# Techniques Comparison for Natural Language Processing

Olena Iosifova[1][0000-0001-6507-0761], Ievgen Iosifov[1][0000-0001-6203-9945], Oleksandr Rolik[2][0000-0001-8829-4645], and Volodymyr Sokolov[3][0000-0002-9349-7946]

[1] Ender Turing OU, Tallinn, Estonia
{oi,ei}@enderturing.com
[2] National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute," Kyiv, Ukraine
o.rolick@kpi.ua
[3] Borys Grinchenko Kyiv University, Kyiv, Ukraine
v.sokolov@kubg.edu.ua

**Abstract.** These improvements open many possibilities in solving Natural Language Processing 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 the 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. As a result, 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 embedding to build deep context understanding. The latest offered architectures based on these approaches require a lot of computational power for training language models, and it is a direction of further improvement. Algorithm for choosing right model for relevant business task provided considering current challenges and available architectures.

**Keywords:** Natural Language Processing, NLP, Language Model, Embedding, Recurrent Neural Network, RNN, Gated Recurrent Unit, GRU, Long Short-Term Memory, LSTM, Encoder, Decoder, Attention, Transformer, Transfer Learning, Deep Learning, Neural Network.



## 1 Introduction

Natural Language Processing (NLP) is computer comprehension, analysis, manipulation, and generation of natural language. NLP covers many 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 NLP 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 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, etc.) leverage the following approaches: contextual embedding, bi-directionality, encoder-decoder architecture, attention mechanism and transformer, pre-training modeling and fine-tuning.

The structure of this paper includes language models architecture observation in Sect. 2. Word and contextual embedding techniques are described in Sect. 3 and encoder-decoder—in Sect. 4. Neural Networks applied as a part of encoder and decoder are observed in Sect. 5. Model choosing algorithm presented in Sect.6. Conclusion in Sect. 7 provides further promising areas of research in NLP.

## 2 Common Approach of Natural Language Processing Downstream Tasks

NLP downstream tasks (machine translation, sentiment analysis, question answering, part of speech tagging, and many more) usually are solved with some different approaches chosen for a specific task. Generally, it comes to supervised learning on task-specific datasets, which is quite consuming in terms of research hours and computational resources. In addition to these inconveniences, systems that are built with this approach are very sensitive to task specifications and changes in the data distribution. Current trends move towards unsupervised universal models and transfer learning as pre-trained models with further fine-tuning. The techniques and components that are offered for observation in this article correspond to current trends and groundbreaking achievements in NLP [1]. It outlines the architectures of the state of the art pre-training models [2]. Everything starts with ready for training or test data. Input data runs through some techniques to result in word embedding [3, 4] or contextual embedding that could be described as multidimensional word knowledge embedding. Despite the use of the term *word*, readers should not be confused. Word embedding is 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 embedding [5] is used as input for an *encoder* which forms a *context vector* and forwards it to a *decoder*. A decoder in its turn forms a set of probabilities necessary to figure out an output.

## 3 Word and Contextual Embedding Approaches

For a neural network to be able to complete its task there is a necessity to provide a numerical token representation of input sequence. Word embedding techniques create vectors out of tokens. Vectors comparison results in tokens semantic similarity. Embedding techniques such as GloVe and Word to vector explain the concept of modeling input sequence through representation [4].

The main idea and task are 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. Hence different words with similar meanings 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 [2]. Currently, the area of representation of input sequence advances far ahead of initial papers and new approaches appear. Contextual embedding [5] creates 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 some other languages.

## 4 High-Level Encoder-Decoder Architecture

The neural network encoder-decoder model significantly improved the performance of language models. Quite simple Recurrent Neural Network (RNN) architecture to process input and output sequences of variable lengths was offered. The 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 embedding sequentially (from left to right) passing to the next timestamp RNN hidden state calculated in the current timestamp RNN. After all inputs proceed to the final time stamp, the final timestamp 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 the decoder which task is to decode and generate predictions based on input hidden state. Decoder process sequentially taking as input to each current timestamp output activation of previous timestamp RNN and output prediction of previous timestamp RNN. For the first timestamp, it takes a *beginning of sentence token* (BOS) as a prediction of the previous layer. The decoder generates predictions until it generates *end of sentence token* (EOS, 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 lengths. Therefore, that it resolves the problem of different lengths of an input and an output sequence in Neural Machine Translation.

The encoder-decoder architecture consists of two RNNs 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.

## 5 Neural Networks for Natural Language Processing Tasks

Progress in the application of Neural Networks to NLP tasks brings huge improvements in both science and business areas.

### 5.1 Recurrent Neural Networks

RNNs [6] is the main starting point in the deep learning NLP area. Deep neural networks uncover a second life for RNNs. A 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 [7] is very simple, the network takes an input vector $X$ and produces an output vector $Y$. 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) $h_t$:

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

If output predictions needed by task then activation function $g$ (or softmax function) with learned weights $W_y$ and bias by might be applied to current output $h_t$ to make output prediction $y_t$:

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

Especially important for NLP areas is that the 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. RNN cells take the output of the previous cell as an input to the current cell, and the previous cell contains information of its previous cell and so on. This type of connection is called a recurrent connection.

Despite the wide adoption, RNN possesses significant drawbacks [7]. Unidirectional learning leads to the problem that the model cannot 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. This was a triggering point to search for a solution that resulted in further useful findings like GRU and LSTM.

### 5.2 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) [1, 8] is to capture long-term dependencies by adding Memory Cell ($C_t$) which in GRU is equal to hidden state (activation) $h_t = (1-z) * h_{t-1} + z_t * \tilde{h}_t$. And each time stamp cell considers rewriting this cell with *Candidate Value* $\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$, using two Gates described by equations (*Update Gate* $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$, and *Reset Gate* $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$) as shown in Fig. 1. 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 timestamp 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 timestamp should be saved for the future. Reset gate, on the other hand, decides how much information from the previous timestamp should be removed.

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

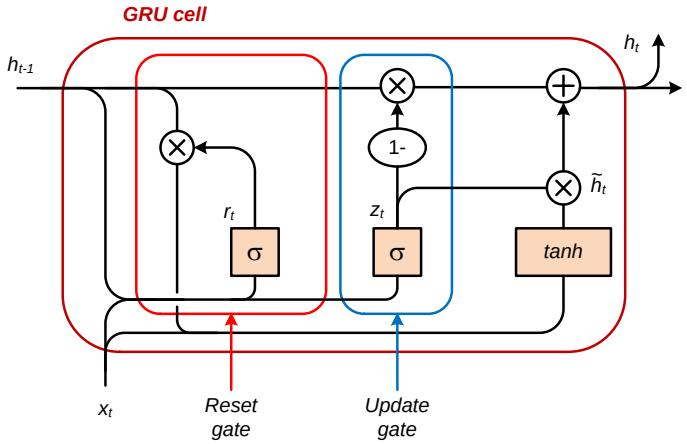


**Fig. 1.** A scheme of GRU cell.

Another type of architecture that can capture mid-term dependencies, even more powerfully than GRU, is Long Short-Term Memory (LSTM) [9]. In a difference to GRU that has two gates, LSTM possesses three gates. An important concept in LSTM is that Memory Cell ($C_t$) is not anymore equal to output hidden state (activation) $h_t$. Output hidden state of the current timestamp in LSTM carry on to the next cell not alone but with updated Memory Cell value.

LSTM, also, uses two separate gates (Update Gate and a Forget Gate) to update Memory Cell value, 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 as shown in Fig. 2.

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

Usage of separate Update Gate and Forget Gate to calculate Memory Cell value $C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$ gives Memory Cell the possibility not only to store new information in the current time step Memory Cell 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 $o_t = \sigma(x_t U^0 + h_{t-1} W^0)$, 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 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.

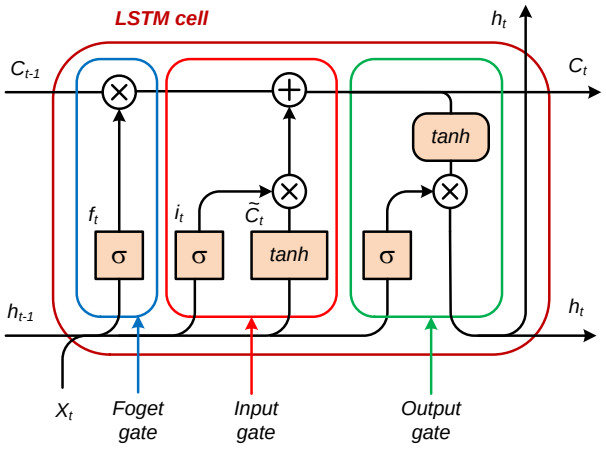


**Fig. 2.** A scheme of LSTM cell.

These networks are quite consuming for computational resources. Moreover, this type of architecture cannot 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. Retrospectively application of RNN, GRU, and LSTM was the major stage in modern NLP that significantly affected future development of the area.

### 5.3 Bidirectionality in 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 Bidirectional RNN (BRNN) [9] is a modification to RNN, GRU, LSTM consists of two RNNs capturing information simultaneously in opposite directions and only then making predictions. BRNN has forward recurrent layer (component) *S* that 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.

To construct even more powerful models researchers propose to stack units of RNN/LSTM/GRU. 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.

### 5.4 Attention Concept

The focus of researchers was the problem of long sentences (sentence contained more than 20 words) which cannot be stored effectively in one output vector of RNN/GRU/LSTM. In [11] demonstrated significant improvement of BLEU score results using attention mechanisms. As a solution to the problem, Attention Mechanism was proposed.

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. Decoder focuses only on particular words at each step (increased saturation represents more attention), not on the full input sequence.

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

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).

The all-time step hidden states of encoder pass with the last layer hidden state of encoder to the 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 $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \alpha_{ij} h_j}$, by softmax function.

Context Vector $c_i = \sum_{j=1}^{Tx} \alpha_{ij} h_j$, 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 amazingly, 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 [12]. This type of architecture cannot 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* can do better.

### 5.5 Self-Attention and Transformer

In the paper [12] 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 the transformer are presented in Fig. 3 and described in detail 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.

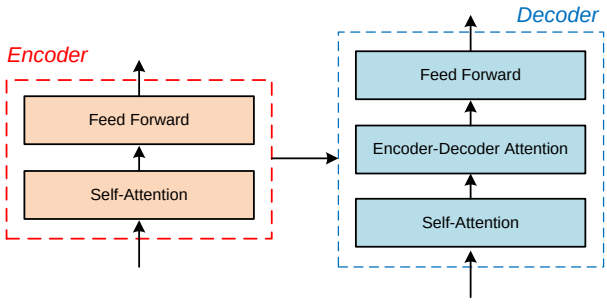


**Fig. 3.** Encoder-decoder architecture applied in the transformer.

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; parallelization (at least on Feed-Forward step which is most expensive) by completely replacing sequential computation (RNNs or Convolutional based) to Attention-based network.

The main components and concepts of this architecture will be presented and described below.

**Encoder.** The encoder consists of multiple stacked Self-Attention and Feed Forward layers with Residual Connections and Positional Encoder. As usual, the embedding layer is applied in the bottom to convert input sequence to numerical representation. 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.

**Decoder.** The decoder also consists of multiple (equal to the encoder) stacked Self-Attention, Feed Forward layers with Residual Connections, and additionally encoder-decoder attention layer in the middle. In comparison to the encoder, Decoder's Self-Attention layer differs. The main idea here is Masking Future Positions. In the encoder, each position can attend to all positions, but in the 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 the decoder is Encoder-Decoder Attention layer which gets outputs of the last Attention layer of the encoder as an input and uses Key and Value attention vectors to focus on appropriate places in the input sequence.

## 6 Comparison Concepts and Architectures Usage

**Embedding.** Almost everyone nowadays uses some kind of embedding technique to tokenize input sequences. If your task is not domain-specific, you will probably end up using one of the pre-trained embeddings with dimensionality (300–512). And if you use domain-specific tasks, the choice would be simply based on available computational resources. Starting from simple Word2Vec and Glove and moving to advanced Contextual embedding techniques and increased dimensionality can solve your tasks with high accuracy.

**Architecture.** As of today, the transformer is the most powerful architecture, which can be trained on enormous amounts of training data with billions of parameters. 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 comes as pre-trained models, which can then be fine-tuned for various scenarios and tasks. It can be achieved by

an additional layer of neurons on the end that was not trained in the pre-training and train them as a part of the new model for specific tasks.

A key advantage of models built using the transformer architecture is that it does not need to be trained with labeled data, so it can learn using any cleaned raw text. This provides the possibility to work with very big datasets and leads to even better accuracy.

The current leaderboard in different NLP competition more and more narrowing to big tech corporations and not universities. That is because of the availability of computational power. On the other hand, leaderboard results and practical implementation is different. Even today, many production-based services use RNN (LSTM and GRU modifications) with attention e.g., for intent classification in widespread chatbot frameworks.

In Fig 4. proposed algorithm for choosing modern approach to solve business NLP tasks considering domain specification of task and availability of resources. Also, it is a good idea to compare BLSTM with Attention and Transformer based architectures results in terms of accuracy, consumption of resources, time to train (hence to improve), interpretability.

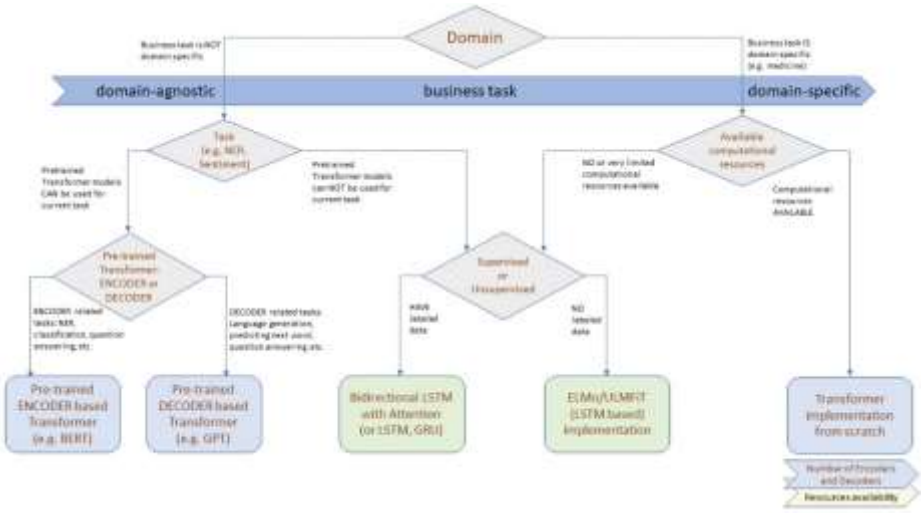


**Fig. 4.** Algorithm for choosing a suitable NLP architecture for a business task.

## 7 Conclusions and Future Work

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. Architectures based on Transformers advances even further and show state-of-the-art results on most of the NLP tasks.

The introduction of deep pre-trained language models in the last couple of year's significant shift to transfer learning in NLP. At the same time, many of the latest approaches are too demanding in computational resources and algorithms for choosing the right model for the business task presented in current work.

The authors see demand in the sentence boundary detection task. They will research the current area in nearest future using the algorithm provided in the current work to choose models and compare obtained results.

## References


1. Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder–decoder approaches. In SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation: 103–111 (2014). https://doi.org/10.3115/v1/w14-4012
2. Firth, J. R.: A Synopsis of Linguistic Theory, 1930–1955 (1957).
3. Pennington, J., Socher, R., Manning, C.: Glove: global vectors for word representation. In 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP): 1532–1543 (2014). https://doi.org/1010.3115/v1/d14-1162
4. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In First International Conference on Learning Representations: 1–13 (2013). http://arxiv.org/abs/1301.3781
5. Peters, M., et al.: Deep contextualized word representations. In 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies **1**: 2227–2237 (2018). https://doi.org/10.18653/v1/n18-1202
6. Rumelhart, D., Hinton, G., Williams, R.: Learning representations by back-propagating errors. Nat. **323**(6088): 533–536 (1986). https://doi.org/10.1038/323533a0
7. Goodfellow, I., Bengio, Y., Courville, A.: Sequence modeling: recurrent and recursive nets, Deep Learning: 367–415 (2016).
8. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. In NIPS 2014 Workshop on Deep Learning and Representation Learning: 1–9 (2014). http://arxiv.org/abs/1412.3555
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8): 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
10. Schuster, M., Paliwal, K.: Bidirectional recurrent neural networks. In IEEE Transactions on Signal Processing **45**(11): 2673–2681 (1997). https://doi.org/10.1109/78.650093
11. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In International Conference on Learning Representations (ICLR): 1–15 (2015). http://arxiv.org/abs/1409.0473
12. Vaswani, A., et al.: Attention is all you need. In Advances in Neural Information Processing Systems **30**: 5998–6008 (2017). http://arxiv.org/abs/1706.03762