Advanced Analytic Techniques 1

John Boland

Project:

Long-Short Term Memory

Mahmoud Ghazi

November 2022

Table of Contents

Introduction

Recurrent Neural Networks (RNNs) are powerful machine learning models and have been used in many areas. RNNs are generally helpful in working with sequence prediction problems. Sequence prediction problems come in many forms and are best described by the types of inputs and outputs it supports.

RNNs-based models have received the most success when working with sequences of words and paragraphs, generally in natural language processing (NLP). They are also used as productive models that produce a sequence output. Recurrent Neural Networks are one of the most prevalent architectures because of their ability to handle variable-length texts.

Why do we need Long Short-Term Memory (LSTM)?

When you read a text (Sivalingam 2020), you comprehend each word based on your understanding of earlier words. You don't throw everything away and begin thinking from scratch again. Your thoughts have persistence.

Traditional neural networks can't do this, where we need to know about the previous data to predict the subsequent output. For example, to expect the next word of a sentence, we should understand its last words. In such procedures, our typical neuron won't help as it doesn't know the previous words.

Recurrent neural networks address this issue. They are networks with loops in them like figure 1, allowing information to persist. When data is dynamic and sequentially managed, such as text or stock market values, a variant called Recurrent Neural Network is employed.

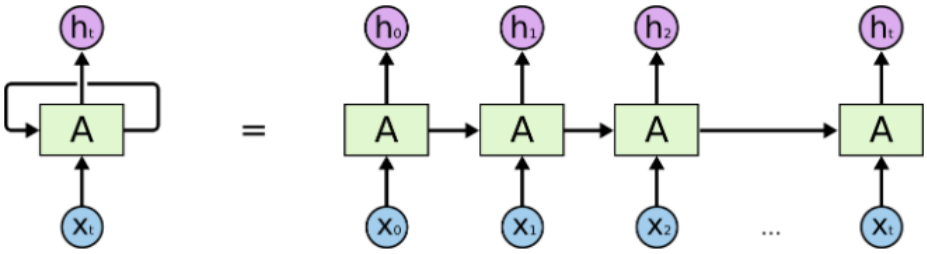


Figure 1- Recurrent Neural Network

Occasionally, we only need to look at recent information to perform the present task. For instance, consider a language model trying to predict the next word founded on the previous

ones. In “the clouds are in the *sky*,” If we are trying to predict the last word, we don’t need any further context.

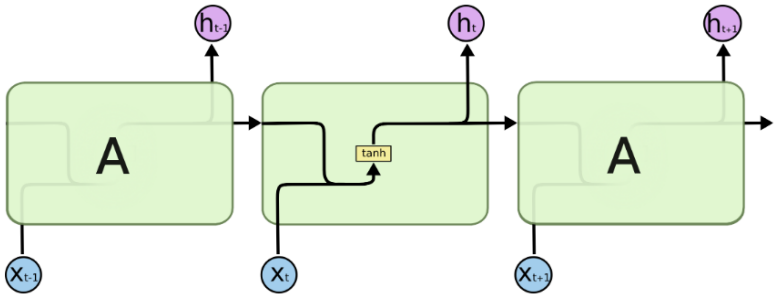


Figure 2- RNN with activation function

But predicting the last word in the text, " I grew up in France… I speak fluent *French*." is a bit challenging. Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France from further back. Unfortunately, RNNs cannot learn to connect the information as that gap grows.

RNN can control data reliance but within short time gaps. This is achieved by feeding the hidden layer's output into the hidden layer's input (figure 2). However, due to the vanishing gradient issue, RNNs aren't entirely practical with time-sensitive data.

In the propagation step, recurrent neural networks encounter the vanishing gradient problem. Gradients are values used to update a neural network's weight. If a gradient value gets extremely small, it doesn't contribute considerably to learning. Figure 4 shows an example that small values in the update phase of backpropagation cannot contribute to updating.

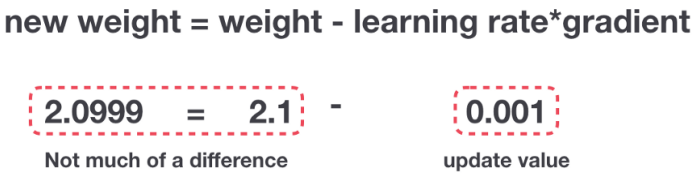


Figure 3- update the weights

This will result in the initial layers not being optimized. The model's overall performance will be degraded since the last layers depend on the low-level features seized by the initial layer. The amount of data that can be stored in the memory significantly decreases when we encounter vanishing gradient issues. Long Short-Term Memory as the subset of RNNs is proposed as a solution. LSTM can store processed data about the longer sequence of data.

LSTM Architecture

LSTM networks (figure 5) are a type of RNN that includes a 'memory cell' that maintains information in memory for long periods (Hochreiter & Schmidhuber 1997). A set of gates

controls when information enters the memory when it's output, and when it's forgotten. This architecture lets them learn longer-term dependencies. A unique feature called Forget Gate is employed to eliminate the vanishing (or exploding) gradient problem. This helps greatly reduce the multiplicative effect of small gradients. Hidden-State is the actual output of that node for a given input. But always hidden because it only enters as input at the next time-step.

Four components:

1. Memory cell (cell state)
2. Forget gate
3. Input gate
4. Output gate

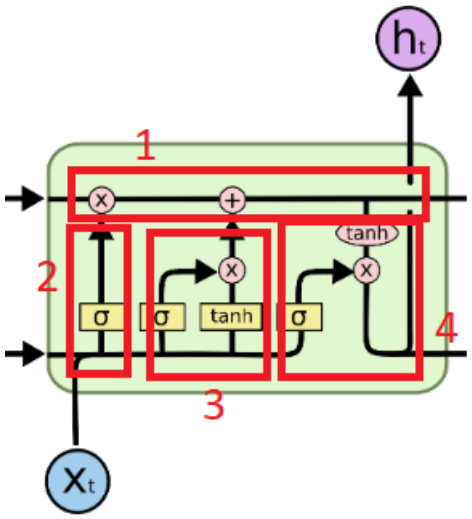


Figure 4- LSTM architecture

Memory cell

The horizontal line noted by Ct (figure 6) is the core concept of LSTM (Devopedia 2019). The memory cell is used for remembering or forgetting things. How it remembers and forgets is based on the context of the input. The cell state is able to carry relevant information throughout the processing of the series. So even data from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory.

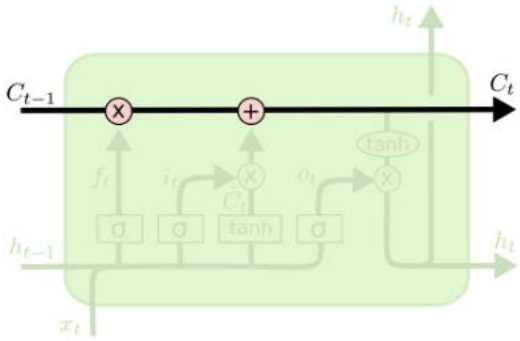
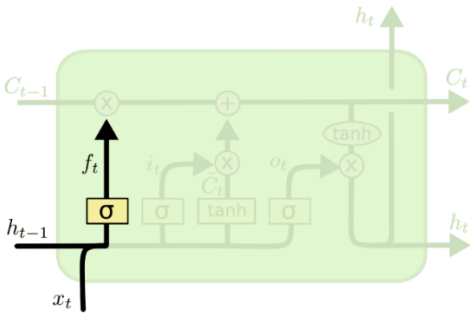


Figure 5- Memory cell

There are three types of gates in an LSTM cell that decide whether the cell state should be modified. The gates are different neural networks that determine what data is permitted in the cell state. Each one of them will determine what portion of the older data has to be thrown away, what amount of newer data has to be remembered, and what part of the memory has to be given out correspondingly.

Forget gate

The forget gate will give an output based on the input feature and the hidden state. First, weights initialize for the input of ht-1 and Xt. These weights getting concatenated along with the input. The product of concatenation will pass through the sigmoid activation function. The sigmoid will give an output range from 0 to 1. The output of the forget gate will be multiplied by the cell state. The cell will forget everything it has learned so far if the output is 0 and will remember everything if the output is 1. Based on the output value of the forget gate, the cell state will change.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

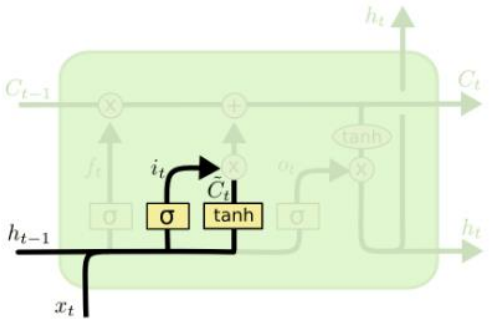
Figure 6- forget gate

For example, when ht-1 is the word "QUEEN" and xt is "KING," which are similar, after concatenation and sigmoid function, we have a vector that most of which are ones. If two words are different, the vector is filled chiefly with zero. Then this vector passes to a pointwise operation with forget or not forget part of the data.

Input gate

Next, we have to choose what part of the newer data is to be stored in the memory. The input gate in figure 8 comes into play here; similar to the forget gate, the hidden state of the previous

time step and the input feature is given to the input gate. Then Tanh function creates a new set of values between -1 and 1 to be kept in the memory.

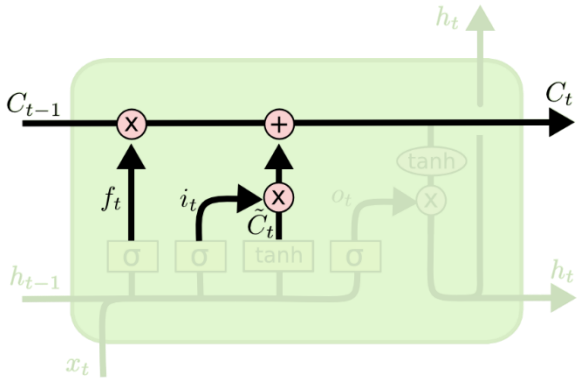


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 7- input gate

Update cell state

It's time to update the old cell state, Ct−1, into the new Ct. The previous steps already decided what to do. We just need to actually do it as figure 9 shows by applying pointwise operators.

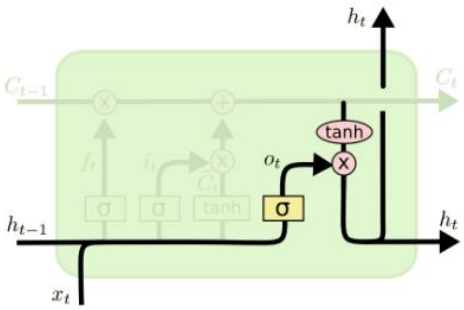


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 8- update the Ct

Output gate

Finally, we need to settle what we're going to output. The output will be founded on the cell state but will be filtered. First, a sigmoid layer is conducted to decide which part of the cell state takes to output. Then, we put the cell state through Tanh to confine the value and multiply it by the result of the sigmoid gate so that we only output the required parts of data.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 9- Output gate

Implementation Examples of usage

Manual analysis of large quantities of such data is challenging, so a likely need for computer processing has emerged. Sentiment analysis processes people's comments toward products or services and identifies the emotional tone behind them. This is a popular way for organizations to define and classify opinions about a product, service, or idea.

In an experimental study by (Murthy et al. 2020) they used the IMDB and Amazon Product datasets. IMDB is a large movie review dataset and is a benchmark for movie reviews. Their work aims to identify the polarity of the given review, that is, whether the review is of positive or negative sentiment. In my work, I do the same sentiment analysis but with a different text mining approach on just IMDB's dataset and compare my result with the published paper.

I am using IMDB movies review and divide it into train and validate set to evaluate the model. This dataset contains 25,000 comments/tweets, of which 12500 are positively polarized, and 12500 are negatively polarized. As the flowchart in figure 11 shows, we need to apply text analysis on the raw text to turn it into a proper embedding vector for the lstm model.

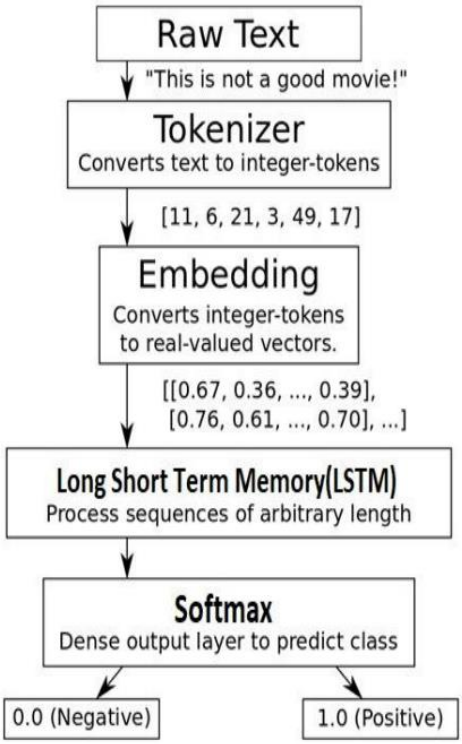


Figure 10- implementation flowchart

First, we need to make lists of reviews by separating out individual reviews. I applied tokenization to split strings into a list of words and remove unnecessary tokens and punctuation. Furthermore, words that rarely appear in the text were filtered. Next, after mapping each word with the index, we should convert each word to a vector of numbers named the embedding layer. In the embedding layer, a neural network is employed to learn a unique vector with a size of 100.

The Lstm model takes the input as a vector, assigns parameters, and gives parameters to the softmax function, which is good at determining multi-class probabilities to assign a class label (positive or negative). LSTM network and softmax generate an output at each time step, and this output is used to train the network using an adam optimization approach.

As the output in figure 12 shows, the training set accuracy in my implementation by increasing epochs get improved, and as a result, the testing set accuracy gets better. Compared with the result of the article in table 13, my work does not show good accuracy. My model's accuracy

can be improved by applying regularization echniques which will be employed in future works.

```
[Epoch:  1/10] | Training Loss: 0.010 | Testing Loss: 0.008 | Training Acc: 75.25 | Testing Acc: 82.03
[Epoch:  2/10] | Training Loss: 0.007 | Testing Loss: 0.007 | Training Acc: 85.78 | Testing Acc: 84.34
[Epoch:  3/10] | Training Loss: 0.005 | Testing Loss: 0.008 | Training Acc: 89.56 | Testing Acc: 85.02
[Epoch:  4/10] | Training Loss: 0.004 | Testing Loss: 0.007 | Training Acc: 92.84 | Testing Acc: 85.07
[Epoch:  5/10] | Training Loss: 0.002 | Testing Loss: 0.009 | Training Acc: 95.50 | Testing Acc: 85.40
[Epoch:  6/10] | Training Loss: 0.002 | Testing Loss: 0.012 | Training Acc: 97.45 | Testing Acc: 84.77
[Epoch:  7/10] | Training Loss: 0.001 | Testing Loss: 0.016 | Training Acc: 98.72 | Testing Acc: 84.50
[Epoch:  8/10] | Training Loss: 0.000 | Testing Loss: 0.020 | Training Acc: 99.25 | Testing Acc: 84.55
```

Figure 11- output of my implementation

Configuration of the model	Epochs	LSTM Units	Accuracy
Embedding Layer + LSTM Layer + Dense Layer	1	128	56.14%
	2	128	75.91%
	3	128	88.13%
	4	128	93.44%
	5	128	96.16%
	6	128	97.67%
	7	128	98.59%
	8	128	99.11%
	9	128	99.36%
	10	128	99.56%

Figure 12- Output of the article implementation

Improvement

Here I introduce to approach to improve the lstm model.

1- Grid LSTM

Grid LSTM (Kalchbrenner, Danihelka & Graves 2015) is a network that is arranged in a grid of one or more dimensions. The Grid LSTM network has LSTM cells along the dimensions of the grid. The deepness is treated like the other dimensions and uses LSTM cells to speak directly from one layer to the subsequent layer.

Grid LSTM network uses LSTM cells along all dimensions and modulates the multi-way interaction in a novel style. The cells have advantages compared to regular connections in

solving tasks such as parity, addition, and memorization. It is considered a robust and adjustable way of using the model for character prediction, translation, and image classification, showing strong performance.

2- QUASI-RECURRENT NEURAL NETWORKS (QRNNs)

Lstm networks are a strong tool for modeling sequential data. Still, the reliance of each timestep's computation on the prior result limits parallelism and makes Lstms bulky for very long sequences. QRNNs (Bradbury et al. 2016) are an approach to sequence modeling that alternates convolutional layers. It applies parallel across timesteps and a minimalist pooling function. Furthermore, stacked QRNNs have better predictive accuracy than stacked LSTMs of the same hidden size. They are up to 16 times faster due to their increased parallelism at train and test times. Investigations on language modeling, sentiment analysis, and other machine translation indicate QRNNs outperform other sequence task models due to a fundamental building block.

Conclusion

Long Short-Term Memory (Hochreiter & Schmidhuber 1997) networks are recurrent neural networks prepared with a precise gating mechanism that governs entry to memory cells. Since the gates can control the rest of the network by changing the contents of the memory cells for multiple time steps, LSTM networks maintain signals and propagate errors for more extended than standard recurrent neural networks. The gates can also learn to attend to distinctive parts of the input signals and overlook other factors by independently reading, writing, and erasing content from the memory cells.

These properties allow LSTM networks to process data with a sequence such as text. In this work, I implemented a sentiment classifier by applying the lstm model for text data gathered by IMDB. This data includes the opinion of people worldwide on different topics. The lstm model shows 99% accuracy on training data and 85% on test data which can be improved by employing regularization methods. Furthermore, Grid lstm and QRNN are two approaches that can enhance the analysis of sequenced data like text.

References

Bradbury, J, Merity, S, Xiong, C & Socher, R 2016, 'Quasi-recurrent neural networks', *arXiv preprint arXiv:1611.01576*.

Devopedia 2019, *LSTM*, <https://devopedia.org/long-short-term-memory>.

Hochreiter, S & Schmidhuber, J 1997, 'Long short-term memory', *Neural computation*, vol. 9, no. 8, pp. 1735-1780.

Kalchbrenner, N, Danihelka, I & Graves, A 2015, 'Grid long short-term memory', *arXiv preprint arXiv:1507.01526*.

Murthy, G, Allu, SR, Andhavarapu, B, Bagadi, M & Belusonti, M 2020, 'Text based sentiment analysis using LSTM', *Int. J. Eng. Res. Tech. Res*, vol. 9, no. 05.

Sivalingam, A 2020, *why do we need lstm*, <https://towardsdatascience.com/why-do-we-need-lstm-a343836ec4bc>.