UDC 004.8

O. Iosifova, I. Iosifov, O. Rolik

TECHNIQUES AND COMPONENTS FOR NATURAL LANGUAGE PROCESSING

Abstract: A dramatic change in the abilities of language models to provide state of the art accuracy in a number of Natural Language Processing tasks is currently observed. These improvements open a lot of possibilities in solving NLP downstream tasks. Such tasks include machine translation, speech recognition, information retrieval, sentiment analysis, summarization, question answering, multilingual dialogue systems development and many more. Language models are one of the most important components in solving each of the mentioned tasks. This paper is devoted to research and analysis of the most adopted techniques and designs for building and training language models that show a state of the art results. Techniques and components applied in creation of language models and its parts are observed in this paper, paying attention to neural networks, embedding mechanisms, bidirectionality, encoder and decoder architecture, attention and self-attention, as well as parallelization through using Transformer. Results: the most promising techniques imply pre-training and fine-tuning of a language model, attention-based neural network as a part of model design, and a complex ensemble of multidimensional embeddings to build deep context understanding. The latest offered architectures based on these approaches require a lot of computational power for training language model and it is a direction of further improvement.

Key words: NLP, language model, embeddings, RNN, GRU, LSTM, encoder, decoder, attention, Transformer, transfer learning, deep learning, neural network.

Introduction

Natural Language Processing (NLP) is computer comprehension, analysis, manipulation, and generation of natural language. NLP covers a lot of different applications like machine translation, speech recognition, optical character recognition, part of speech tagging, information retrieval, summarization, question answering, dialog systems building and many more. Since the last decade, there have been a great number of breakthroughs towards making machines understand the language (text or voice) better, that raised enormous interest in this field of scientific research. The highest goal for all scientists working in the area of Natural Language Processing is to build such techniques that will allow computers to comprehend natural language as text or voice at a human level which is not reached yet. Once achieved, computational systems will be able to understand and generate accurate human-like language, be it a text or an audible language. This paper analyses widespread

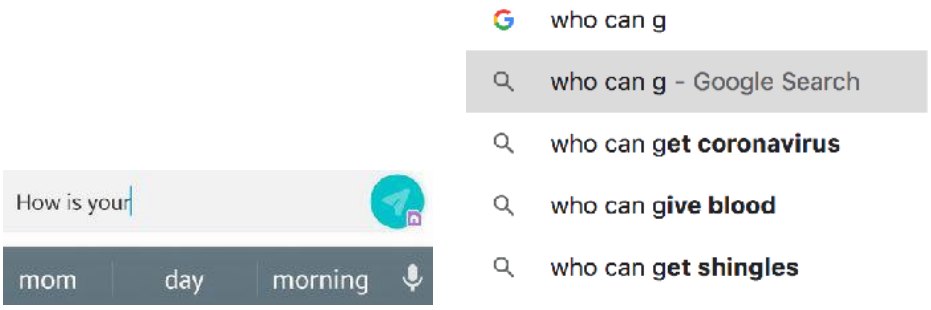
© O. Iosifova, I. Iosifov, O. Rolik

techniques and components in building language models to give a scientist thorough information for further research and improvement.

There have been a lot of discoveries of language models from computational linguistics scientists, and those new techniques showed great results on specific tasks. But when it comes to the broad spectrum of NLP tasks solved by the same language model not many show the same high results. The recent state of the art architectures (BERT, RoBERTa, Transformer-XL, XLNet, and others) leverage the following approaches: contextual embeddings, bidirectionality, encoder-decoder architecture, Attention mechanism and Transformer, pre-training modeling plus fine-tuning.

Nowadays, all best state of the art results in the NLP area achieved using the Neural Networks approach of learning and finding weights (parameters), which can be applied to inputs to output prediction as outputs. Hence, below will be covered high-level details on each of the major concepts from NLP that are incorporated in almost any current state of the art and production-based solutions.

Despite the fact of neural networks existence for several decades, it was not a widely applied technique in the NLP area. That was changed and neural language models appeared. The main driver of the new family of neural networks was an NLP task called Language Modeling. Language models are applied to many tasks including Part of Speech tagging, parsing, machine translation, speech to text conversion, relationship extraction, etc. With the help of language modeling, we can solve the task of the next word in a sequence prediction as shown in Figure 1.



a *b*

Figure 1. – A demonstration of a word sequence prediction in mobile phone keyboard application (*a*), Google search (*b*)

The structure of this article includes language models architecture observation in section 2; Word and contextual embedding techniques are described in section 3; Encoder-Decoder – section 4; Neural Networks applied as a part of Encoder and

 ISSN 1560-8956

Decoder are observed in section 5. Conclusion in section 6 provides further promising areas of research in NLP.

The most common approach to NLP downstream tasks

NLP downstream tasks (machine translation, sentiment analysis, question answering, part of speech tagging, and many more) usually are solved with a number of different approaches chosen for a specific task [1, 2]. Generally, it comes to supervised learning on task-specific datasets, which is quite consuming in terms of research hours and computational resources [4, 5]. In addition to these inconveniences, systems that are built with this approach are very sensitive to task specifications and changes in data distribution. Current trends move towards unsupervised universal models [5, 6] and transfer learning as pre-trained models with further fine-tuning [7, 8].

The techniques and components that are offered for observation in this article corresponds to current trends and groundbreaking achievements in NLP [9, 10]. It outlines the architectures of the state of the art pre-training models [11–15].

Everything starts with the ready for training or test data. Input data runs through a number of techniques to result in word embeddings [16–19] or contextual embeddings that could be described as multidimensional word knowledge embeddings. Despite the use of the term ‘word’, readers shouldn’t be confused. Word embeddings are a form of a vector that can be based on characters, subwords, words, sentences or even longer sequences each of which is called a token. Contextual embeddings [20–22] are used as input for Encoder which forms a Context Vector and forwards it to Decoder. A decoder in its turn forms a set of probabilities necessary to figure out an output. Example of the described approach called sequence to sequence is shown in Figure 2.

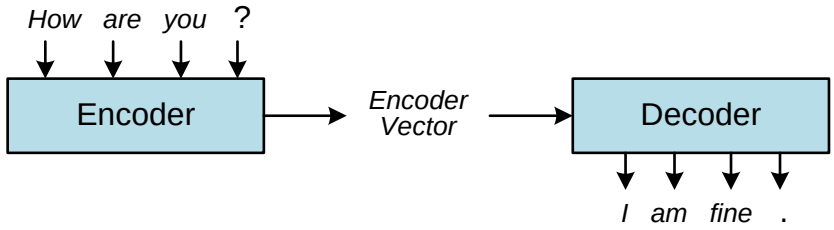


Figure. 2. An example of an Encoder-Decoder architecture

Input sequence of words [“How”, “are”, “you”, “?”] go through the Embedding layer whose main task is not to generate predictions, but to encode input sequences and present it in output Encoder Vector. Decoder takes the output of Encoder and uses

it to generate output predictions (in this case generate the best answer to the input question).

Word and contextual embeddings approaches

For a neural network to be able to complete its task there is a necessity to provide numerical token representation of input sequence. Word embeddings techniques create vectors out of tokens. Vectors comparison results in tokens semantic similarity [16, 19].

Embedding techniques such as GloVe [17] and Word to vector [18] explain the concept of modeling input sequence through representation.

The main idea and task is to represent and map words (documents, phrases, context, a piece of a word, or a character) as a vector of numbers to use probability distributions or likelihoods of tokens in language corpora to separate semantic similarity categories [23]. Hence different words with similar meaning will have similar vectors and different by meaning groups of words should be separable in vector space.

The underlying idea that "a word is characterized by the company it keeps" was popularized by Firth. Currently, the area of representation of input sequence advances far ahead of initial papers and new approaches appear [24].

Contextual embeddings [16, 20–22] create a representation for each token taking into account its context, meaning getting information of a token usage in different contexts and encode knowledge that is transferable to a number of other languages.

There are also ensemble possibilities in the concept of embedding as shown in [14]: for a given token, embeddings of a token itself, position and segment where it was used are summed into a resulting representation.

High-level Encoder-Decoder architecture to solve Natural Language Processing tasks with different input and output sequences length

The neural network model Encoder-Decoder significantly improved the performance of Language Models [3, 24]. Quite simple Recurrent Neural Network (RNN) architecture to process input and output sequences of variable lengths was offered as shown in Figure 3. Input sequence of words ["How", "are", "you", "?"] go through the Embedding layer to get numerical representation, after which numerical representation goes sequentially to RNN. RNN process input embeddings sequentially (from left to right) passing to the next time stamp RNN hidden state calculated in the

current time stamp RNN. After all inputs proceed to the final time stamp, the final time stamp RNN produces output representing all input sequences in one Hidden state. This part called Encoder as the main task is not to generate predictions, but to encode input sequences. After Encoder finishes to encode, Hidden State passes to Decoder which task is to decode and generate predictions based on input Hidden State. Decoder process sequentially taking as input to each current time stamp output activation of previous time stamp RNN and output prediction of previous time stamp RNN. For the first time stamp it takes a <BOS> token ("beginning of sentence") as prediction of the previous layer. Decoder generate predictions until it generates <EOS> token (depending on implementation can be until some length or different parameter).

The strongest part of this approach is the ability to train an end-to-end model right on the source and target data as well as the possibility to handle input and output sequences of different length. So that it resolves the problem of different lengths of an input and an output sequence in Neural Machine Translation.

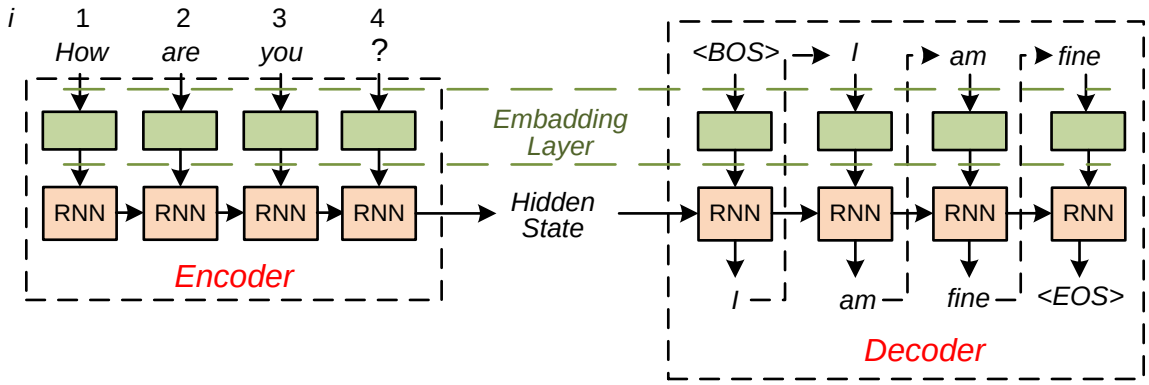


Figure. 3. An illustration of Recurrent Neural Network Encoder-Decoder with embedding layer

The encoder-Decoder architecture consists of two Recurrent Neural Networks or more often Long Short-Term Memory or Gated Recurrent Unit to avoid the problem of vanishing gradient covered later in this article. The encoder encodes all input sequences and stores all information in Context or Encoder Vector (in simplest architecture last hidden state used) that is input to Decoder which decodes by result predictions.

Neural Networks for Natural Language Processing tasks

Progress in the application of Neural Networks to Natural Language Processing [25, 26] tasks bring huge improvements in both science and business areas.

Main Neural Networks applied to solve NLP tasks will be covered below.

1. Early adoption of Deep Learning for Natural Language through Recurrent Neural Networks

Recurrent neural networks or RNNs [27, 28] is the main starting point in the deep learning NLP area. Deep neural networks uncover a second life for RNNs. Strong RNN advantage for the NLP area is that RNN can store the conditions of all cells that processed language data before sequentially.

The main idea behind RNNs [29–31] is very simple, the network takes an input vector *X* and produces an output vector *Y*. As shown in Figure 4, each RNN cell takes as input current x_t and previous hidden state (activation) h_{t-1}. It learns weights (parameters) W_h, W_x, and bias b_a through the weights learning process. At each iteration of Forward Propagation nonlinear *activation function g* such as tanh (or rarely ReLU) applied to calculate output hidden state (activation) h_t, as shown in expression (1)

$$h_t = g\left(W_h h_{t-1} + W_x x_t + b_a\right). \quad (1)$$

If output predictions needed by task then activation function *g* (or softmax function) with learned weights W_y and bias b_y might be applied to current output h_t to make output prediction y_t, as shown in expression (2)

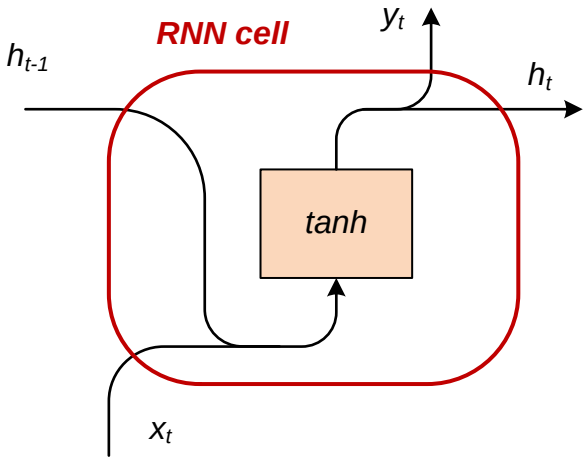


Figure. 4. RNN cell design

$$y_t = g\left(W_y h_t + b_y\right). \quad (2)$$

Especially important for NLP areas is that output vector's contents are calculated not only by the one current input but based on the entire history of inputs that network processed in the past. As shown in Figure 5, RNN cells take the output of the previous cell as an input to the current cell, and the previous cell contains

 ISSN 1560-8956

information of its previous cell and so on. This type of connection is called a recurrent connection.

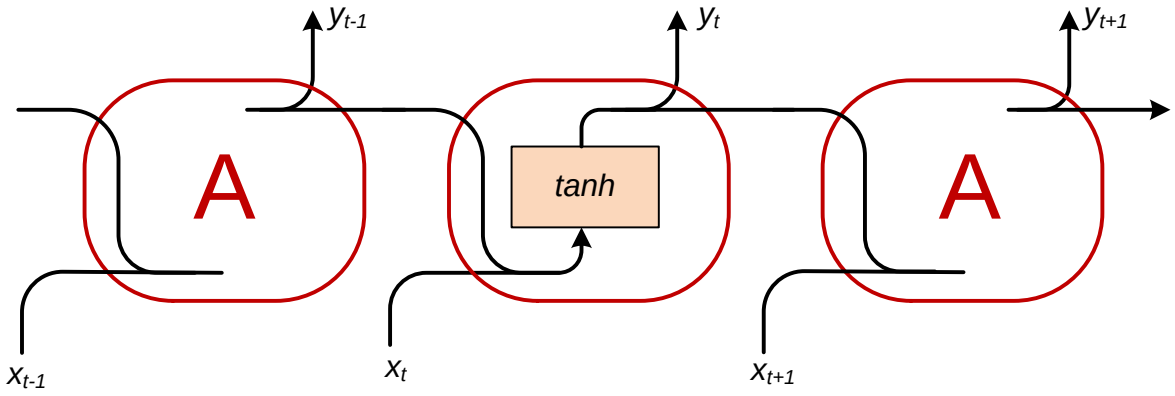


Figure. 5. Recurrent neural network design

Despite the wide adoption, RNN possesses significant drawbacks [32, 33]. Unidirectional learning leads to the problem that the model can't rely on information from the later part of a sequence while working on the beginning of a sequence. For RNNs it is hard to capture mid and long-term connections/dependencies inside a sequence – this issue is known as long-range dependencies problem or Gradient Vanishing [34]. This was a triggering point to search for a solution that resulted in further useful findings like GRU and LSTM.

2. Improving medium-range memory of Neural Networks for Natural Language Processing tasks using Gated Recurrent Unit and Long Short-Term Memory

In response to medium and long-range dependency problems researchers propose two architectures, with the core idea of Cell State (kind of residual connection).

The main idea in Gated Recurrent Unit (GRU) [35, 36] is to capture long-term dependencies by adding *Memory Cell* (C_t) which in GRU is equal to hidden state (activation) h_t, $h_t = (1-z) * h_{t-1} + z_t * h_t$. And each time stamp cell considers rewriting this cell with *Candidate Value* (h_t, $h_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$, using two Gates described by equations (*Update Gate* (*z*), $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$, and *Reset Gate* (*r*), $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$) as shown in Figure 6. Update gate (*z*) takes a value between 0 and 1 (most times close to 0 or 1), computed by application of sigmoid activation function to current time stamp input x_t and previous time stamp hidden state (activation) h_{t-1} with learned weights (parameters) W_z through the weights learning

process. This Update Gate is the main decision-maker of updating the hidden state as shown in equations. Update Gate decides how much information from the previous time stamp should be saved for the future. Reset gate, on the other hand, decides how much information from the previous time stamp should be removed.

These Update and Reset Gates are the key concepts behind GRU and dealing with dependencies problems of basic RNNs.

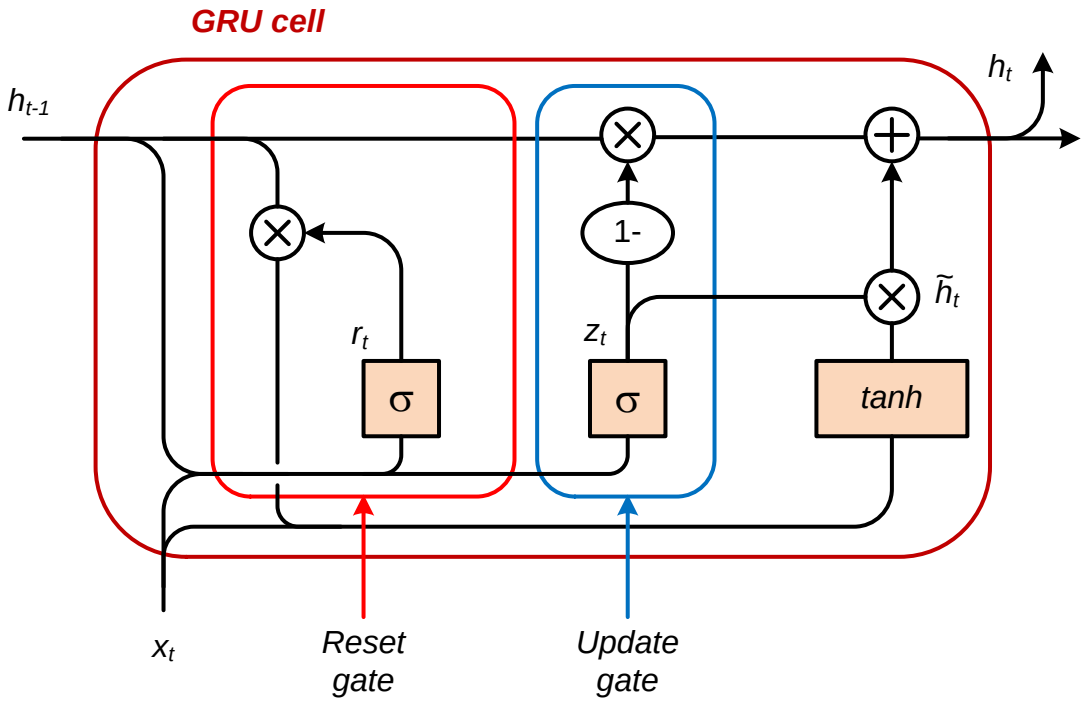


Figure. 6. A scheme of GRU cell

Another type of architecture that can capture mid-term dependencies, even more powerfully than GRU, is Long short-term memory (LSTM) [37–40]. In a difference to GRU that has two gates, LSTM possesses three gates. Important concept in LSTM is that *Memory Cell* (C_t) is not anymore equal to output hidden state (activation) h_t. Output hidden state (h_t) of the current time stamp in LSTM carry on to the next cell not alone but with *updated Memory Cell value* (C_t).

LSTM, also, uses two separate gates (*Update Gate* and a *Forget Gate*) to update Memory Cell value (C_t), instead of using single Update Gate in GRU (that either keep OR forget previous memory cell value). And instead of using Reset Gate in Candidate Value, it uses element-wise multiplied with *Memory Cell value* (C_t) as shown in Figure 7.

Input Gate (*i*), $i_t = \sigma(x_t U^i + h_{t-1} W^i)$, decides which information crucial to keep and *Forget Gate* (*f*), $f_t = \sigma(x_t U^f + h_{t-1} W^f)$, decides which and how much

 ISSN 1560-8956

information not to keep, in other words, to forget (which might be intersecting or not with input gate). Input and Forget gates do this using previous time step hidden state h_{t-1} and current input x_t. Both gates using sigmoid activation function, which gives possibility in most cases to have values of gates either close to 0 or to 1.

Usage of separate *Update Gate* and *Forget Gate* to calculate *Memory Cell value* (C_t), $C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$, gives Memory Cell the possibility not only to store new information in the current time step *Memory Cell* (C_t) by using *Candidate Memory Cell* ($\tilde{C}_t$), but also the option to keep some amount of information from previous time step *Memory Cell* (C_{t-1}). Output Gate (o), $o_t = \sigma(x_t U^0 + h_{t-1} W^0)$, at the end uses to calculate the current time step output hidden state $h_t = \tanh(C_t) * o_t$, based on updated Memory Cell value calculated before.

The state of a cell is absolutely straight forward. It flows down the whole unit with minor linear changes. This is why two proposed architectures were very good at memorizing long-term dependencies [41].

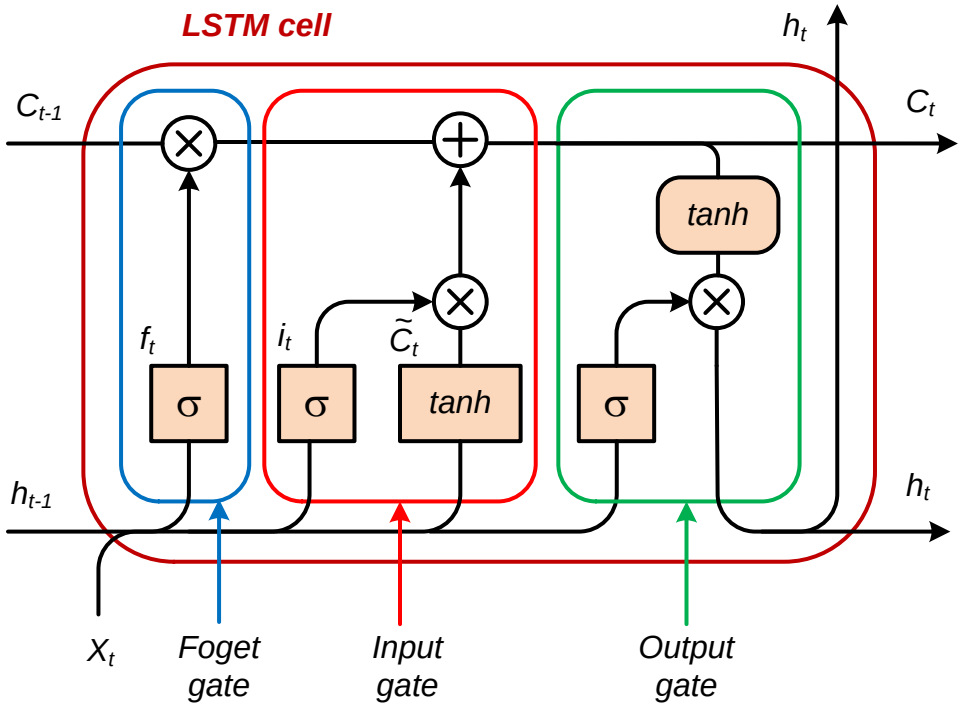


Figure. 7. A scheme of LSTM cell

These networks are quite consuming for computational resources. Moreover, this type of architecture can't be parallelized: hence it is very expensive to train on a big corpus of data. And despite the fact it works much better with longer sequences

there is a noticeable loss in sequences with more than 20 words. In [34] demonstrated BLEU score results of different RNN networks.

Retrospectively application of RNN, GRU, and LSTM was the major stage in modern NLP, that significantly affected future development of the area. The architecture of the three major networks presented in Figure 9. It reflects how complexity, and hence, resource consumption, have increased over time from basic RNN to LSTM.

3. Adding Bidirectionality to Recurrent Neural Networks

Additionally, there was a significant amount of work on the bidirectionality of RNN to provide models with the possibility to capture and use information from both earlier and later in the sequence.

If to express in simple words BRNN (bidirectional RNN) [42, 43] is a modification to RNN, GRU, LSTM consists of two RNNs capturing information simultaneously in opposite directions and only then making predictions. Bidirectional RNN has *forward recurrent layer (component) S* which takes as input current X and feeds the output to help predict current output Y forward in time. On the other hand *backward recurrent layer (component)* S_i' which takes as input current *X* and feeds the output to help predict current output *Y* backward in time as shown in Figure 8.

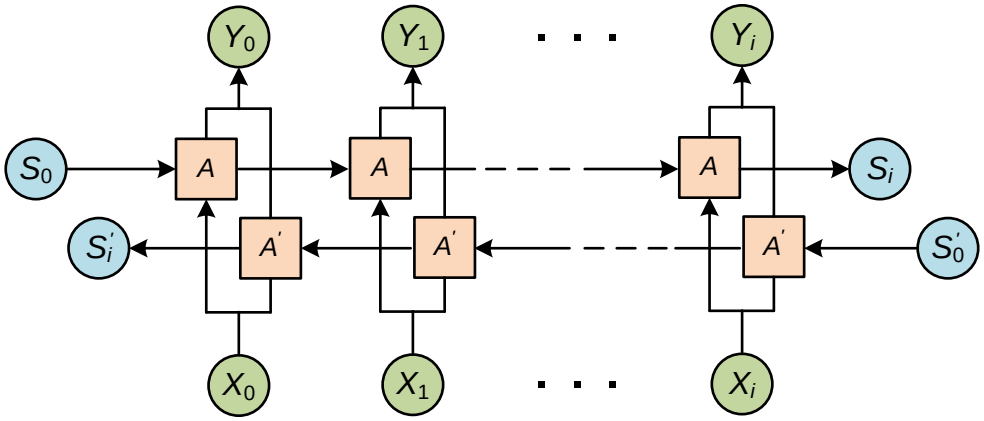


Figure. 8. Bidirectionality in NN scheme

To construct even more powerful models researchers propose to stack units of RNN/LSTM/GRU [44]. This type of architecture is called Deep RNN.

The bottleneck of BRNN is that it needs the entire sequence of data before making any predictions.

Deep RNN is also much more expensive in computation. All of the networks presented had problems in neural machine translation as the input and output sequences regularly were of different lengths because of different language semantics.

4 Solving long-range dependencies of Neural Networks in Natural Language Processing tasks using Attention concept

The main focus of researchers was the problem of long sentences (sentence contained more than 20 words) which can't be stored effectively in one output vector of RNN/GRU/LSTM. In [45] demonstrated significant improvement of BLEU score results using attention mechanism. As a solution to the problem Attention Mechanism was proposed [45, 46, 47].

The main intuition behind Attention is that humans do not read and memorize whole long sentences at once, but part by part. And for a decoder, it would be valuable to know while decoding (for example translation), to which part of the input sequence it should pay more attention. An idea of attention: at each step, the decoder focuses on some particular part of the source. As shown in Figure 9 decoder will focus only on particular words at each step (increased saturation represents more attention), not on the full input sequence.

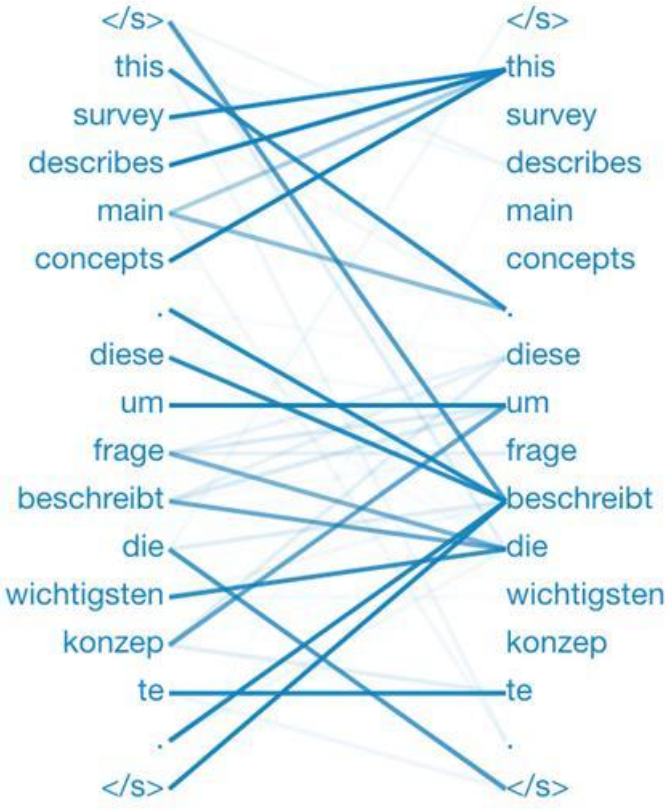


Figure. 9. Visualization of an attention mechanism word dependencies

Attention mechanism uncover such possibility to a decoder by Attention Weights and Context Vector

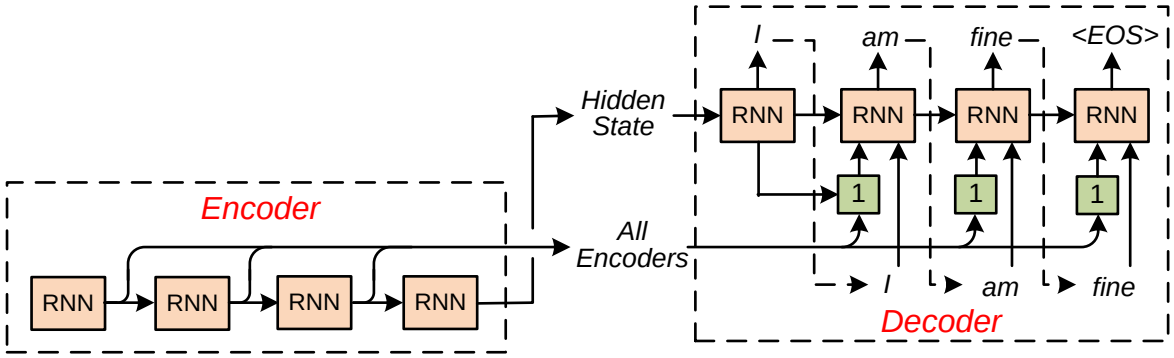


Figure. 10. Simplified Bahdanau Attention [45] overview

In addition to BRNN (which can also be BGRU, BLSTM) *Attention concept* utilizes the idea of *alignment scores* and *attention weights* (the amount of attention Decoder should pay while calculating current time step prediction).

As shown in Figure 10, the all-time step hidden states of Encoder pass with the last layer hidden state of Encoder to Decoder. Central processing occurs in the Decoder. Each time step of Decoder, a set of features (about words and surrounding words) computes and called *Alignment Scores* $e_{ij} = a(s_{i-1}, h_j)$ (differences between encoder and decoder hidden states), which are used to calculate *Attention Weights* (α), $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \alpha_{ij} h_j}$, by softmax function.

Context Vector (*c*), $c_i = \sum_{j=1}^{Tx} \alpha_{ij} h_j$, (shown as 1 in Figure 10) is calculated for each time step of Decoding by combining Attention Weights with the previous Decoder outputs to be passed to Decoder RNN, $\alpha_{t_s} = \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'=1}^{S} \exp(\text{score}(h_t, \bar{h}_{s'}))}$.

Although it is amazing that such a simple and generic architecture as bidirectional LSTM with attention (just a few equations and few tens lines of code) can predict (translate, classify) with such a great result this architecture admits mistakes and has bottlenecks [34, 48]. This type of architecture can't be parallelized – attention mechanism provided for sequential RNNs helped solve long term dependencies issues by using more appropriate context at each step, but the problem of parallelization of computation raised even more.

Additionally if to analyze the NLP area not only through the prism of translation where most time machines just translate sentence by sentence and focus on Natural Language Understanding area RNNs do not show good results in overall

context understanding and modeling, especially during text generation tasks. This is exactly where the architecture of the Transformer is able to do better.

5 Removal of Recurrent Neural Networks and shifting to fully Attention-based architecture using Self-Attention and Transformer

In the paper [48] researchers from Google introduced Transformer, a novel neural network architecture for Language Understanding based on a self-attention mechanism. High level architecture of Encoder and Decoder of Transformer are presented in Figure 11 and described in details below.

The main novelty was that to build a language model there is no need for any recurrent (RNN) or convolutional (CNN) layers at all. Solely self-attention and feed-forward layers are enough.

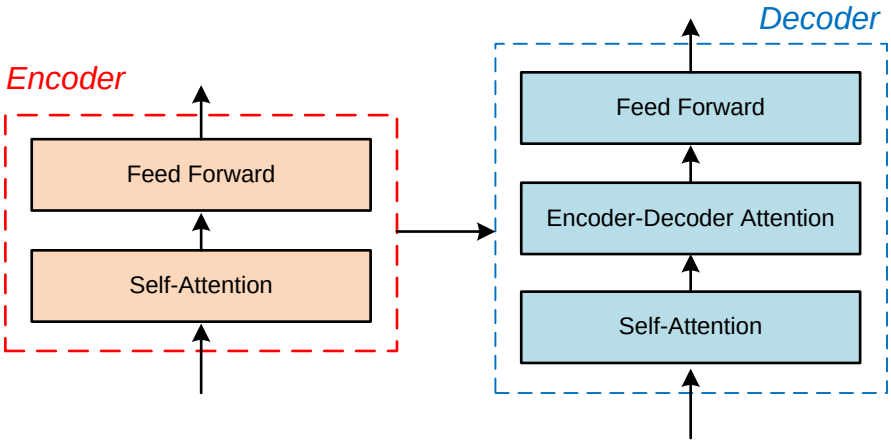


Figure. 11. Encoder-Decoder architecture applied in Transformer

The Transformer includes a lot from what was described before:

– Bidirectionality;

– Encoder-Decoder Architecture to support the different length of input-output;

– Self-Attention. In addition to attention, researchers develop the idea of attention and presented self-attention in the Transformer [48];

– Parallelization. It also uncovers the possibility to parallelize (at least on Feed-Forward step which is most expensive) by completely replacing sequential computation (RNNs or Convolutional based) to Attention-based network.

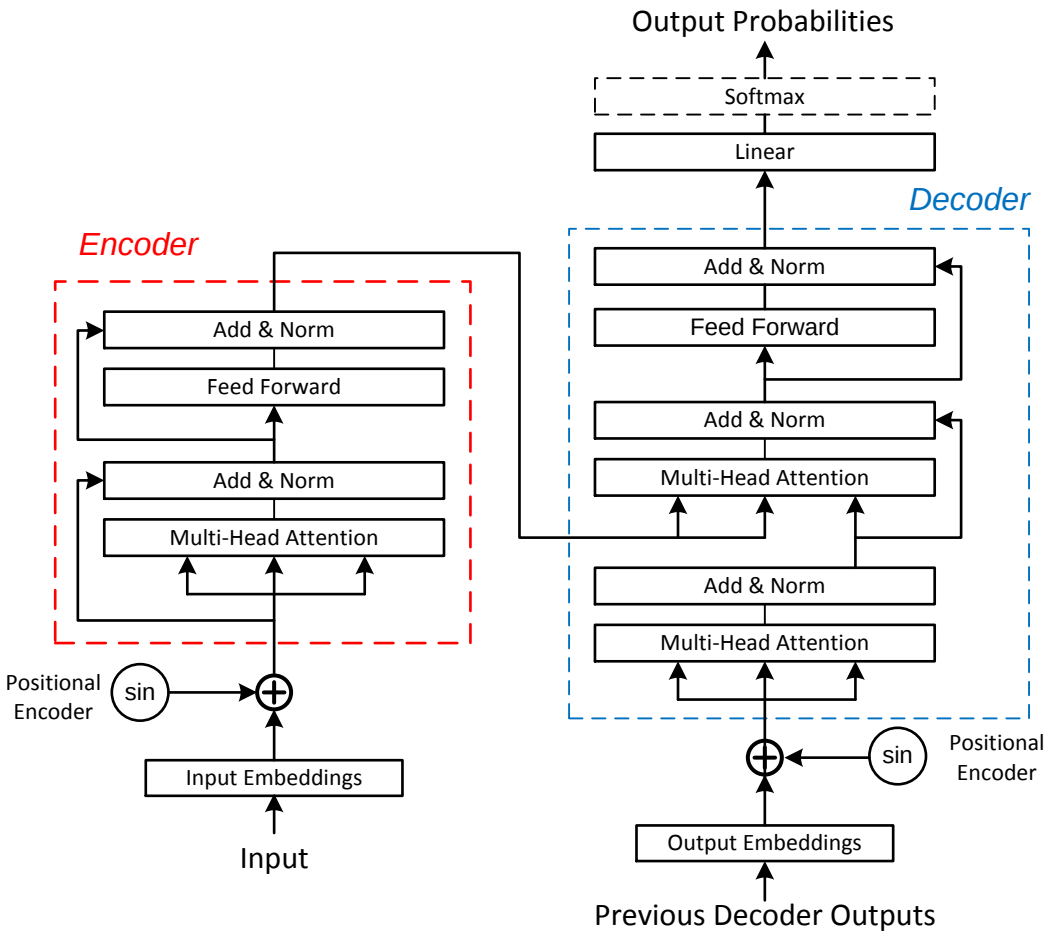


Figure. 12. The Transformer – model architecture

In Figure 12 presented the architecture of the Transformer. Below will be presented and described the main components and concepts of this architecture.

Encoder

The encoder consists of multiple (six in the original paper) stacked Self-Attention and Feed Forward layers (as shown in Figure 13) with Residual Connections, and Positional Encoder (as shown in Figure 12). As usual, the embedding layer is applied in the bottom to convert input sequence to numerical representation. As seen from Figure 13, is that Feed Forward Network does not have dependencies and thus can be parallelized. This is an important concept behind the Transformer possibility to learn on a truly big amount of data that LSTM and GRU cannot afford.

Self-Attention Layers help to understand the model which parts of the input sequence (words) to focus on while Encoding sequence. The most important novelty is using three vectors: Query vector, Key vector, and Value vector to create "query", "key", and "value" projection of each word in the input sentence.

 ISSN 1560-8956

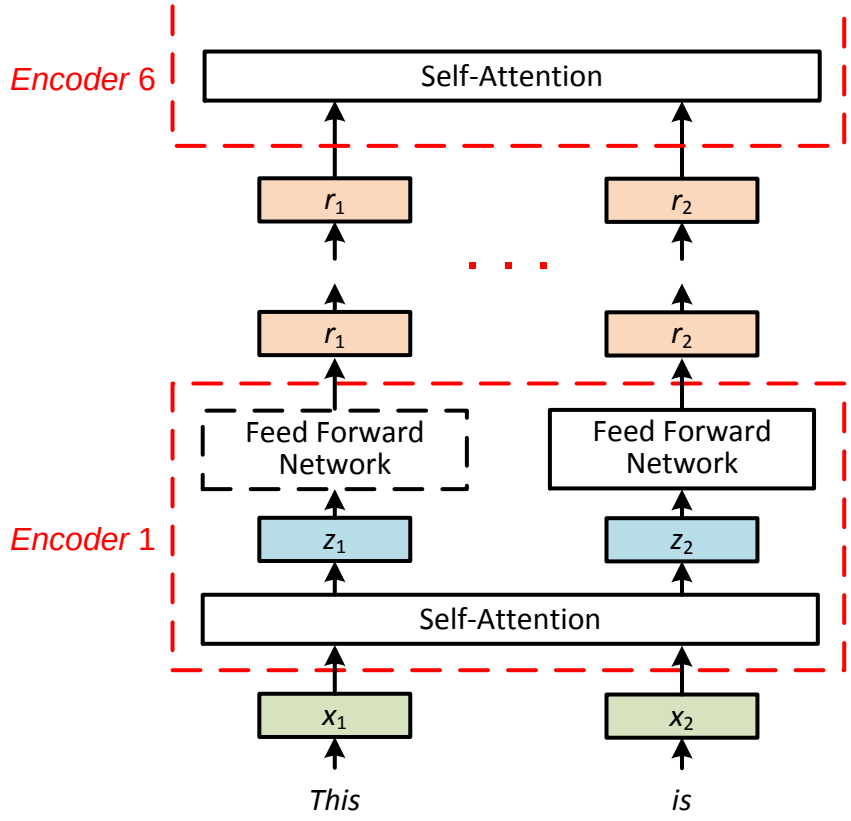


Figure. 13. Transformer Self-Attention parallel processes

Positional Encoding

The model needs to store information about the absolute and relative position of each word, but since there is no recurrence or convolution Positional Encodings were proposed to present the order of input sequence.

Sine and Cosine functions of different frequencies and Learned Positional Embeddings were proposed as Positional Encodings. The authors made a choice to Sine and Cosine as it also helped to solve a bottleneck if the length of a sequence in production would be longer than training sequence as shown in Figure 12.

Decoder

The decoder also consists of multiple (equal to Encoder) stacked Self-Attention, Feed Forward layers with Residual Connections, and additionally Encoder-Decoder Attention layer in the middle as shown in Figure 11 and Figure 12.

In comparison to Encoder, Decoder's Self-Attention layer differs. The main idea here is Masking Future Positions. In Encoder each position can attend to all positions, but in Decoder to prevent leftward information flow to preserve the auto-regressive property, each position can attend only to early positions in the output sequence.

Another important layer of Decoder is Encoder-Decoder Attention layer which gets outputs of the last Attention layer of Encoder as input and uses Key and Value attention vectors to focus on appropriate places in the input sequence.

There is nothing novel and specific in the position-wise fully connected Feed-Forward Network layer of both Encoder and Decoder. Linear transformations are identical all over the positions but what makes a difference is a set of variable parameters in each layer.

As of today, Transformer is the most powerful architecture which can be trained on enormous amounts of training data with hundreds of millions of parameters (550 million parameters for RoBERTa and even more for XLM-R [11–15].

A key advantage of models built using Transformer architecture is that it doesn't need to be pre-trained with labeled data, so it can learn using any plain text. It provides the possibility to work with very big datasets and leads to even better accuracy.

It is clear that it is impractical to train such a big network from scratch every time and for every particular task (even today it will cost hundreds of thousands of USD and enormous computational GPU power), hence such a big model come as pre-trained [13–15] models which can then be finetuned [49] for various scenarios. It can be achieved by an additional layer of neurons on the end that were not trained in the pre-training and train them as a part of the new model for specific tasks.

Conclusion

Significant progress was made in the last decade in the NLP area. Application of deep learning changed rules and uncovered new possibilities with RNNs, BLSTMs, and Attention. Architecture based on Transformers advances even further and shows a state-of-the-art results on most of the NLP tasks.

The introduction of deep pre-trained language models in the last couple of years (ELMO, BERT, ULMFIT, Open-GPT, etc.) signals the same very important shift to transfer learning in NLP that computer vision got earlier.

From a technical and methodological perspective, it seems there is still a room for the application of simple architectures based on RNNs and LSTMs, whilst the BLEU computational metrics push us towards more advanced designs, like Transformer, it is still very expensive computationally to use the Transfomer Architecture for every task, hence LSTM and GRU will still be used frequently for many practical implementations.

In the most of NLP tasks LSTMs and Attention combined in one design will deliver required results. Once there are enough computational resources it is reasonable to use Transformer to achieve the best result possible.

The prerequisites of a specific task to be solved and computational power to be applied should be considered when comparing designs and techniques. Many of the latest approaches are too demanding in computational resources that is a direction of further improvement.

REFERENCES

1. Britz, D., Goldie, A., Luong, M.-T., & Le, Q. Massive exploration of neural machine translation architectures. Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017, pp. 1442–1451.

2. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. Learning phrase representations using rnn encoder–decoder for statistical machine translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1724–1734.

3. Sutskever, I., Vinyals, O., & Le, Q. Sequence to Sequence Learning with Neural Networks. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, 2014, pp. 3104–3112.

4. Krizhevsky, A., Sutskever, I., & Hinton, G. E. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 2017, pp. 84–90.

5. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. Language models are unsupervised multitask learners. OpenAI Blog 1(8):9, 2019.

6. Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., & Stoyanov, V. Unsupervised cross-lingual representation learning at scale. ArXiv:1911.02116, 2020.

7. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. ArXiv:1810.04805, 2019

8. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. ArXiv:1906.08237, 2020.

9. Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. On the properties of neural machine translation: Encoder–decoder approaches. Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, 2014, pp. 103–111.

10. Nayak, T., & Ng, H. T. Effective modeling of encoder-decoder architecture for joint entity and relation extraction. ArXiv:1911.09886, 2019.

11. Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., & Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 2978–2988.

12. Gordon, M. A., Duh, K., & Andrews, N. (2020). Compressing bert: Studying the effects of weight pruning on transfer learning. ArXiv:2002.08307.

13. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. ArXiv:1906.08237, 2020.

14. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. ArXiv:1810.04805, 2019.

15. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. ArXiv:1907.11692, 2019.

16. Liu, Q., Kusner, M. J., & Blunsom, P. A survey on contextual embeddings. ArXiv:2003.07278, 2020.

17. Pennington, J., Socher, R., & Manning, C. Glove: Global vectors for word representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543.

18. Mikolov, T., Chen, K., Corrado, G., & Dean, J. Efficient Estimation of Word Representations in Vector Space. In Proceedings of the First International Conference on Learning Representations, 2013, pp. 1–13.

19. Lebret, R., & Collobert, R. Word embeddings through hellinger pca. Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, 2014, pp. 482–490.

20. Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. Deep contextualized word representations. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2018, pp. 2227–2237.

21. Kristoffersen, M. S., Wieland, J. L., Shepstone, S. E., Tan, Z.-H., & Vinayagamoorthy, V. Deep joint embeddings of context and content for recommendation. ArXiv:1909.06076, 2019.

22. Zhang, Y., & Ma, Q. Citation recommendations considering content and structural context embedding. ArXiv:2001.02344, 2020.

23. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. Distributed Representations of Words and Phrases and their Compositionality. In Advances in Neural Information Processing Systems 26, 2013, pp. 3111–3119.

24. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. Learning phrase representations using rnn encoder–decoder for statistical machine translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1724–1734.

25. Arjovsky, M., Shah, A., & Bengio, Y. Unitary Evolution Recurrent Neural Networks. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, 2016, pp. 1120–1128.

26. Zilly, J., Srivastava, R., Koutník, J., & Schmidhuber, J. Recurrent Highway Networks. Proceedings of the 34th International Conference on Machine Learning, 2017, pp. 4189–4198.

27. Rumelhart, D., Hinton, G., & Williams, R. Learning Representations by Back-propagating Errors. Nature, 323(6088), 1986, pp. 533–536.

28. You, Y., & Nikolaou, M. (1993). Dynamic process modeling with recurrent neural networks. AIChE Journal, 39(10), pp. 1654–1667.

29. Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. How to construct deep recurrent neural networks. ArXiv:1312.6026, 2014.

30. Chung, J., Ahn, S., & Bengio, Y. Hierarchical Multiscale Recurrent Neural Networks. International Conference on Learning Representations (ICLR), 2017.

31. Goodfellow, I., Bengio, Y., & Courville, A. Deep Learning. Sequence Modeling: Recurrent and Recursive Nets, 2016, pp. 367-415.

32. Semeniuta, S., Severyn, A., & Barth, E. Recurrent Dropout without Memory Loss. In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, 2016, pp. 1757–1766.

33. Zhang, S., Wu, Y., Che, T., Lin, Z., Memisevic, R., Salakhutdinov, R., & Bengio, Y. Architectural Complexity Measures of Recurrent Neural Networks. Neural Information Processing Systems, 2016, pp. 1822-1830.

34. Bengio, Y., Simard, P., & Frasconi, P. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 1994, pp. 157–166.

35. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. NIPS 2014 Workshop on Deep Learning and Representation Learning, 2014.

36. Dey, R., & Salem, F. M. Gate-variants of Gated Recurrent Unit (Gru) neural networks. 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017, pp. 1597–1600.

37. Sak, H., Senior, A., & Beaufays, F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. INTERSPEECH, 2014, pp. 338-342.

38. Li, X., & Wu, X. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 4520–4524.

39. Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., & Schmidhuber, J. Lstm: A search space odyssey. IEEE Transactions on Neural Networks and Learning Systems, 28(10), 2017, pp. 2222–2232.

40. Hochreiter, S., & Schmidhuber, J. Long short-term memoryNeural computation, 9(8), 1997, pp. 1735–1780.

41. Weiss, G., Goldberg, Y., & Yahav, E. On the practical computational power of finite precision rnns for language recognition. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), 2018, pp. 740–745.

42. Schuster, M., & Paliwal, K. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11), 1997, pp. 2673–2681.

43. Berglund, M., Raiko, T., Honkala, M., Kärkkäinen, L., Vetek, A., & Karhunen, J. Bidirectional Recurrent Neural Networks as Generative Models. Neural Information Processing Systems, 2015, pp. 856-864.

44. Mousa, A., & Schuller, B. Contextual bidirectional long short-term memory recurrent neural network language models: A generative approach to sentiment analysis. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, 2017, pp. 1023–1032.

45. Bahdanau, D., Cho, K., & Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. In International Conference on Learning Representations (ICLR), 2015.

46. Luong, T., Pham, H., & Manning, C. D. Effective approaches to attention-based neural machine translation. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015, pp. 1412–1421.

47. Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001.

48. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., & Polosukhin, I. Attention is All you Need. In Advances in Neural Information Processing Systems 30, 2017, pp. 5998–6008.

49. Howard, J., & Ruder, S. Universal language model fine-tuning for text classification. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2018, pp. 328–339.