RNN: EVOLUTION AND LIMITATIONS IN GENERATING TEXT SEQUENCES

A literature survey submitted to Prof. Kazunori Okada of
San Francisco State University
In partial fulfillment of
the requirements for
CSC 872 Pattern Analysis and Machine Intelligence

Master of Science

In

Business Analytics

by

Shailesh Krishna

San Francisco, California

December 2019

Abstract

Recurrent Neural Networks (RNNs) are compelling sequence models that are capable of learning features and long-term dependencies from data. A well-trained RNN can model any dynamical system; however, they do not enjoy widespread use because it is challenging to train them properly. Issues in learning long-term dependencies, mostly impede training RNNs. This literature survey discusses RNNs, briefly describing its architecture and types. This survey describes a few critical challenges faced by the researchers, along with the architectural advances made in the field of RNN research. Lastly, the survey concludes by discussing the current state of research and its future discourse.

Table of Contents

LIST OF TABLES

Table Page

LIST OF FIGURES

Figures Page

Introduction

Artificial neural networks (ANNs) are computing systems that are inspired by, but not identical to the human brain. They consist of layers of connected units called artificial neurons. Such computing systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules [14]. As the number of layers increases, the complexity of network increases too. ANNs with recurrent connections are called recurrent neural networks, or RNNs (Rumelhart et al., 1986a), which are a family of neural networks for processing sequential data. RNNs process an input sequence one element at a time, maintaining in their hidden units a "state vector" that implicitly contains information about the history of all the past elements of the sequence [7].This structure enables RNNs to store, remember, and process past complex signals for extended periods.

RNNs are robust dynamic systems, but training them has proved to be problematic because the backpropagated gradients either grow or shrink at each time step, so over many time steps, they typically explode or vanish [2]. Thanks to advances in their architecture and ways of training them, RNNs are very good at predicting the next character in the text or the next word in a sequence, but they can also be used for more complex tasks [1].

RNNs can scale to much longer sequences than would be practical for networks without sequence-based specializations, but before diving deep into RNNs, we first need to understand what a sequence is.

Sequence Modeling

A sequence consists of a stream of data that can either be finite or infinite. Modeling such data where either the input or output is a sequence is termed as sequence modeling. Sequence models

can be applied to a variety of domains in deep learning, such as text prediction, machine language translation, handwriting synthesis, and video translation [12].

RNN Architecture and Types

RNNs are a class of supervised machine learning models, made of artificial neurons with one or more feedback loops. A simple RNN has three layers, which are input, output, and a hidden layer.

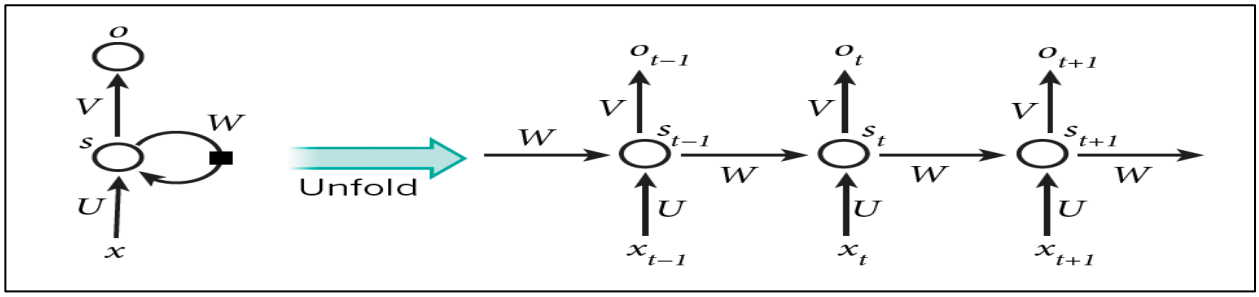

Figure 1. Basic RNN Architecture [13]

Figure 1 above summarizes the underlying RNN architecture. RNNs scan data from left to right; the artificial neurons (for example, hidden units grouped under node s with values s_t at time t) get inputs from other neurons at previous time steps (this is represented with the black square, representing a delay of one-time step, on the left). In this way, a recurrent neural network can map an input sequence with elements x_t into an output sequence with elements o_t, with each o depending on all the previous x_t' (for $t' \leq t$). The same parameters (matrices U, V, W) are used at each time step [13].

The goal here is to calculate the gradient of errors for our parameters. For each layer of the network, we calculate the loss and then sum up the losses to obtain the entire loss for the sequence. The backpropagation algorithm, which in this network is called BPTT or "backpropagation through time" can be directly applied to the computational graph of the

unfolded network on the right, to compute the derivative of a total error for all the states s_t and all the parameters [13].

Since RNNs allow us to operate over sequential data, based on the application needs, different models can be created. Figure 2 below shows some of the RNN types.

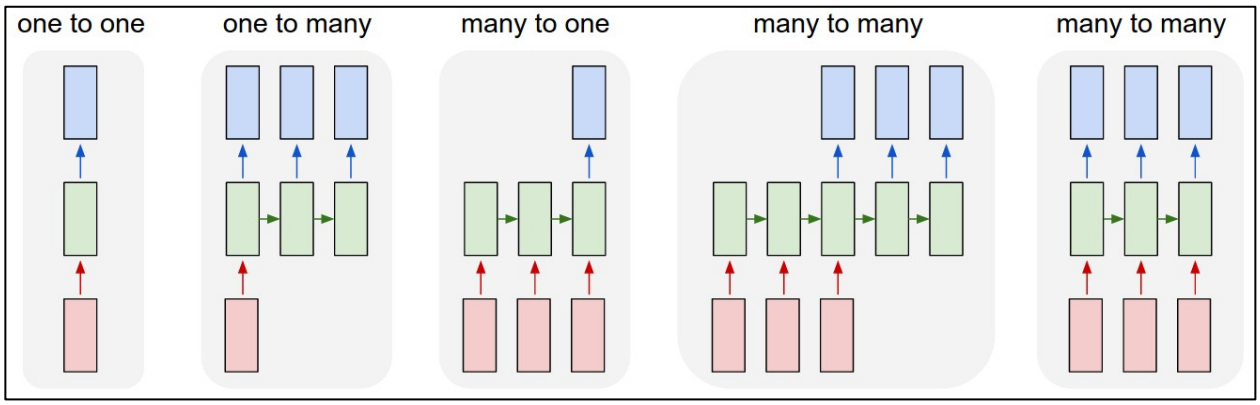

Figure 2. Types of RNNs [12]

As depicted above in Figure 2, the RNN structure varies with application; however, the most common form of architecture is Encoder-Decoder architecture [13]. The design is simple. An encoder processes the inputs sequence and emits the context, usually as a simple function of its final hidden state. The decoder is conditioned on that fixed-length vector to generate the output sequence [7]. There is no constraint that the encoder must have the same size of the hidden layer as the decoder.

In this literature survey, the focus is on understanding the contexts as to how RNNs can be used for language modeling to predict the next character in the text or the next word in a sequence [1]. The survey comprises of sections describing various techniques and architectural advancements described in the study research papers, followed by a detailed critique of various techniques, along with their strengths and weaknesses. Finally, the literature survey ends with a conclusion and related future course of work.

A Neural Probabilistic Language Model [1]

A statistical language model is a probability distribution over a sequence of words. Given such a sequence of words, say of length m, it assigns a probability to the whole sequence of words [11]. The traditional statistical modeling of language did not exploit distributed representations: it was based on counting frequencies of occurrences of short symbol sequences of length up to N (called N-grams). The number of possible N-grams is on the order of V^N, where V is the vocabulary size, so taking into account a context of more than a handful of words would require extensive training corpora. N-grams treat each word as an atomic unit, so they cannot generalize across semantically related sequences of words [13].

In this research paper, Bengio et al. proposed a neural probabilistic language model that mitigates the problem knows as the curse of dimensionality. The proposed model learns simultaneously, a distributed representation for each word along with the probability function for word sequences, expressed in terms of these representations. Sequence of unrecognized words are assigned higher probabilities if they comprise of words similar to the words present in the existing sentences [1]. This logic helps with the generalization.

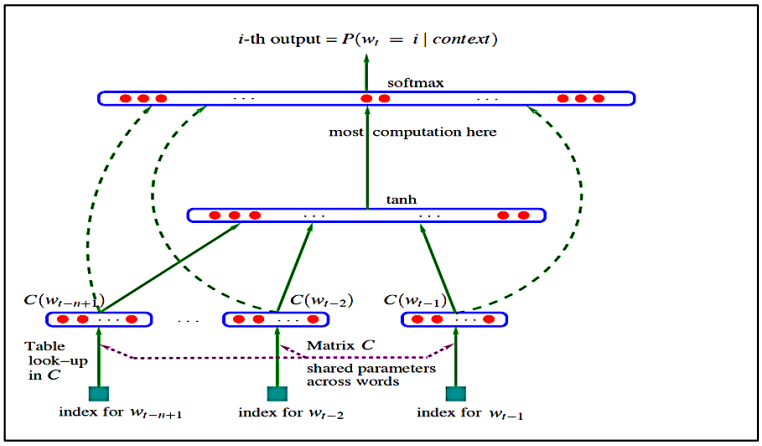

Figure 3. Neural Architecture

The neural architecture model depicted in Figure 3 resembles a multi-layer perceptron. In the above model, the number of free parameters only scales linearly with V, the number of words in the vocabulary, which is a large but finite set. It consists of a softmax output layer, which guarantees positive probabilities summing to 1. The middle "Tanh" layer is a sigmoid activation function; extreme negative values get assigned values very close to -1 and positive ones in the opposite manner. The bottom layer comprises of the input layer. The dotted green lines are the direct connections from input to output.

Based on the experiments conducted on two large datasets of AP News and Brown corpus [1], they were able to show that the neural language model performed better than n-gram models. The neural network was able to handle a broader context as compared to n-gram models [13]. An essential contribution of this paper was to demonstrate the feasibility of training RNNs at scale [1]. Though the computations were expensive, the neural language models were able to generate good results. Lastly, they also emphasized on future improvements mainly aimed at the architectural level, computational efficiency, and broadening context.

Learning Long-Term Dependencies with Gradient Descent is Difficult [2]

The neural language model described above demonstrated that RNNs could be effectively used for learning distributed representation of words in a more extended context than the traditional language models [1]. RNNs were suitable for this task as they have an internal state that allows them to store and update context information [7]. Historically, learning algorithms used for RNNs utilized a simple yet popular optimization method called gradient descent (GD). The weights of the model are adjusted by calculating the error function derivatives concerning a member of the weight matrices in the model [7]. In short, computing the gradient of a cost function with respect to the weights of the network. Due to the intrinsic structure of RNNs, the

relationship amongst the parameters made gradient computation a challenge. They were not able to learn long-term dependencies when gradient descent is utilized as an optimization algorithm during training. The fundamental problem reported was that the gradients propagated over many stages tend to either vanish or explode [2].

Separate researchers independently discovered the vanishing and exploding gradient problem for RNNs (Hochreiter, 1991; Bengio et al., 1993, 1993) [7]. The vanishing gradient problem refers to the exponential decay of gradient, as it is backpropagated through time, thus, making it difficult to know which direction the parameters should move to improve the cost function. The alternate scenario is the exploding gradients that can make learning very unstable [7].

It would not have been wrong to assume that the underlying problem can be mitigated by staying in a limited region where the gradients do not vanish or explode. However, Bengio et al., through their findings, proved that in order to store information that is robust to minuscule disturbances, one needs to be in space where the gradients vanish. Specifically, whenever the model can represent long-term dependencies, the gradient of a long-term interaction has exponentially smaller magnitude than the gradient of a short-term interaction. This means not that it is impossible to learn, but that it might take a very long time to learn long-term dependencies because the signal about these dependencies will tend to be hidden by the smallest fluctuations arising from short-term dependencies [7]. To demonstrate this, Bengio et al. conducted experiments to show that as we increase the span of the dependencies that need to be captured, gradient-based optimization becomes increasingly difficult, with the probability of successful training of a traditional RNN via stochastic gradient descent rapidly reaching 0 for sequences of only length 10 or 20 [7].

For evaluation purposes, they defined a parametric dynamic system (PDS) having the following characteristics:

- The system can store information for an arbitrary duration
- The system is resistant to noise.
- The system parameters are trainable.

A minimal task satisfying the above constraints was defined, and experiments were conducted to understand the issue with learning simple long-term dependencies. Results indicated that either the system is susceptible to noise or the derivatives of the cost at a time for the system activations converge exponentially to 0 as time increase [2]. The situation described was the essential reason for the difficulty in using gradient descent to train a dynamical system to capture long-term dependencies. In their experiments, alternate optimization methods were evaluated, and the results indicated that there might be ways to reconcile learning with storing [2].

Long Short-Term Memory [3]

As described above, training RNNs to store information over extended timesteps using gradient descent takes a very long time, mostly due to exponential decay of gradient as it backpropagates through time. All the models so far described in the previous paper fell short as the span of input/output dependencies were increased. This practical challenge of training RNNs led to the near abandonment of RNN research [14].

Through this research paper, (Hochreiter and Schmidhuber, 1997) describe the analysis done by Hochreiter in 1991 for the problem of learning long-term dependencies and introduce an efficient recurrent architecture termed as Long Short-Term Memory (LSTM) [7]. The experiments conducted using datasets containing distributed, real-valued, and noisy pattern representations demonstrate the efficiency of new LSTM based architecture. LSTM is local in space and time; its computational complexity per time step and weight is O(1).

Memory Cell

The LSTM architecture introduces the idea of self-loops to create paths for gradients to flow for long durations, thus overcoming these error back-flow problems [7]. LSTM recurrent networks have "Memory Cells" that have an internal recurrence (self-loop), in addition to the outer recurrence of the RNN [7]. Each cell has the same inputs and outputs as an ordinary recurrent network but also has more parameters and a system of gating units that control the flow of information.

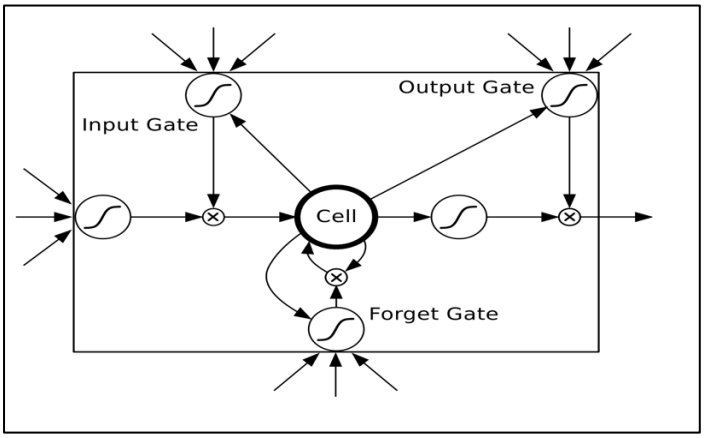

Figure 4. LSTM Memory Cell

The figure above (Figure 4) describes the architecture of a memory cell comprising of an input gate that writes to cell, an output gate that reads from cell and a forget gate that resets the information present in the cell.

As mentioned in the paper, the LSTMs can learn to bridge time intervals above 1000 steps even in case of noisy, incompressible input sequences, without loss of short time lag capabilities. The experiment premises set up to demonstrate the capabilities of this architecture defines the task with two constraints. Firstly, the minimal time lags between relevant input signals and corresponding teaching signals must be very long for all the training sequences. The second

essential requirement is that the tasks should be sophisticated enough that they cannot be solved quickly by utilizing simple-minded strategies such as random weight guessing [3].

Experiments

Below mentioned are some of the experiments described in the paper.

- Long time lag problems with input and signal on the same line:

  In this experiment, they focused on solving the Bengio et al.'s "2-sequence problem" and its variants by utilizing different LSTM architectures. They were able to demonstrate that despite the noisy targets, LSTM was able to learn the expected target values.

- Long time lag problems involving distributed, continuous-valued representations:

  Using the toy adding problem dataset in this experiment, they demonstrated that LSTM could work well with distributed representations and is also able to learn to perform calculations involving continuous values.

- Extract information conveyed by the temporal order of widely separated inputs:

  Since relevant information can be spread across different positions in the input sequence, in this experiment, it was demonstrated that LSTM can extract information conveyed by the temporal order of widely separated inputs.

This new LSTM architecture proved that they are better at storing and accessing information than standard RNN. The different variants of LSTM architecture have been found hugely successful in many applications such as unconstrained handwriting recognition (Graves et al., 2009), speech recognition (Graves et al., 2013), and handwriting generation (Graves, 2013) [7].

Generating Text with Recurrent Neural Networks [5]

While training RNNs, the principal issue reported by researchers is that naive gradient descent approach is unsuitable for optimizing objectives that exhibit pathological curvature. Second-

order optimization methods, which model the local curvature and correct for it, were quite successful on such objectives. Thus, it was reasonable to assume that the application of such techniques could help mitigate the problem.

In this paper, Sutskever et al. demonstrated that by training RNNs using second-order optimization methods such as Hessian-free optimization [4], RNNs could be trained to learn good language models. While doing so, they proposed a new architecture MRNN and demonstrated that it is more suitable than the standard RNNs for performing such tasks. The Hessian-free optimization method used in the paper was developed by Martens (2010), which was powerful enough to train profound neural networks from random initializations. Since RNNs are extremely deep neural networks with weight sharing across time, the same HF optimizer should be able to train RNNs as well.

Hessian-free Optimization

The basis of the Hessian-free second-order optimization method developed by Martens (2010) stems from the truncated-Newton method [4]. However, the Hessian-free version developed makes a series of essential modifications to the primary approach, such as using the positive semi-definite Gauss-Newton curvature matrix in place of the possibly indefinite Hessian [6]. The figure 5 below shows the information flow in the HF optimization method.

Algorithm 1 The Hessian-free optimization method

1: **for** $n = 1, 2, \ldots$ **do**
2: $g_n \leftarrow \nabla f(\theta_n)$
3: compute/adjust λ by some method
4: define the function $B_n(d) = \mathbf{H}(\theta_n)d + \lambda d$
5: $p_n \leftarrow \text{CG-Minimize}(B_n, -g_n)$
6: $\theta_{n+1} \leftarrow \theta_n + p_n$
7: **end for**

Figure 5. Hessian-free Optimization Method

Multiplicative RNN

Multiplicative RNNs uses multiplicative (also called "gated") connections to allow the current input character to determine the transition matrix from one hidden state vector to the next. Using this architecture they demonstrated the power of a large RNN trained with the HF optimizer by applying them to the task of predicting the next character in a stream of text [5].

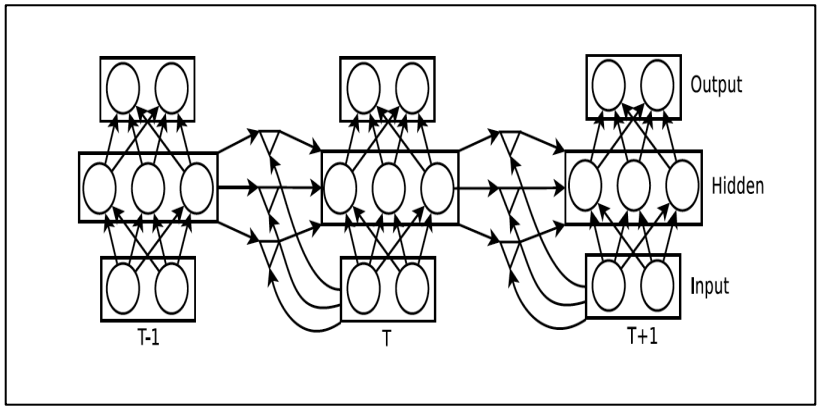

Figure 6. Multiplicative RNN Architecture [5]

The above figure (Figure 6) shows the underlying architecture of MRNNs. The multiplicative recurrent neural Network "gates" the recurrent weight matrix with the input symbol. Each triangle symbol represents a factor that applies a learned linear filter at each of its two input vertices. The product of the outputs of these two linear filters is then sent, via weighted connections, to all the units connected to the third vertex of the triangle. Consequently, every input can synthesize its own hidden-to-hidden weight matrix by determining the gains on all of the factors, each of which represents a rank one hidden-to-hidden weight matrix defined by the outer-product of its incoming and outgoing weight-vectors to the hidden units. The synthesized weight matrices share "structure" because they are all formed by blending the same set of rank-

one matrices. In contrast, an unconstrained tensor model ensures that each input has an entirely separate weight matrix [5].

Experiments

For evaluation purposes, three large text corpora comprising of Wikipedia, articles from NYT, and a corpus of research papers were chosen. Rather than training the entire set sequentially, they trained MRNN on shorter sequences to predict only the last 200 timesteps of the 250-long training sequences as predicting the first few characters was computationally expensive at that time. The results were compared with other existing systems such as Memoizer and PAQ. The below table summarizes the results.

DATA SET	MEMOIZER	PAQ	MRNN	MRNN (FULL SET)
WIKI	1.66	1.51	1.60 (1.53)	1.55 (1.54)
NYT	1.49	1.38	1.48 (1.44)	1.47 (1.46)
ML	1.33	1.22	1.31 (1.27)	

Table 1. MRNN Experiment Results [6]

Hence, they were able to demonstrate that using a robust learning system like MRNN, the task of learning words becomes very easy.

Critique

The field of artificial intelligence has evolved over the years. Machine learning and deep learning have become an essential part of it nowadays. As the research progressed, various novel improvements and approaches were suggested to tackle unsolved challenges. One such issue was data-driven natural language processing. In natural language processing or the NLP, the goal is

to teach the machine to understand the human dialect was a daunting task due to a large number of discrete variables involved. In the context of language modeling, some success was achieved by linguistics. They were able to build language models that were based on counting frequencies of occurrences of short symbol sequences of length up to N (called N-grams). However, they did not take advantage of the distributed representation of words.

The papers studied in the literature survey provided valuable insights about how RNNs have proved to be a gamechanger in the language modeling domain, especially in predicting the next character in the text or the next word in a sequence. Language modeling comprises modeling sequences of text. RNNs have proved to be very powerful in tasks involving sequence modeling. The papers mentioned in this literature survey provided insights into the challenges faced by researchers and the multiple approaches adopted by them to handle issues involving training RNNs.

In "A Neural Probabilistic Language Model," Bengio et al. described a statistical language model that utilized RNNs to learn distributed representation of words. The model fared better than the traditional N-gram models in learning a more extended context of words. The proposed model was successful in tackling the curse of dimensionality as it was able to take advantage of the learned distributed representations of words. However, the model was still naive in terms of learning long-term contexts and required additional training to be able to perform the required tasks. The model proved to be a stepping stone for understanding long-term contexts in terms of language modeling as it replaced tables of conditional probabilities by a more compact and smoother representations based on distributed representations that can accommodate far more conditioned variables. Through this model, they were able to shift the focus of researchers from

restricting or summarizing the conditioning variables to avoid overfitting to other areas of computation and memory management.

As demonstrated by Bengio et al., RNNs are very powerful in their ability to represent context and can easily outperform other static networks.

In "Learning Long-Term Dependencies with Gradient Descent is Difficult" the problem known as vanishing gradient is described, which was the typical problem reported by the researchers while training RNNs on tasks in which the temporal contingencies present in the input/output span long intervals. During that time, the learning algorithms used for recurrent neural networks were usually based on computing the gradient of a cost function for the weights of the network. Through their experiments, they proved that in order to store information that is robust to minuscule disturbances, one needs to be in space where the gradients vanish. Specifically, whenever the model can represent long-term dependencies, the gradient of a long-term interaction has exponentially smaller magnitude than the gradient of a short-term interaction. This means not that it is impossible to learn, but that it might take a very long time to learn long-term dependencies because the signal about these dependencies will tend to be hidden by the smallest fluctuations arising from short-term dependencies. The experiments described in the paper helped to understand the underlying problem in a better context and also paved the way for future optimization algorithms and architectures.

The "Long Short-Term Memory" architecture opened new avenues in the natural language processing domain. The addition of memory cell to facilitate gradient flow for long durations was the main contribution of this paper. The memory cell's gated internal architecture helped bridging very long-time lags. The experiments conducted in the paper proved the ability of LSTMs to handle noise, distributed, and continuous-valued representations, and the ability to

extract information conveyed by the temporal order of widely separated inputs facilitated further improvements and development of LSTM variants. Though it was a huge success, still some limitations needed improvements. The application of LSTM variants found massive success in many applications such as unconstrained handwriting recognition (Graves et al., 2009), speech recognition (Graves et al., 2013), and handwriting generation (Graves, 2013).

As LSTMs evolved, so did the optimization techniques. Bengio et al. demonstrated that by fine-tuning the learning rate, the problem of vanishing gradient could be mitigated. Martens and Sutskever (2011) proposed the idea that second-order derivates may vanish at the same time that the first derivatives vanish. In the paper "Generating Text with Recurrent Neural Networks" Martens and Sutskever demonstrated that using second-order optimization methods primarily Hessian-free optimization, RNNs can be trained to perform character-level language modeling tasks. Also, a new RNN architecture called Multiplicative RNN was introduced that was supposed to be better than the standard RNNs in performing language modeling tasks. Training methods and the test corpora used for experiments successfully demonstrated the capacity of new architecture. In the paper, they also commented on the need for more training to achieve better performance, which was mainly restricted due to the need for more computation power.

Thus, to summarize, we can say that the gradient computation was the primary challenge faced by the researchers while training RNNs. As the architecture evolved, the advanced optimization methods came into light that helped to mitigate this issue, but even then, the computation cost was very high. The emphasis on the cost computation in all the papers concurs with this.

Conclusion and Future Work

Through my literature survey study, I reviewed the challenges and the advancements made in the field of RNN research and how the researchers applied them for language modeling tasks. One of

the main problems in training RNNs is learning long-term dependencies in data. Several architectures were proposed to mitigate this problem. Designs like LSTM and MRNNs have shown significant high performance when applied to a variety of tasks such as text generation, handwriting, and speech synthesis. Alternate optimization methods introduced while training RNNs also showed improvements, but there is no standard way to generalize RNN training [14]. Further research in this field is needed, and alternate architecture such as attention-based models is the need of the hour.

References

[1] Bengio, Y., Ducharme, R. & Vincent, P. A neural probabilistic language model. In Proc. Advances in Neural Information Processing Systems 13 932–938 (2001).

[2] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE transactions on neural networks, vol. 5, no. 2, pp. 157–166, 1994.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] J. Martens, "Deep learning via hessian-free optimization," in Proceedings of the 27th International Conference on Machine Learning (ICML10), 2010

[5] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 1017–1024.

[6] I. Sutskever, "Training recurrent neural networks," University of Toronto, Toronto, Ont., Canada, 2013.

[7] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org.

[8] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013

[9] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in International Conference on Machine Learning, 2013

[10] J. Martens and I. Sutskever, “Learning recurrent neural networks with hessian-free optimization,” in Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011

[11] Wikipedia, https://en.wikipedia.org/wiki/Language_model
[12] Andrej Karpathy’s blog, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[13] Deep Learning: LeCun,Bengio,Hinton, Nature2015

[14] H Salehinejad, S Sankar, J Barfett, E Colak and S Valaee, “Recent advances in recurrent neural networks”, 2018