Recurrent neural networks used in natural language processing

Sütő Evelyne

December 22, 2019

Contents

Abstract

The aim of this paper is to present the architecture of recurrent neural network and in more detail a specific recurrent network that has been used in several Natural Language Processing tasks, Long Short-Term Memory Networks.

We will start from a simple neural network architecture then present the additional architectural difference of a recurrent network. We go through the backpropagation algorithm to understand why is it challenging to train a recurrent network. Since our main goal is to use recurrent network for natural language processing problems we introduce the Long Short-Term memory networks which are suitable to process long sequence of natural language sentences. We also cover several improvement of the initial Long Short-Term memory architecture which has been made throughout the years.

Furthermore we will investigate the Encoder-Decoder architecture's specific properties and description to show how they are usually used in these types of problems.

Lastly we present a few experiments done in different NLP taks using the above mentioned elements.

1 Introduction

Natural Language Processing (NLP) problems have been researched heavily in the last few decades, however its impact has never been so great as it is today. This is mostly because we are trying to automate more and more processes. However this automation process becomes very difficult once the understanding of natural language is needed. In those cases, such as customer support chat-bots, we need to solve problems such as text to speech recognition, language modelling, text generation and more. One architecture in the field of machine learning which is suitable for these problems is Recurrent Neural Networks (RNN).

The aim of this paper is to provide a presentation of the architecture of recurrent neural networks. As mentioned above, our main interest is to explore the domains of NLP and for that reason we will introduce a specific variant of the RNN architecture, the Long Short-Term Memory networks (LSTM).

In order to have a better understanding of the structure of RNN first we discuss simple neural networks then introduce what does a LSTM has in plus. We also explore the Backpropagation algorithm and it's problems when training recurrent or deep networks.

Furthermore we will provide an overview of a popular model used for NLP tasks: Encoder-Decoder model 2.7, which usually uses two separate RNN for the encoder and the decoder module. In addition we will have a look into two algorithms 2.8 which will help us decode the predictions given by our model: Greedy algorithm and Beam search algorithm.

After this theoretical background we will focus on a variety of applications in which this architecture can be used 3. We will see what additional changes each applications adds to achieve high performance in each tasks.

The tasks mentioned in this paper will be: neural machine translation 3.1, text summarization 3.3, speech recognition 3.2 and paraphrase generation 3.4.

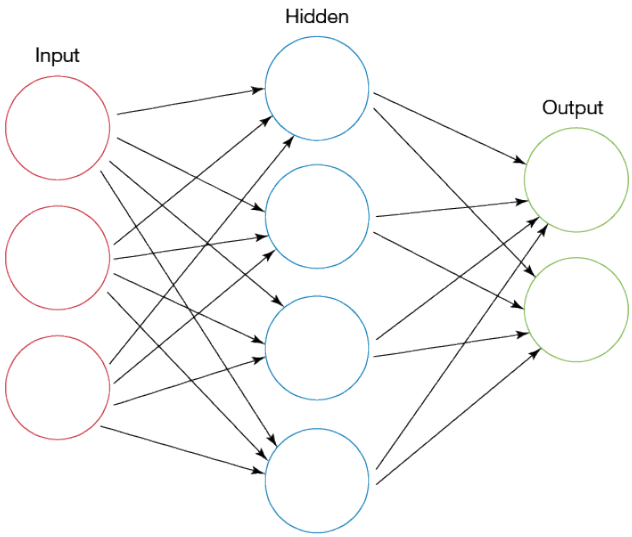

Figure 1: The representation of a simple Artificial Neural Network with one hidden layer. Image source: https://en.wikipedia.org/wiki/Artificial_neural_network

2 Theory

2.1 Neural Network

First let us consider a simple *neural network*. (Figure 1) In this case we only have one hidden layer but this can be extended with as many hidden layers as needed for our purposes in that case we have *deep neural networks*.

A standard neural network consists of many simple, connected processors called neurons, each producing a sequence of real-valued activations. Some of these neurons get activated by the environment and some are activated through weighted connections from other neurons. The learning or training of the network consists of finding weights for which the network approximates the desired hypothesis for our task. [Schmidhuber, 2015]

2.2 Backpropagation

The *backpropagation algorithm* is used to find the best approximation of the desired weights for the network.

The algorithm is based on gradient descent which determines a weight vector which minimizes the error by starting from an initial random vector which will be modified repeatedly in small steps in the direction that produces the steepest descent. Let us consider a simple representation of the error as:

$$E = \frac{1}{2}\sum_{n=1}^{N}(t_d - o_d)^2 \tag{1}$$

Where t_d is the targeted output and o_d is the output given by our network. In order to find the direction in which we can minimize this error we need to calculate the derivative of this error function with respect to our network weights. This derivative will give us the direction that produces the steepest increase in the error.

As a result our weight updates which minimize our error will be:

$$w_i = w_i - \eta \frac{\partial E}{\partial w_i} \tag{2}$$

Where:

w_i: is the weight

E: is the error of the network (loss)

η: is a positive real number called the learning rate

Since we mostly use architectures with more than one layers we need to propagate the error backwards and calculate the partial derivatives respectively with the help of the chain rule. More detailed description of the backpropagation algorithm in: [Mitchell, 1997]

2.3 Recurrent neural network

Recurrent neural networks are a subset of neural networks which uses the output unit at time t as the input to another unit at time t+1. For this reason RNNs are usually used for sequence learning problems.

As seen on Figure 2 we can unfold this recurrent network to understand it's structure better. We have several copies of the network which are connected between each others.

The input is represented as a sequence x = (x_1, . . . , x_T) from which the RNN computes the hidden vector state h = (h_1, . . . , h_T) and the output y = (y_1, . . . , y_T) by iterating the following equations from t = 1 to T:

$$h_t = \mathcal{H}(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{3}$$

$$y_t = W_{hy}h_t + b_y \tag{4}$$

where the W represents the weight matrices (e.g. W_{xh} is the input-hidden weight matrix), the b terms denote bias vectors (e.g. b_h is hidden bias vector) and $\mathcal{H}$ is the hidden layer's activation function e.g. tanh or sigmoid function. [Graves et al., 2013]

Since RNNs contain a finite number of networks we may train the unfolded network using backpropagation and updating the weights with the mean value of the corresponding weights of the copies. [Mitchell, 1997]

2.4 The problem in the recurrent network architecture

When training an RNN there is two problems we need to consider: *vanishing and exploding gradients*.

As mentioned in section 2.3 these networks can be trained by unfolding this model and using backpropagation on this unfolded network. However with standard activation functions the cumulative backpropagated errors either shrink rapidly, or grow out of bounds. In fact, they decay exponentially in the number of layers, or they explode. This is also known as the

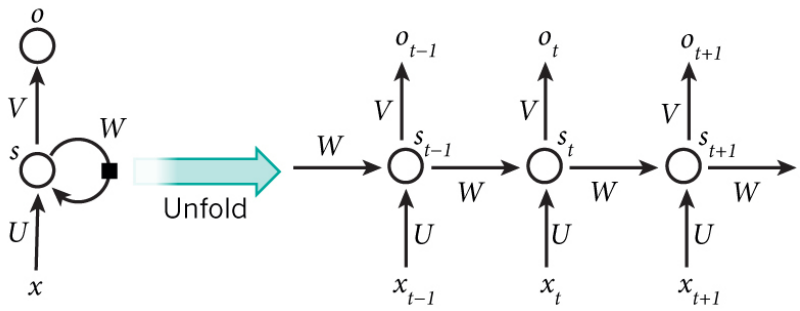

Figure 2: The representation of an unfolded Recurrent Neural Network .
Image source: https://magenta.tensorflow.org

long time lag problem. This makes it challenging to train very deep neural networks because they can't find the correlation between distant elements. [Schmidhuber, 2015]

There are several solutions to avoid these problems. We may use a threshold value for our gradients and choose to scale it whenever the value goes over that specific threshold. This may solve the problem of exploding gradient. While the solution of the vanishing gradient problem lies in using LSTM cells instead of regular cells thus creating a Long Short-Term Memory network. [Pascanu et al., 2013]

2.5 Long Short-Term Memory Cell

To avoid vanishing gradient an architecture was proposed which allows constant error flow, through special, self connected gate units. Since the goal is to have cells that can store information from previous states and are able to ignore certain information this could lead to conflicts if we use simple cells.

To resolve these conflicts LSTM (Figure 3) uses a multiplicative input gate unit which protects the information content in j from irrelevant information and a multiplicative output gate unit which protects other units from the current state's irrelevant information. [Hochreiter and Schmidhuber, 1997]

The memory cells internal structure consists of a number of connected computational units, including the sigmoid function, the tanh function, and the gating function, which are connected in a graph structure. (Figure 3) The fact that these units are differentiable ensures the memory cell as a whole can be used in conjunction with BPTT using the chain rule as a connecting principle. [Bayer et al., 2009]

The architecture uses a sigmoid gate to filter the input from unnecessary information, the next sigmoid together with the tanh gate defines what new information should be added or kept from the previous states and the last two gates will define what the final output of the cell should be.

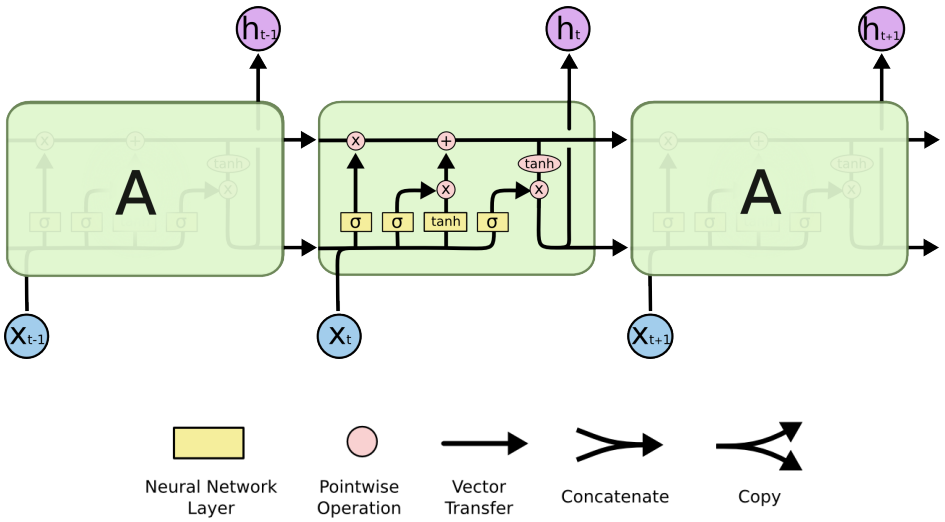

Figure 3: An LSTM network with a cell that contains four gates. Image Source: http://beta.cambridgespark.com/courses/jpm/05-module.html

These elements can be calculated through:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{5}$$

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{6}$$

$$\tilde{C}_t = tanh(W_{\tilde{C}} * [h_{t-1}, x_t] + b_{\tilde{C}}) \tag{7}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{8}$$

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{9}$$

$$h_t = o_t * tanh(C_t) \tag{10}$$

While these types of networks do well in tasks where long lags are present between the events they do have their limitations as well which were even discussed when they were first published in [Hochreiter and Schmidhuber, 1997].

- They do increase the number of weights that are needed to be used in our architecture.
- In some cases training LSTM networks may take longer than solving a problem by random weight guessing in trivial cases.
- They struggle to solve problems such as the "strongly delayed XOR" problem, since only storing one of the inputs does not help in the error reduction.

2.6 Possible ways to improve LSTM

As it is described in the [Bayer et al., 2009] publication this form of the LSTM cell has been developed in incremental steps.

The [Greff et al., 2017] publication provides a detailed overview of how different variants of LSTM networks have been developed throughout the years. The first variation of LSTM introduced in [Hochreiter and Schmidhuber, 1997] included only input and output gate with an internal state. The first improvement was proposed in [Gers et al., 1999] which has recognised the limitation of this architecture, there was no way for the LSTM to reset its own internal state thus enabling it to forget unnecessary information. To solve this problem they introduced the forget gate, which allowed the cells to reset their states adeptively. They later expanded this model with peephole connections, connections from the cells to the gates, which can control the gates in order to make precise timings easier to learn. They also improve the training of the original LSTM networks by replacing the mixture of Real Time Recurrent Learning and Backpropagation Through Time with full Backpropagation Trough Time training. [Greff et al., 2017]

Other researches to evolve the structure of LSTM cells used evolutionary algorithms [Bayer et al., 2009], where the researchers viewed the LSTM cell itself as a network and used it to create RNN on which the evolutionary algorithm was used to mutate these cells and calculate the fitness function which is applied to select the best entities. They came to the conclusion that the cell structures generated this way were able to outperform simple LSTM cells in several tasks such as T-Maze test from Reinforcement learning or the learning of context-free languages. This proves that these cells can be used in more general problems as well (e.g. reinforcement learning).

Another interesting approach in the literature is to create hybrid models using LSTM networks with other well known architectures. An example of such case is the Convolutional LSTM network [Xingjian et al., 2015]. In this paper the researchers introduce this novel approach to solve the problem of Precipitation Nowcasting. This architecture recognizes the shortcomings of simple LSTM of making correlation between spatial attributes of the dataset. As we can notice from their experiments this model outperforms the standard LSTM architecture not just in Precipitation Nowcasting problem but other application as well e.g. Moving-MNIST Dataset.

2.7 Encoder-Decoder Model

In order for us to understand the experiments that will be presented in this paper first we need to have a better understanding of the Encoder-Decoder model which will be used in many of these examples. This model is also referred to as the sequence to sequence model. One simple representation of this model can be seen on Figure 4

Since the input can be of varying size which could not be modelled by a recurrent network a simple strategy is to map the input sequence to a fixed sized vector using one RNN and then map the vector to the output sentence with another RNN. Since this model might need to learn really long sentences we Long Short-Term Memory Networks are often used in these problems. The idea is to use an LSTM to read the input data one time step at a time to obtain a fixed dimensional vector representation, and then to use another LSTM to

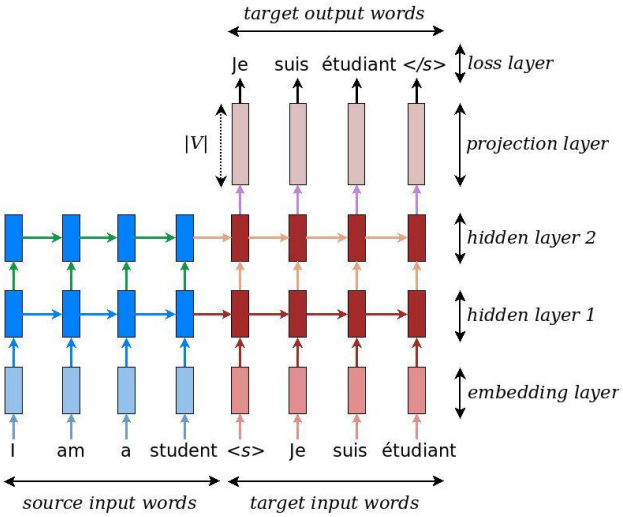

Figure 4: Neural machine translation example of a deep recurrent architecture proposed by for translating a source sentence "I am a student" into a target sentence "Je suis tudiant". Here, "s" marks the start of the decoding process while "/s" tells the decoder to stop. Image source: https://github.com/lmthang/thesis/blob/master/thesis.pdf

extract the output sequence from that vector. So the LSTM's goal will be to estimate the probability of $P(y_1, y_2, .., y_l|x_1, x_2, .., x_t)$ where x is the input sequence and y is the output sequence whose length might be different. It is also required for the input sequence to end with a special *Stop* character. The model is based on two components: the first one for the input sequence the a second for the output sequence. [Sutskever et al., 2014]

It is also important to mention that the researchers in [Sutskever et al., 2014] have found that reversing the source sentence can lead to better results even though they can't really explain why that might be.

The result of the second layer will be a matrix of probabilities from which we can decode our prediction using two approaches: maximum likelihood method and beam search algorithm which will be discussed in 2.8.

As we can see on Figure 4 a conventional Encoder-Decoder model usually has an Embedding layer (in NLP problems) which translates the input sentence to a dense vector representation using the Vocabulary that we provide in the beginning. The output of the embedding is then fed to the encoder model which will return the input for the decoder. The output from the decoder layer then can be processed to retrieve the prediction.

2.8 Decoding Algorithms

In the literature there is two main algorithms which are used to extract the output of the decoder.

2.8.1 Greedy Decoding

Greedy decoding is one of the simplest approaches for decoding sentences and it is based on conditional possibility calculations.

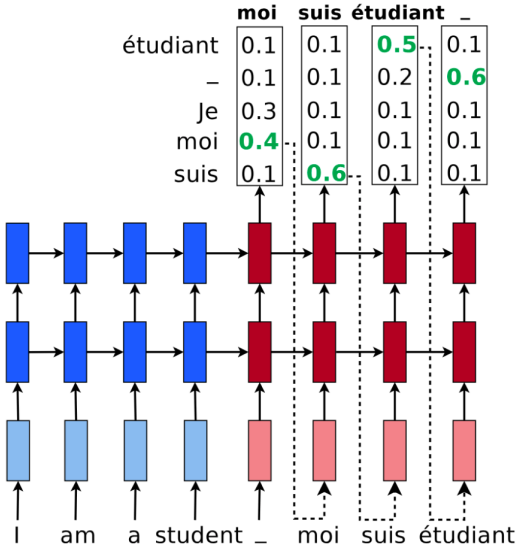

Figure 5: Greedy decoding algorithm example of a deep recurrent architecture proposed by for translating a source sentence "I am a student" into a target sentence "moi suis etudiant" Image source: https://github.com/lmthang/thesis/blob/master/thesis.pdf

In greedy decoding, we follow the conditional dependency path and pick the symbol with the highest conditional probability so far at each node. This is equivalent to picking the best symbol, one at a time, from left to right in conditional language modelling. [Gu et al., 2017]

However this does not give us the best results because it only takes into consideration the element at previous time step which means that if it makes a wrong prediction every prediction made after that will be based on false information.

2.8.2 Beam-Search algorithm

Beam search keeps $K > 1$ hypotheses, unlike greedy decoding which keeps only one during decoding. At each time step t, beam search picks K hypotheses with the highest scores

$$\prod_{i=1}^{t} p(y_t|y < t, X)$$

When all the hypotheses terminate (outputting the end-of-the sentence symbol), it returns the hypothesis with the highest log-probability. Despite its superior performance compared to greedy decoding, the computational complexity grows linearly w.r.t. the size of beam K, which makes it less preferable especially in the production environment. [Gu et al., 2017]

The reason these algorithms and model has been presented in detail is because many experiments that will be presented uses them.

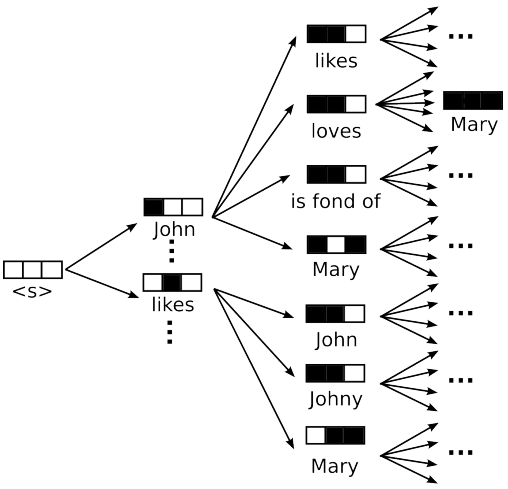

Figure 6: Beam-Search algorithm At each time step each node is expanded with the K most promising new nodes.
Image source: http://mttalks.ufal.ms.mff.cuni.cz/index.php?title=File:Beam-search.png

3 Experiments

3.1 Neural Machine Translation

One of the fields where sequence to sequence models were successfully applied and has yielded state of the art results if machine translation.

One of these researches were conducted for English-French translation in [Sutskever et al., 2014] where researchers have used two connected LSTM models. They have concluded that deep LSTMs can significantly outperform shallow LSTMs. Another important element introduced by that was the fact that reversing the source sentence can improve the model's accuracy. Furthermore they have shown that LSTM sequence to sequence models are able to map even very long sentences to the translation language successfully. This simple unoptimized model was able to produce state of the art results with relatively short training. For example models which use Statistical Machine Translation methods with neural networks produce 37 BLEU score while this models best result was 36.5 BLEU score.

BLEU is an automatic evaluation metric used for machine translations tasks. The way that Bleu metrics work is to compare the output of a machine translation system against reference human translations. The primary reason that Bleu is viewed as a useful stand-in for manual evaluation is that it has been shown to correlate with human judgements of translation quality. [Callison-Burch et al., 2006]

Later on this model's ([Sutskever et al., 2014]) shortcomings are addressed in [Wu et al., 2016] to improve Google's translation system. The main shortcomings of this systems recognised by them was the expensive training, lack of robustness, poor translation when rare words are present and the issue to NMT systems in practical deployments and services. They propose many changes to the architecture itself such deeper encoders and decoders (8 layers), Attention mechanism added between the encoder and decoder module, they introduced residual connections between the LSTM layers to speed up their training and they use

Bi-Directional Encoder to get a better context for the translation. The problem of rare word translation has been solved partially by copying the rare word entirely in the target sentence since these words usually represent names and they also use a more intelligent approach with sub-word units implementation. The evaluation of their model is done on two translation tasks: English-Frech and English-German, where on the English-French dataset the results improved by 1 BLEU score while the English-German with 0.4 BLEU score. It is also presented that such a model can easily be used in production as well improving translation errors by more than 60% even for such challenging tasks as English-Chinese translation.

3.2 Speech recognition: Speech to text translation

Another subfield where LSTM can be used is for speech recognition task.

As used by the authors from [Graves and Jaitly, 2014]. The authors have chosen to use an LSTM to exploit the long range of contexts from the training data and the LSTM's capability to store information. Instead of using a simple RNN they have chosen to use a Bidirectional RNN to exploit the future context as well, not just previous ones. A Bidirectional RNN processes the information in both directions with two separate hidden layers, then calculating the output based on these informations. In order to improve the training of the network they use Connectionist Temporal Classification function which allows the RNN to be trained sequence transcription without requiring prior alignment between target and input data. Their chosen decoding algorithm to decode the sentences was the Beam Search algorithm. In order to train their model they have chosen the Wall Street Journal corpus. The experiments were then scored on word error and character error rate. Their experiments conclude that even though it does not outperform the baseline model, considering that this model does not need data preprocessing it can be viewed as a successful model. Their best scores on the 14 hour dataset was 13.5 while the baseline is 9.4 and on the 81 hour dataset their result was 8.2 while the baseline was 7.8.

3.3 Text summarization

Abstractive text summarization is the task of generating a headline or a short summary consisting of a few sentences that captures the salient ideas of an article or a passage. [Nallapati et al., 2016]

The researchers in [Nallapati et al., 2016] view this problem as mapping a sequence of data to another sequence. As a result they have chosen the Encoder-Decoder model with a bi-directional GRU-RNN as the encoder and a uni-directional GRU-RNN. They also use an Attention mechanism between the two modules. They also address the problem of rare words by using a so called Switching Generator-Pointer model. In this model, the decoder is equipped with a switch that decides between using the generator or a pointer at every time-step. If the switch is turned on, the decoder produces a word from its target vocabulary in the normal fashion. However, if the switch is turned off, the decoder instead generates a pointer to one of the word-positions in the source. The word at the pointer-location is then copied into the summary. The switch is modelled as a sigmoid activation function over a linear layer based on the entire available context at each time step. [Nallapati et al., 2016] In their experiments they have used 3 datasets: Gigaword corpus, DUC corpus and

CNN/Daily Mail corpus. Their model has outperformed the state of the art results on DUC and CNN/Daily Mail corpus.

3.4 Paraphrase generation

Paraphrasing, the act to express the same meaning in different possible ways, is an important subtask in various Natural Language Processing (NLP) applications such as question answering, information extraction, information retrieval, summarization and natural language generation. [Prakash et al., 2016] Paraphrase generation can be used to improve many NLP tasks such as question answering, translation, text summarization just by generating new forms of the inputs.

Even though paraphrase generation can be used to improve many tasks it was just recently started to spark interest in researchers.

In [Prakash et al., 2016] the authors try to solve this problem by using an Encoder-Decoder model with Deep LSTMs with stacked residual connections. This model is evaluated on three very different datasets. The PPDB is a paraphrase dataset which includes only various short paraphrases, WikiAnswers with 18M question pairs and MSCOCO which contains image captions. The authors have compared 3 other models with their proposed one. On each corpus their model outperformed the other 3: sequence to seqeunce, sequence to sequence with attention and Bi-directional LSTMs.

4 Conclusion

We have seen an overview of how a basic neural network works and what is the aspect that a RNN offers to solve sequence problems.

We also presented an overview of a RNN and researched what are their weaknesses that make them a slightly worse approach for NLP. We gained some insight on the architecture of LSTM networks which introduced the solution for the drawback of RNN. We introduced some researches which have contributed to the improvement of the original LSTM to reach its current state. From the insight of the architecture of the LSTM and the recent researches that has been made to evolve this model even further we can safely declare that it does solve the initial shortcomings for which they were introduced and they are worth to research further.

We should also mention that while this architecture has been used for many years for sequence modelling some recent study shows that it might not be the most efficient for this task. In [Elbayad et al., 2018] the researchers introduced a new approach with using a 2D Convolutional network for sequence modelling and have showed that even though their model is simpler than the LSTM and uses less parameters it was able to yield results that can compete with the traditionally used models.

However, having seen the various experiments done in this field we can conclude that the use of recurrent neural network is very popular in NLP tasks. One of the main reasons for that is the LSTMs power of capturing long term dependencies in data. The results reported in these researches also show that in most tasks in order to get successful results there is a need to alter the baseline model to the task in various ways e.g. adding Attention mechanism,

using residual connections. However after introducing the needed alterations they can yield in state of the art results as seen in these experiments.

References

[Bayer et al., 2009] Bayer, J., Wierstra, D., Togelius, J., and Schmidhuber, J. (2009). Evolving memory cell structures for sequence learning. In *International Conference on Artificial Neural Networks*, pages 755–764. Springer.

[Callison-Burch et al., 2006] Callison-Burch, C., Osborne, M., and Koehn, P. (2006). Re-evaluation the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*.

[Elbayad et al., 2018] Elbayad, M., Besacier, L., and Verbeek, J. (2018). Pervasive attention: 2d convolutional neural networks for sequence-to-sequence prediction. *arXiv preprint arXiv:1808.03867*.

[Gers et al., 1999] Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

[Graves and Jaitly, 2014] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772.

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

[Greff et al., 2017] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232.

[Gu et al., 2017] Gu, J., Cho, K., and Li, V. O. (2017). Trainable greedy decoding for neural machine translation. *arXiv preprint arXiv:1702.02429*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill.

[Nallapati et al., 2016] Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. (2016). Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

[Prakash et al., 2016] Prakash, A., Hasan, S. A., Lee, K., Datla, V., Qadir, A., Liu, J., and Farri, O. (2016). Neural paraphrase generation with stacked residual lstm networks. *arXiv preprint arXiv:1610.03098*.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

[Xingjian et al., 2015] Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810.