CS 475 Intro to Machine Learning: Long Short-Term Memory Project Report

Charlie Wang
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA
TODO: fillin...@jhu.edu

Gilbert Maystre
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA
TODO: fillin...@jhu.edu

Abstract

TODO: This document contains the instructions for preparing a camera-ready manuscript for the proceedings of NAACL HLT 2010. The document itself conforms to its own specifications, and is therefore an example of what your manuscript should look like. Authors are asked to conform to all the directions reported in this document.

1 Introduction

Long Short-Term Memory (LSTM) is an artificial neural network architecture, which is in turn considered as a machine learning model. First proposed by (Hochreiter and Schmidhuber, 1997), LSTMs are a subclass of recurrent neural networks, a subclass of artificial neural networks. LSTMs build upon the decades of artificial neural network research to solve new types of problems.

The artificial neural network architecture is based on the idea of biological neurons in the brain and how neurons are organized. It was discovered that neurons receive multiple inputs from other neurons, does something simple with the inputs, and outputs a signal which can be relayed to many other neurons. In the same idea, the nodes in artificial neural networks replicate the biological neurons. Artificial neural network nodes receive multiple inputs from other nodes, which are weighted. A summation is performed on the inputs, and this sum is then passed through an activation function. Many times, the activation function is a simple logistic sigmoid function. Then, the node would output the value to other nodes through weighted connections. Although each neuron is seemingly simple of-by-themselves, together, they can solve many complex problems, including the estimation of a regression surface by (Specht, 1991). The artificial neural networks were generally organized with an input layer, which consisted of many input nodes with an identity activation function. The nodes in the input layer would be connected to the nodes in the hidden layer, where the nodes in the hidden layer would in turn be connected to the nodes in the output layer. This high-level overview is the general architecture of artificial neural networks.

Much work was done on artificial neural networks, such as adding many hidden layers to form deep learning neural networks. Another work that arose in the artificial neural network world was with recurrent neural networks. Recurrent neural networks are similar to a traditional artificial neural network architecture of input layer, hidden layer, and output layer. However, the main difference is that the weights coming out of a node does not just go to other nodes in other layers – the weights can loop back to the node itself or other nodes in that respective layer. This new idea became the recurrent neural network (Elman, 1990).

Recurrent neural networks were able to extend upon artificial neural networks, but more was wanted from what recurrent neural networks were able to perform and achieve. This led to the birth of architecture of Long Short-Term Memory (LSTM), which is built on recurrent neural networks, which is in turn built on artificial neural networks in general. Based on the work of (Hochreiter and Schmidhuber, 1997), greater complexity was added to the nodes in the hidden layer of the neural network. To start, the idea of a hidden layer node was extended into the idea of a memory block. For each memory block, it consists of the node itself, which is renamed as a cell, and the novel idea of gates. Each memory block had two gates, an input gate and an output gate. The gates were considered multiplicative, and the output of the gates are multiplied with the respective weighted values coming from and to the cell in the memory block. To clarify, the output of the input gate is multiplied with the summed input weights to the memory block. The output of the output gate is multiplied with the output value of the memory block. The goal of the gates is to control the constant error flow through discrete time steps due to the effect of the loops in the LSTM. Thus, the need for an activation function for the cell was discarded, and instead, an edge that flows back into the input of the cell was added. The value of the cell was named cell state.

Recall that the LSTM proposed by (Hochreiter and Schmidhuber, 1997) is built on the idea of recurrent neural network. Thus, the general overview of the connections between nodes can be illustrated in this way: each input node output is connected/flows to each of the memory block, each gate, and each output node. The output of the memory blocks is connected to the memory block itself (the recurrent part), the output nodes, and the gates.

More research was conducted on the LSTM architecture, and (Gers et al., 1999) discovered that there was a flaw with the cell. In fact, due to the loop from the output of the cell to the input of the cell on each time step, (Gers et al., 1999) discovered that the loop could cause the cell state to grow indefinitely, thus causing the whole network to be skewed so much that the output of the network is of not much use. In other words, the whole network becomes broken after too many time steps. Thus, (Gers et al., 1999) added onto the complexity of the original LSTM architecture by adding an additional gate, which they named the forget gate. The forget gate is similar to the input and output gates, but instead of targeting the input to the cell like the input gate or the output of the cell like the output gate, the forget gate targeted that recurrent loop of the cell. Similarly, as with the input gate and the output gate, they received weighted inputs from the input nodes and the memory block itself. Thus, the number of weights in the network grew and the complexity of the memory block grew, but (Gers et al., 1999) were able to show that this improved the network results in the long term.

However, there was still a flaw present in the LSTM architecture. The LSTM architecture after the work of (Gers et al., 1999) was ill-suited for tasks where accurate measurement of time intervals was needed. For example, the distinction is needed in some tasks to know when spikes or other abnormalities occur either 100 times steps apart, or 101 time steps apart. Thus, (Gers et al., 2002) proposed the idea of adding peephole weighted connection from the cell to the input, forget, and output gates. Without the peephole connections, the gates would only be able to see what the output of the cell is after being multiplied by the output of the output gate. If the output gate is closed, in other words close to 0, the gates would not be receiving an accurate picture of what the cell state is giving as an output. Thus, with the peephole connection which serve as a direction connection for gates to see what the cell state is currently like, network performance was shown to increase by (Gers et al., 2002).

These advancements, including many other following advancements, have made the LSTM architecture applicable to many problems out in the world today. For example, Google was able to achieve breakthrough in generating image captions with the help of the LSTM architecture combined with a deep convolutional net, which is described in (Vinyals et al., 2014). In fact, in this specific example, natural language captions were able to be generated on images by recognizing objects in the picture, such as what was the main subject and what the background is like.

The capabilities, complexities, and potential of LSTM is why the project was chosen to be done on the LSTM architecture and on the application of a LSTM network. For the project, an application of LSTM was chosen that incorporates some of the important advancements with the architecture, such as by (Gers et al., 1999) and by (Gers et al., 2002). Thus, the project is conducted on the works of (Gers and Schmidhuber, 2001), which is on the topic of learning simple context-sensitive languages (CSL).

2 Problem

The research project is based on replicating the solution obtained by (Gers and Schmidhuber, 2001) to the problem that their research is looking to solve. Based on (Gers and Schmidhuber, 2001), the problem that the paper looks to solve is building a network that is able to accurate classify a context-free language (CFL) and a context-sensitive language (CSL). Recall that CSL includes all CFL, and CFL includes all regular language (RL). According to (Gers and Schmidhuber, 2001), it has been shown that LSTM is able to outperform recurrent neural networks when working with RL, but they are unsure if this performance gap exists also when working with CFL and CSL though they know that recognizing CFLs require a stack which the LSTM would need to be able to replicate.

(Gers and Schmidhuber, 2001) set up the problem for the LSTM architecture in this way: a string belonging to the language will be presented to the LSTM. The string is formatted in such a way that it has a unique start symbol S and end of string symbol T. For example, for the CFL language $a^n b^n$, a string example that can be presented to the LSTM is $SaaabbbT$ for $n = 3$. Furthermore, the string will be presented to the LSTM in such a way that only 1 character of the string is presented at a time until all characters (the whole string) is presented.

Thus, the goal of the LSTM is to recognize what character is presented to it and predict all the next possible characters that can follow the character just presented to it. For example, with the $SaaabbbT$ example from before, when the LSTM is presented with the character S, the LSTM needs to be able to predict that the characters that can follow are T and a. When the LSTM is presented with the character a, the LSTM needs to be able to predict that only the characters a and b can follow. When the LSTM is presented with the character b, the network needs to be able to predict that the only characters that can follow is b except for the last b character presented to the network, which the LSTM needs to predict that only T can follow.

After the whole string is presented to the LSTM, the network, during testing, needs to make a classification decision of whether the string that it was presented with belongs within the language or not. The criteria for classifying is that if all predictions by the LSTM were correct, then the string belongs in the language. If one or more predictions by the LSTM were made on the string, then the string does not belong in the language.

When one encounters a traditional classification problem, such as whether a string belongs in the language or not, one might expect that two sets of data need to be presented to the artificial neural network: one set consisting of strings that are in the language and another set consisting of strings that are not apart of the language. In this problem, only strings that are part of the language is presented to the LSTM network. The set of strings that are not part of the language is not needed to be presented to the LSTM network to help it learn because the LSTM network is actually learning to predict characters and "fit" the language. Presenting strings that are not apart of the language could actually skew the learning process. Thus, the set of strings that are not part of the language is only needed during testing to see if the LSTM network can classify correctly.

3 Network Architecture and Forward Pass

The details of the network is presented with minimal detail in the paper, just enough that someone with existing LSTM knowledge can have an idea of what the architecture looks like. For example, (Gers and Schmidhuber, 2001) says that for the language $a^n b^n$ in particular, there are 38 adjustable weights. The only clarification available of those 38 weights is the text that follows in parentheses: 3 peephole, 28 unit-to-unit, and 7 bias. Figuring out exactly what all these 38 weights meant, especially what exactly are connected with respect to the 28 unit-to-unit, required sitting in front of a whiteboard and attempting to draw out the network with the weights. Thus, extensive time was placed into understanding what the network architecture looked like such that understanding of how the network operates and coding of the network could be achieved. The following is of our best understanding of the network architecture.

The LSTM architecture consists of three layers: an input layer, a hidden layer, and an output layer. In the input layer, the number of nodes corresponds to the language that is presented to the network. In fact, the number of nodes that is needed to however many different characters are in the language plus an additional node for recognizing the S and T. For example, with the $a^n b^n$ language, there will be one node for the a, one node for the b and one node for the S/T for a total of three nodes in the input layer. This is done in such a way so that the LSTM network can recognize which character from the string is presented to the network. Similarly, since the LSTM network is predicting what the next character will be, the number of nodes in the output layer would be the same as the number of nodes in the input layer.

The hidden layer consists of memory block units. Each memory block unit has one input gate, one forget gate, and one output gate. Although it is possible for a memory block to have multiple cells, (Gers and Schmidhuber, 2001) stated that their networks have only one cell per memory block. This decision was reached possibly after testing and to better see how the network is able to predict since there are less variables involved. For each language, though it is not stated by (Gers and Schmidhuber, 2001), we theorize that they chose the number of memory block units needed based on how the stack(s) would operate for recognizing the language. Thus, the number of memory block units that are needed is likely based on background knowledge of the language itself. Thus, for the language $a^n b^n$, only one memory block is in the hidden layer. However, for the language $a^n b^m B^m A^n$, two memory blocks are in the hidden layer. Additionally, peepholes exist from the input gate, forget gate, and output gate to the cell for each respective memory block.

To better understand exactly what is happening in each layer, more details will be clarified layer by layer. Starting with the input layer, the encoded vector of the character from the string is presented to the nodes in the input layer. A +1 value for the element of the vector indicates that particular character being present. A -1 value indicates that it is not that character. For example, with the language $a^n b^n$, when the a character is presented, the encoded vector of [S/T, a, b] would be [-1, 1, -1].

The nodes in the input layer take the respective input (element from the encoded vector), run it through an activation function consisting of the identity function, and output it. The output from each input layer node runs through each of the connections coming out of the input nodes. In the LSTM architecture, each input node is connected to all output nodes, each memory block, and the input gate, forget gate, and output gate of each respective memory block. Also, since the connections are weighted in the LSTM architecture, each output from the input nodes are multiplied by the weight of the connection upon reaching the destination node.

Recall that the hidden layer consists of memory blocks. Each memory block contains 1 input gate, 1 forget gate, 1 output gate, and 1 cell. The inputs to the memory block is summed. Then, the resulting value is multiplied with the output from the input gate. The new resulting value is then passed to the cell, summed with another connection called the CEC (recurrent connection from the cell's output to the cell's input), and it is stored in the cell as the cell state. Also, take note that the CEC, before it is summed, is also multiplied with the output from the forget gate. Now, the output from the cell is multiplied with the output from the output gate, and it is now passed out to the peephole connections (connected to all three gates) and out of the memory block as the memory block's output. The memory block's output is passed via weighted connections to all the output nodes in the output layer, to all the memory block's gates (input, forget, output), and back to the memory block itself as one of the inputs to the memory block. Furthermore, take note that the activation function for the gates is the logistic sigmoid of [0,1]. Each gate also sums their respective inputs and pass it through their respective activation functions.

The output layer consists of the same number of nodes as the ones in the input layer. Similar to a node in a traditional artificial neural network, the output nodes take all the inputs, sum them, then pass them through an activation function to output a value from the node. In this case, according to the paper, the activation function for the output units is a variant of the sigmoid function: a sigmoid function with range [-2,2].

The description above serves as our understanding of the LSTM architecture and how values are passed around the layers after piecing the parts together from (Gers et al., 1999), (Gers and Schmidhuber, 2001), and (Gers et al., 2002). Also, this is based on the research that has occurred on artificial neural networks, recurrent neural networks, and

LSTM in general. It has been tested to be effective to some sense, which is why we theorize it is used in such a way in this application. Take note that the architecture outlined here has elements of a traditional artificial neural network (input and output layer), recurrent neural network (how weights from the output of the hidden layer loop back to the hidden layer), and LSTM (the memory blocks, gates, and peepholes). Furthermore, this understanding was used to code the LSTM network for this project report. The code is modeled after the network architecture outlined here.

On a side note, one interesting thing to take into account is that the network architecture used in (Gers and Schmidhuber, 2001) uses peepholes, which were not published until 2002 by (Gers et al., 2002). Thus, it is possible that while (Gers et al., 2002) was still being written, work on (Gers and Schmidhuber, 2001) was being conducted with the most recent unpublished advancements of the LSTM architecture. Thus, one can expect that some things in (Gers and Schmidhuber, 2001) may be unclear. As one would expect, we had to look for many outside resources, including (Gers et al., 2002), to understand what was going on in the network of concern in (Gers and Schmidhuber, 2001).

4 Learning Algorithm and Backward Pass

Additional difficulty was encountered with the backward pass. The paper of concern with research on CFL and CSL with LSTM, (Gers and Schmidhuber, 2001), had a section devoted to the backward pass. It seems like that with a dedicated section, one should not have any trouble understanding and implementing it. Unfortunately, the section in (Gers and Schmidhuber, 2001) is really short and only mentions, along the lines of, "an efficient fusion of slightly modified, truncated BPTT and a customized version of RTRL" are used, and for more details, to reference other research papers. Thus, using the resources available, below is our best understanding of the backward pass and is what we implemented in our code.

The backward pass is where learning takes place in an artificial neural network. Similarly, a learning algorithm is run during the backward pass of a LSTM. In a traditional artificial neural network, a common learning algorithm is the backpropagation, where the error is propagated from layer to layer. The difference in error is first calculated at the output layer, then it is passed along the weights, changing the weight values in the process, until it reaches the input layer. In the case of the LSTM, it works similarly as well. In fact, the LSTM implements a gradient-based error propagation flow, just like a backpropagation learning algorithm. Thus, that is our learning algorithm for our LSTM, which is run on the backward pass.

To understand what makes up the learning algorithm, more detail will be explained. First, the LSTM network is provided with the target values that the output nodes should be outputting. Recall that each output nodes is representative of one of the characters in the language, and each output is either a value of +1 to indicate that character is predicted or a value of -1 to indicate that character is not predicted. Each output node would receive its target value and compare it to its own respective output from the forward pass. The error would be calculated by subtracting the output of the forward pass from the target value.

With each target error that has just been calculated, it is multiplied with the respective derivative of the activation function of the summed inputs to the node, which is then multiplied with a learning rate that is a parameter given to the network. This is then added to the previous weight value to update that respective weight coming into the output node in the output layer.

Next, partial derivatives are calculated. These are to help with updating the weights that are to follow. The partial derivatives that will be calculated corresponds with each respective memory block's cell state, input gate, and forget gate. For each of the partial derivatives, the derivative of the activation function is used, such as the derivative of the logistic sigmoid function, $f(x) \times (1 - f(x))$, is used for the gates. The formula for the partial derivatives consist of taking the previous time step's partial derivative and adding it to the influence of the peephole times the inputs to the unit.

Now, with the partial derivatives, one can continue with the backward pass. The set of weights to be updated are for those going into the memory blocks and each memory block's respective input gate, forget gate, and output gate. Before the updating can happen, some more error calculations are needed, specifically with the output gate and the memory block cell. To calculate the derivative/error of the output gate, the derivative of the activation function of the summed input to the output node (from the previous paragraph) is used here by summing over all those output node derivatives and multiplying it to the cell state, which is then multiplied with the activation function's derivative of the inputs to the output gate. This value just calculated is the derivative/error of the output gate. The derivative of the cell state is calculated in a similar fashion, but with the formula of multiplying the derivative of the activation function of the summed input to the output node with the output from the memory block. To clarify, recall that these derivatives being calculated in this paragraph represents the respective error, which has been propagated from the output layer.

Finally, update of the weights going into the memory blocks and their respective gates can occur. The partial derivatives from before are multiplied with each respective derivative/error, output gate with the derivative/error of the output gate and input/forget gate with the derivative/error of the cell state, and are multiplied with the learning rate to calculate the weight changes needed. Then, the weight changes are added to the old weights.

This, in summary, is how the learning algorithm works. Error is propagated from the output layer to the hidden layer. Propagating it directly to the input layer is not needed since all weights from the input layer is connected to the hidden and output layers. A gradient approach was used for the learning algorithm, which helps the network converge to the ”language” that it is trying to learn. Also, notice that since a target value is presented to the network that the learning occurring is called supervised learning in machine language terms.

On a side note, notice how that though the bias was mentioned in the network architecture, it has not been mentioned since. This is also what occurred in (Gers and Schmidhuber, 2001). However, though not much information on the bias was found in (Gers and Schmidhuber, 2001), we were able to piece together that the bias is incorporated in the weight updates by treating it similarly to the other input connections coming into the respective node. Thus, it is incorporated in all the learning and also in the forward pass algorithm, though it is not mentioned explicitly in the paper. One should assume that it is one of the many inputs to the respective units, and it is incorporated in the code written. Furthermore, we discovered that the bias is needed because the ”language” that is learned may not necessarily be centered where the initialization of the weights is referencing. Thus, with the help of the bias, specifically to the hidden layer units and the output layer units, the network could learn a language that is not just centered around a specific origin. This provides flexibility to the LSTM, which allows it to learn languages better.

5 Coding the Solution

The information explained in Section 3 and Section 4 were coded after much time was spent on understanding just what to code. Likewise, due to the ever changing understanding of the network, there were many changes that were made as classes were created/deleted and data structures changed. Nonetheless, Section 3 and Section 4 is our most updated understanding and what our code is based on. Also, these two sections together represent the main LSTM network.

Another part that was coded, which was not explained in much detail before, was generating the example languages in (Gers and Schmidhuber, 2001). TODO

6 Results

TODO Results: analyze the results using machine learning concepts to explain why get the results.

7 Comparison to Proposal

- Goal 1: Datasets are generated. LSTM is coded, able to run, and output results. **Achieved**

- Goal 2: Code is written to test the LSTM on the datasets. **Not Achieved**

- Goal 3: Comparable performance to the research paper on similar datasets. **Not Achieved**

Only the first goal was achieved. The complexity of the network and the need of piecing together information through many hours of discussion and other resources slowed the expected progress. Nonetheless, the fundamental network was coded and some output was achieved from it. Also, the algorithms for generating the datasets are made.

8 Conclusion

Although the project did not truly follow every goal of the proposal, we felt that we have learned a lot more than we had anticipated. A lot of research has been spent in the area of artificial neural networks, and the power and capabilities of the LSTM naturally leads to complexity in the architecture. Nonetheless, we were able to code the fundamental pieces of the LSTM network, step-by-step, and truly gain an understanding of how each piece fit into the network. We recognize that it would have been possible to take a library and use that to complete the goals of the proposal without spending more than the time specified in the project guidelines to understand and code the LSTM from scratch, but that would have taken away all the struggles, personal achievements, and learning. Ultimately, this paper and the code that accompanies it is representative of the understanding and work that we have dedicated to the project.

References

Donald F. Specht. 1991. A General Regression Neural Network. *Neural Networks*, 2(6):568–576.

Felix A. Gers, Jurgen Schmidhuber, and Fred Cummins. 1999. Learning to Forget: Continual Prediction With LSTM. *Artificial Neural Networks*, 2(470):850–855.

Felix A. Gers and Jurgen Schmidhuber. 2001. LSTM Recurrent Networks Learn Simple Context-Free and Context-Sensitive Languages. *IEEE Transactions on Neural Networks*, 12(6):1333-1340.

Felix A. Gers, Nicol N. Schraudolph, and Jurgen Schmidhuber. 2002. Learning Precise Timing with LSTM Recurrent Networks. *Journal of Machine Learning Research*, 3:115–143.

Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive Science*, 14(2):179–211.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2014. Show and Tell: A Neural Image Caption Generator. *CoRR*, abs/1411.4555.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.