Understanding RNNs & LSTMS and Reviewing Their Application In Precipitation Nowcasting

Sarwan Shah
Arizona State University
sshah219@asu.edu

Abstract

Long Short-Term Memory (LSTM) networks and Recurrent Neural Networks (RNNs) are fundamental architectures in machine learning, particularly when it comes to sequential data analysis. This paper, in an attempt to develop an understanding of these machine learning paradigms, reviews the mathematical theory and modeling of these architectures. It also makes an effort to highlight the practical implications & limitations of these architectures, while also reviewing the literature on their usage in precipitation nowcasting for contextualizing arguments made.

1 Introduction

The use of modern machine learning, particularly the evolution of deep neural networks, has today opened up a gateway that allows us to capture complex patterns in problems, in data, that have historically been difficult to model mathematically, with an increasingly high degree of accuracy. Consequently, there has been a huge influx of research that is attempting to understand this ability of theirs and also experimenting with their application across a variety of domains. Following this suite, assuming having a foundation established in traditional machine learning modalities (regression, SVM, etc) and their theory, in this paper we attempt to develop an understanding of how modern neural networks work and build upon their traditional counterparts. More particularly our discussion explores the modeling of recurrent neural networks (RNNs) and their variation known as Long Short-Term Memory (LSTM). We highlight some of the key features and limitations of such networks. Following this we also review their application in addressing the challenging problem of precipitation nowcasting. Lastly, we conclude by commenting on the key insights that were taken away from this study.

2 Neural Networks

2.1 Modelling & Architecture

The most basic architecture of a neural network consists of a single neuron that forms a singular hidden layer (in this case) before its output and is fed by inputs via edges that have associated weights, which also form the weights of that neuron. The neuron computes the product of the input and the weights associated with the incoming edge and then applies a function, known as the 'activation function', to give the final output. This architecture is known as a feed-forward network. See figure 1 (a).

$$y = sign\{\mathbf{W}.\mathbf{X}\} = sign\{\sum w_i x_i\} \tag{1}$$

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

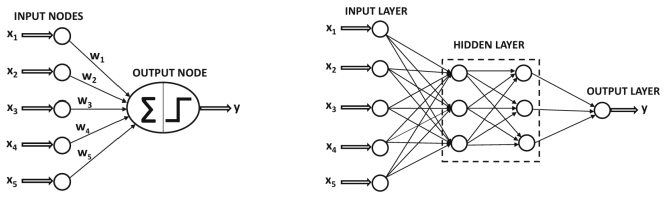

Figure 1: Singe (a) & Multi-Layered Neural Network Architecture (b) [1]

It is not difficult to see that with a 'Sign' activation function, this network models the perceptron for binary classification, as shown in equation (1). Similarly, if an identity activation function is used, the network models' linear regression. However, we know such ML algorithms fail to generalize well on data that is not linearly separable.

The additional neurons in each hidden layer and the introduction of more hidden layers - which have inter-connected neurons amongst each other using edges with associated weights - coupled with non-linear activation functions allow for a significant scaling in the expressive power of such networks. Intuitively, with 'Sign' activation it is like combining multiple perceptron-like units to capture statistically complex patterns in data. This is shown in figure 1 (b). And while additional neurons per layer enable capturing more complex patterns, a better trade-off, that has been observed in literature, is to have more layers, or deeper networks, to achieve the same results with fewer overall parameters.

An interesting result that was realized was that the non-linear expressiveness of scaled or deeper neural networks is essentially enabled by the use of non-linear activation functions for neurons, as it was shown in [1] that deep neural networks reduce to linear regression if the neurons consist of only identity activations. Non-linear activations such as 'Sigmoid', 'ReLU', 'Tanh' etc, enable non-linear expressiveness by reducing or mapping the non-linear solution space to a linear one.

Mathematically, it is useful to represent the inputs, outputs, weights, and associated layers in a neural network as vectors and matrices. Thus, for a d-dimensional input d_i, and given h_r hidden-layers of length p_r where $r = \{1, 2, ...k\}$, the first hidden-layer weights have dimensions d x p_1, where as successive r hidden-layers have weights p_r x p_{r+1}, and the output is p_k x d_o. Additionally, the outputs at each stage of the network are governed by the following equations, where f represents the activation function, $\bar{x}$ the input vector, W the weight matrices, and h_k the output vector of a hidden-layer.

$$\bar{h}_1 = f(W_1^T \bar{x}) \tag{2}$$

$$\bar{h}_r = f(W_r^T \bar{h}_{r-1}) \tag{3}$$

$$\bar{y} = f(W_{out}^T \bar{h}_k) \tag{4}$$

2.2 Training via Back Propagation

Another important aspect of machine learning paradigms is training for a solution that generalizes well over unseen data. The quantity of data required to train neural networks as a rule of thumb is recommended to be 2 to 3 times the number of parameters in the network; which is governed by the number of neurons and edges it has. This ratio helps achieve a good balance against model complexity, helping avoid issues of under or over-fitting. However, this requirement can be relaxed in some cases using techniques such as initialization of weights via pre-training on shallower networks. On the other hand, loss functions used to train neural networks must be differentiable and take the same form as with any optimization-based problem, while their choice remains largely application-specific.

The loss helps optimize edge weights across the network in a stochastic gradient descent fashion using a technique known as backpropagation which calculates the partial derivative of the loss with respect to the weights in the network and updates their weights accordingly. This is enabled by employing the multivariate chain rule in reverse from the output and along the different paths in the network.

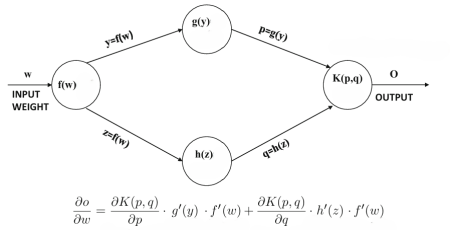

Figure 2: Backpropogation through single-layer two neuron network [1]

2.3 Challenges

A common challenge in the training process for deep neural networks is addressing vanishing and exploding gradients while backpropagating. This happens because the derivatives of activation functions have a saturating nature, in which case, if their gradient computes to < 1, then, with derivative products taking place amongst multiple layers, the successive gradient quickly approaches zero. On the contrary, if their gradient computes to > 1, then, their successive gradient can quickly approach larger values.

This inherent instability in training neural networks is partially addressed by choosing activation functions such as 'ReLU' or 'hard Tanh' whose derivatives take values of either 1 (in their region of operation) or 0, and are thus less prone to vanishing or exploding gradients. However, on the flip side, their extremeness can lead to dead neurons (because of 0 gradients if pre-activation values lie outside their region of operation) early on in the training process at times, and thus, additional variants such as 'leaky ReLU' have been devised to address this issue, as they continue to leak some gradient when backpropagating even when pre-activation values like outside their region of operation. This ensures the neurons don't die entirely suddenly. Additionally, techniques such as batch normalization between layers also help limit the aforementioned along paths in the network.

It is worth commenting that some of this characteristic instability at times is important and adds to the expressiveness of neural networks. For instance, having dead neurons may actually turn out to be essential to be able to learn a desired pattern or relationship in the data.

3 RNNs And LSTMs

3.1 RNNs

3.1.1 Modeling & Architecture

Recurrent Neural Networks are a type of neural network architecture that enables the capturing of temporal relationships in data sequences. In the case of language modeling, the temporal relationship translates to a semantic one between word sequences.

From a modeling perspective, this is achieved by having a separate hidden layer (or layers) in the network for each time-stamp or position, that is input from a time-series or sequence of data. This is done by sharing weights between the consecutive time-stamp hidden layers such that while each time-stamp hidden layer (or layers) has the same input and output weights, it additionally depends on weights from the previous time-stamp hidden layer (or layers), allowing the network to capture temporal dependencies in the data. Architecturally, this is accomplished by incorporating a weighted self-loop, W_{hh}, on the hidden layer which captures the layering through time behavior. This is shown in fig 3 (a) & (b) for a neural network with a single hidden layer. Another key advantage of this architecture is that it allows the network to keep a fixed number of parameters while allowing variable inputs.

Mathematically, in this case, the equations governing the output of the network for $r = 1, 2...k$ hidden states and a sequence t inputs are (5) and (6), where t captures the layering through time and r through hidden layers. This recursive nature of these equations is what gives RNNs their name.

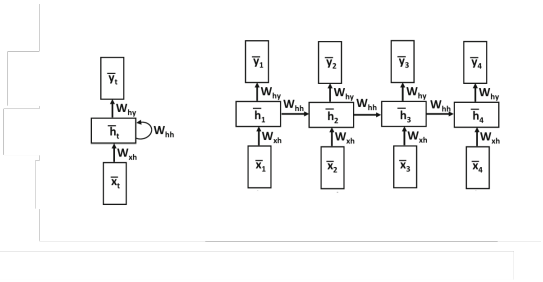

Figure 3: RNN architecture (a) & RNN architecture unfurled through time (b) [1]

$$\bar{h}_t = tanh(W_{xh}\bar{x}_t + W_{hh}\bar{h}_t - 1) \tag{5}$$

$$\bar{h}_t^r = tanh(W^r[\bar{h}_t^{r-1}\bar{h}_{t-1}^{r-1}]^T) \tag{6}$$

3.1.2 Training via Backpropogation

Backpropagation in RNNs similarly takes place as in conventional neural networks. While performing backpropagation, the layering through time is ignored and the shared weights involved in this process are assumed to be independent of each other during gradient computations. However, once these independent gradients have been computed, they are simply summed across the time dimension to give the actual gradients for W_{xh}, W_{hh}, and W_{hy}. This is known as backpropagation through time (BPTT).

3.1.3 Challenges

One of the challenges with training RNNs is that the time dimension can grow significantly long in some cases. This causes a computational and memory overload during gradient computations. Thus, a truncated version of BPTT is used, which truncates the gradient computation through time to a user-defined length.

Additionally, owing to the deep depth of such networks during backpropagation updates across the timed layers, the gradient gets multiplied with the shared weights consistently. If the shared weight w < 1, this quickly leads to vanishing gradients, or exploding ones in the opposing case. This adds on top of the already existing vanishing and exploding gradient problem caused by activation functions. This makes training RNNs a challenge. To address this inherent instability of RNNS, techniques such as strong regularization, gradient clipping, and layer normalization have been used. Moreover, it has given rise to additional architectural variants of the RNNs such as Echo-State networks: randomize the majority of the hidden-layer weights and only train output layers, LSTMs: maintain a cell state for stability, and GRUs (Gated Recurrent Units): achieve similar stability as the LSTMs but with fewer parameters.

3.2 LSTMs

3.2.1 Modeling & Architecture

LSTMs (Long Short-Term Memory) are an architectural variation of RNNs that add more stability by reducing the vanishing or exploding gradient problem by introducing cell states. These cell states store information about the past states of the layer and are incorporated as a part of the updates of the hidden states. Architecturally, this is achieved by introducing an intermediary cell state vector $\bar{c}_t^r$ that is computed as in (7, 8), via the generation of intermediary input $\bar{i}$, forget $\bar{f}$, output $\bar{o}$, new cell state $\bar{c}$ gate values. These gate values govern whether the new cell state needs to be added, previously forgotten/reset, or allowed to leak slightly into the hidden state being updated, as represented by $\bar{i} * \bar{c}$, $\bar{f} * c_t^{r-1}$, and, $\bar{o} * tanh(\bar{c}_t^r)$ in equations (8, 9) respectively.

$$[\,\bar{i}\;\bar{f}\;\bar{o}\;\bar{c}\,]^T = [\text{sigm sigm sigm tanh}]^T(W^r[\bar{h}_t^{r-1}\bar{h}_{t-1}^{r-1}]^T) \tag{7}$$

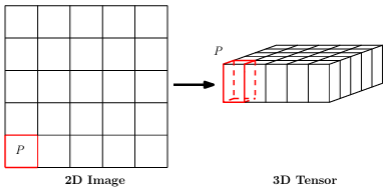

Figure 4: 2D to 3D Tensor [4]

$$\bar{c}_t^r = \bar{f} * c_t^{r-1} + \bar{i} * \bar{c} \tag{8}$$

$$\bar{h}_t^r = \bar{o} * tanh(\bar{c}_t^r) \tag{9}$$

The presence of these cell states adds more randomization and uniqueness to each hidden state compared to just having the same shared weights temporally. This helps significantly in tackling the vanishing and exploding gradient problem in conventional RNN architectures.

3.3 LSTMs for Precipitation Nowcasting

Although a very powerful modeling tool, a natural disadvantage of RNNs and LSTMs is that they cannot be used with very large sequences of data as in the case of time-series data for weather forecasting over a period of days. This is because, as explored in the previous sections, very large sequences lead to a significant increase in the temporal depth of the network, which in turn leads to computational, memory, and performance-related limitations.

However, another realm of weather forecasting known as 'Nowcasting' concerns short-range weather predictions that usually span a window of 0 to 6 hours. In Meteorology, this is usually considered a more challenging problem, as it is difficult to model the inherently turbulent nature of weather phenomena over a short range, and computationally very expensive to simulate. Thus, in this section, we will explore some literature that experiments with the use of LSTMs to generate short-range forecasts for the case of precipitation nowcasting.

3.3.1 From LSTM to ConvLSTM

In [4] they present a neural network architecture that builds upon the LSTM to give a convolutional LSTM (ConvLSTM) that is used to predict the future states of precipitation radar maps.

The key identification they make is that while LSTMs can capture temporal relationships, there is no way for them to effectively capture spatial relationships, which are often critical to weather forecasting. To add this spatial awareness they add two spatial dimensions to their inputs, hidden-states, and outputs, forming them into 3D tensors (see figure 4, thus giving each layer a shape similar to convolutional layers in a CNN (convolution neural network).

Moreover, they use the convolution operation as a part of their recurrence conditions/equation to ensure spatial patterns are captured, and highlight, how larger kernels can capture the influence of data points further away, potentially useful for tracking faster movements (e.g., fast-moving weather systems).

They conclude by demonstrating how their ConvLSTM performs significantly better across a variety of metrics: rainfall mean squared error (Rainfall-MSE), critical success index (CSI), false alarm rate (FAR), probability of detection (POD), and correlation, compared to state-of-the-art ROVER algorithm (which is a statistical algorithm for nowcasting using radar maps) and a normal LSTM.

3.3.2 From ConvLSTM to TrajGRU

This [5] paper builds upon the work conducted in [4]. The key improvements this paper highlights are that it is used as GRU (Gated Recurrent Unit) instead of the LSTM to introduce performance gains

and reduce the number of parameters required in the network. Moreover, it highlights that even with a ConvGRU, it uses a fixed connection structure and weights for all locations, which may not capture motion patterns that involve rotation and scaling effectively. Thus, it proposes the TrajGRU, which learns a location-variant connection structure that can better model these motion patterns. It does this by calculating continuous optical flows between inputs to represent the connections between locations by embedding it as a part of its recurrence update conditions structure.

The paper concludes by showing how the TrajGRU performs better than ConvLSTM & ConvGRU-based networks using newly defined metrics on the benchmark dataset HKO-07 they created. The new metric adjusts for high-impact cases such as the correct or incorrect prediction of heavy rainfall, and assigning the scenario higher weights.

4 Conclusion

In conclusion, through this paper we have provided a comprehensive exploration into the fundamental architectures of Long Short-Term Memory (LSTM) networks and Recurrent Neural Networks (RNNs), delving into their mathematical theory, modeling, training methodologies, and practical applications.

Key takeaways made through this review were:

an understanding of the evolution of neural networks from single-layer perceptron-like structure to multi-layered architectures capable of capturing complex patterns in data and that is largely enabled by the use of non-linear activation functions. using only linear activations in a neural network reduces them to a linear regression problem. a theoretical and mathematical understanding of why challenges such as vanishing and exploding gradients arise while training deep neural networks, and why the techniques used to mitigate them are effective, such as the use of specific activation functions and normalization techniques. realizing that instabilities in the training process of neural networks add to expressiveness. a theoretical and mathematical understanding of the modeling and architecture of RNNs and LSTMs, how simple modeling tricks enable use to layer sequences through time and preserve states to address critical challenges inherent to the architecture. an appreciation that while RNNs & LSTMs are powerful, their limitations with very long sequences make them less than ideal for tasks like long-term weather forecasting. observing that LSTMs excel at short-term prediction tasks like precipitation nowcasting, where the time window is limited. Understanding the evolution of LSTMs to ConvLSTMs and TrajGRUs to incorporate spatial characteristics to the network and improve their forecasting ability in a targetted manner for the task of precipitation forecasting.

In summary, this paper contributes to the understanding of LSTM and RNN architectures, their training methodologies, and their application in real-world scenarios like weather forecasting. As machine learning continues to advance, further research into enhancing the capabilities of these architectures will undoubtedly lead to even more impactful applications across various domains.

References

[1] Aggarwal, C. C. (2023). Neural Networks and deep learning a textbook. Springer International Publishing AG.

[2] Goodfellow, Ian, et al. “Deep Learning.” Deeplearningbook.org, 2016, www.deeplearningbook.org/. .

[3] Hochreiter, Sepp, and Jürgen Schmidhuber. “Long Short-Term Memory.” Neural Computation, vol. 9, no. 8, Nov. 1997, pp. 1735–1780.

[4] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. 2015. Convolutional LSTM Network: a machine learning approach for precipitation nowcasting. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS’15). MIT Press.

[5] Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, & Wang-chun Woo. (2017). Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model.

[6] Hou, L., Zhu, J., Kwok, J., Gao, F., Qin, T., & Liu, T.Y. (2019). Normalization Helps Training of Quantized LSTM. In Advances in Neural Information Processing Systems. Curran Associates, Inc..

[7] Jiahao Su, Wonmin Byeon, Furong Huang, Jan Kautz, & Animashree Anandkumar (2020). Convolutional Tensor-Train LSTM for Spatio-temporal Learning. CoRR, abs/2002.09131.