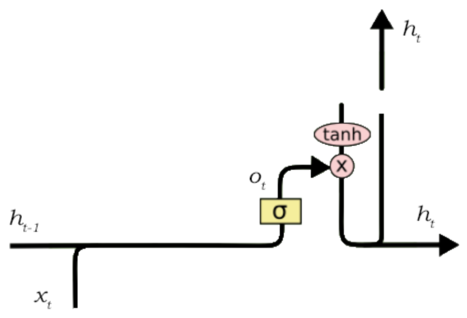

Figure 4: Architecture of the Output Gate [13]

3. Methodology

For the analysis conducted in this paper, we will see the different ways that the activation functions can be changed, and how how these changes affect the outcome. The changes will be made to both the main activation function and to the recurrent activation functions, and should have drastic impacts on the accuracy of the models since some of the activation functions tested will be unbounded rather than original bounded functions. This experiment will simply test the accuracy and convergence rates of the model to see the activation functions performance in the given scenario. According to the literature, the answer has already been solved with the current activation functions, the best being hyperbolic tangent (5) as the main activation function and the sigmoid function (2) as the recurrent activation function. Therefore the objective of this study activation functions affects accuracy, and not determining which is the best. With this in mind it will be more important to discuss the gradient of the activation functions tested. The inclusion of a custom activation function known as the Squish will also be included. As of the writing of this paper, the Squish function has not yet been published so it will be briefly discussed ahead. Different activation functions will be tested at both the main and recurrent to see the effect on the test accuracy. The activation functions are shown below, with the ReLU (9), Swish (10), and SoftMax (11) functions chosen to also be compared.

$$ReLU = max(x, 0) \tag{9}$$

$$Swish = \frac{x}{1 + e^x} \tag{10}$$

$$SoftMax = \frac{x}{1 + |x|} \tag{11}$$

The ReLU function was chosen due to its overwhelming popularity in most neural networks, and its exploding gradient. The LSTM model's reaction to an exploding gradient is important to see in this experiment. Another function with an exploding gradient, the Swish was chosen due to the negative half of the function having a valuable trait. The comparison between the ReLU Function and the Swish function will show how important the "swish" gradient in the Swish function is in the LSTM. The Swish was also chosen since it is part of the Squish function, and comparing them will show how important the differences are. Finally the SoftMax function was chosen due to its vanishing gradient and it being part of the Squish function. If the SoftMax performs the best then it will show the importance of vanishing gradients and the need for simple activation function design within the architecture of the LSTM model.

The important takeaway from this experiment is how the LSTM acts with different kinds of input values at each gate in the cell. This is why the experiment will focus on changing the activation functions rather than changing the data set. The data set chosen to run the experiment is the MNIST data set, a commonly used data set for experiments in neural networks. The MNIST is an image based data set that is roughly 60,000 hand drawn images of numbers. It is a classification data set trying where the neural network's objective is to determine the number depicted in the image.

The LSTM model used in this experiment will have architecture as follows, two LSTM cell layers stacked followed by a dense net classification layer. Each of the LSTM cell layers will have the activation functions changed, while all other factors will remain the same. The images will be fed through the LSTM model 28 pixels at a time to turn the MNIST into a time-based data set and will have a mini-batch size of 64 pixels. Each test will run for 10 epochs and then the final loss and accuracy will be recorded and compared between each model.

3.1. The Squish Function

The Squish function (12) is a combination function with the objective of capturing properties from several different activation functions.

$$f(x) = min\left(\frac{x}{1+e^x}, \frac{p \times x}{1+e^x}\right) + max\left(\frac{x}{1-|x|}, 0\right) \tag{12}$$

The combination can be adjusted through this p value, where p is a percentage inclusion of the specified function. In this usage of the Squish function, the Swish function will be used in for all negative inputs. It has been determined by through research of literature, that the shape of the gradient for the negative side of the Swish function is a highly desirable trait ([14]). The exploding gradient also works well since it allows the p value to give a percentage want of that exploding gradient to be applied to a separate vanishing gradient. To achieve the vanishing gradient, the soft max function will be used. Since both functions cross through zero allowing for a continuous gradient. Figure 5 shows the different gradients of the Squish function given specific p value. For the purposes of this experiment, the p value will be set to 0 since it is already known that bounded functions perform the best. For future experiments outside the RNN, the p value can adjusted to find the best solution, but due to time restriction and the Squish function taking on average 38x longer per epoch, we will only run the Squish value once.

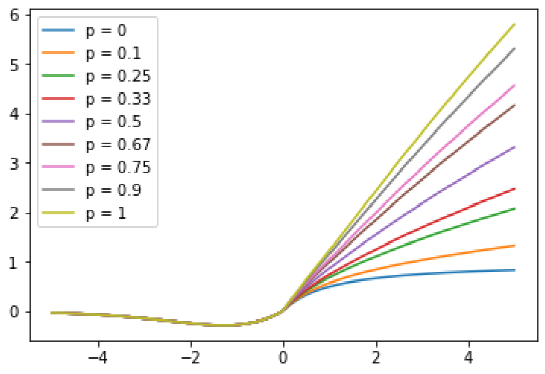

Figure 5: Gradients of the Squish Function

4. Experimental Results

After the experiment was conducted, the best performing duo activation functions was the base model with some close contenders. The results of the experiment are found in Table 4). The Squish and Swish in the main were top contenders for beating out the base model, it seems as the negative component in the main activation function can be of value. The Swish/Squish's derivative change in the negative side can also prevent learning slowdown, and this can be seen through both of their quick convergence rate and rapid approach to high accuracy scores as shown in Figure 6 and Figure 7. The next noteworthy result is that the SoftMax function failed, possibly due to error in calculations or function calling; however the Swish function also failed in the recurrent activation function position. The negative values from the Swish and SoftMax functions could of possibly led to the convergence failures. Especially since both activation functions can experience major learning slowdown in the negative component as well. The next major notable result is that the exploding gradient performed almost as well as the vanishing gradient activation functions. This shows that adjusting/tuning the Squish function will most likely lead to great success.

Table 1: Results of Experiment

Activation Function	Location	Loss	Accuracy
Base Model	N/A	0.03406	0.9911
Squish	Main	0.0379	0.9986
Squish	Recurrent	0.0389	0.9897
ReLU	Main	0.0420	0.9872
ReLU	Recurrent	0.1710	0.9424
Swish	Main	0.0365	0.9882
Swish	Recurrent	1.5770	0.4990
SoftMax	Main	2.3009	0.1134
SoftMax	Recurrent	2.3010	0.1135

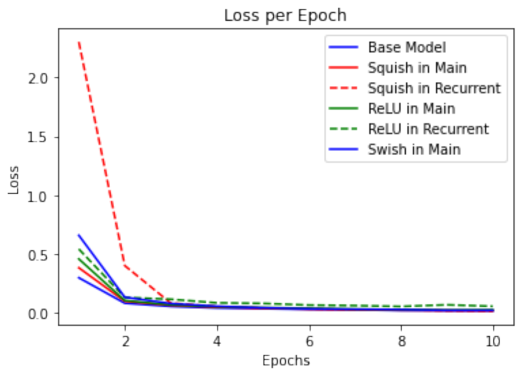

Figure 6: The Loss per Epoch without the Convergence Failures

5. Conclusion

Overall, after learning about the forget gate and its creation, it has seemed to be missing a variable. This look into the architecture and design of the LSTM has shown that there

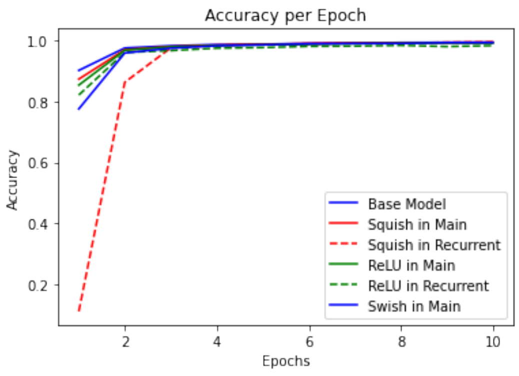

Figure 7: The Accuracy per Epoch without the Convergence Failures

are still places to improve the LSTM model. The Squish function proved to be a useful tool and if properly tuned could show better results than the standard for the LSTM. This study has also shown that in future work for improving the architecture of the forget gate it is crucial to run that value through the recurrent activation function since non-percentage based values seem to lead to a high chance of convergence failure. Overall the LSTM did not behave as expected with the changes from vanishing to exploding gradient and non-negatives to negatives in gradient showing high accuracy results. The methodology was built around showing major differences in results, that simply did not occur in this experiment and can be improved upon. This experiment can be improved by having a higher amount of epochs and data sets to further investigate the differences in results. The different activation functions showed strange properties of the LSTM that will require future work to truly understand why the vanishing and exploding gradients perform at the same level despite the LSTM model's use of the activation function as a percentage. These results could have been a product of the low amount of epochs trained not allowing for the LSTM model's cell state to "explode" or maybe that is not an issue due to the forget gate. Overall, this experiment shows surprising results that should be evaluated.

References

[1] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 806–813.

[2] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[3] H. Xiao, M. A. Sotelo, Y. Ma, B. Cao, Y. Zhou, Y. Xu, R. Wang, and Z. Li, "An improved lstm model for behavior recognition of intelligent vehicles," *IEEE Access*, vol. 8, pp. 101 514–101 527, 2020.

[4] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," *Advances in neural information processing systems*, vol. 9, 1996.

[5] F. Qian and X. Chen, "Stock prediction based on lstm under different stability," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, 2019, pp. 483–486.

[6] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[7] C. Zhou, C. Sun, Z. Liu, and F. Lau, "A c-lstm neural network for text classification," *arXiv preprint arXiv:1511.08630*, 2015.

[8] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[9] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[10] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE, 2013, pp. 273–278.

[11] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[12] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[13] N. Pranj52, "Long short term memory: Architecture of lstm," May 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

[14] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.