Applying Machine Learning to the Task of Generating Search Queries

Alexander Gusenkov [1][0000-0003-4019-7322] and Alina Sittikova [1][0000-0002-9539-764X]

[1] Kazan Federal University, Kazan, Russia
gusenkov.a.m@gmail.com, sitti.alina@mail.ru

Abstract. In this paper we research two modifications of recurrent neural networks – Long Short-Term Memory networks and networks with Gated Recurrent Unit with the addition of an attention mechanism to both networks, as well as the Transformer model in the task of generating queries to search engines. GPT-2 by OpenAI was used as the Transformer, which was trained on user queries. Latent-semantic analysis was carried out to identify semantic similarities between the corpus of user queries and queries generated by neural networks. The corpus was converted into a bag of words format, the TFIDF model was applied to it, and a singular value decomposition was performed. Semantic similarity was calculated based on the cosine measure. Also, for a more complete evaluation of the applicability of the models to the task, an expert analysis was carried out to assess the coherence of words in artificially created queries.

Keywords: natural language processing, natural language generation, machine learning, neural networks.

1 Introduction

Natural language generation is a process of creating meaningful phrases and sentences in the form of natural language. Two main approaches can be distinguished among the algorithms for creating texts: methods based on rules and methods based on machine learning. The first approach allows to achieve high quality texts, but requires knowledge of the rules of the language and is time consuming to develop [1], while the second approach depends only on training data, but often makes grammatical and semantic errors in the created texts [2].

Currently, generation of texts using neural networks is being actively researched; one of the most popular algorithms is recurrent neural networks [3]. The second leading architecture is the Transformer model [3]. These architectures were considered in the solution of the generating search queries task.

The purpose of this article is to study the above-mentioned architectures, analyze their quality and applicability to this task. The use of automatically generated queries to search engines is relevant, since most companies do not issue their search queries for free, and a search engine must be tested while being developed. Also, the received queries can be used to improve the efficiency and optimize the search engine.

Search queries from users of AOL (America Online), which were anonymously posted on the Internet in 2006, were used in this paper. Although the company did not identify its users, personal information was present in many queries [4], which companies are now trying to avoid. Algorithms have been proposed to help preserve user anonymity, but the question is whether data that can be safely published is of practical use. To solve this problem, it is proposed to use automatically generated queries.

1.1 Subject area overview

Natural language text generation algorithms are actively studied and used in many software systems, so at the moment there is a large amount of research in this area.

One of the first approaches is the fill-in-the-gap template system. It is used in texts that have a predefined structure and, if it is necessary to fill in a small amount of data, this approach can automatically fill in the blanks with data obtained from spreadsheets, databases, etc. An example of this approach is Microsoft Word mailmerge [5].

The second step was to add general-purpose programming languages to the first approach that support complex conditionals, loops, etc. This approach is more powerful and useful, but the lack of language capabilities makes it difficult to create systems that can generate quality texts.

The next step in the development of template-based systems is the addition of word-level grammatical functions that deal with morphology and spelling. Such functions greatly simplify the creation of grammatically correct texts. Next, systems dynamically create sentences from representations of the values they need to convey. This means that systems can handle unusual cases without the need to explicitly write code for each case, and are significantly better at generating high-quality "micro-level" writing. Finally, in the next stage of development, systems can generate well-structured documents that are relevant to users. For example, a text that needs to be persuasive can be based on models of argumentation and behavior change [5].

After moving from templates to dynamic text generation, it took a long time to achieve satisfactory results. If we consider the generation of texts in natural language a subsection of natural language processing, then there is a number of the most developed algorithms – Markov chains [6], recurrent neural networks, long short-term memory netwroks and the Transformer model. There are text generation tools based on these methods, for example commercial Arria NLG PLC, AX Semantics, Yseop and others, as well as open source programs Simplenlg, GPT, GPT-2, BERT, XLNet.

Also, the use of generative adversarial networks for text generation is currently being researched, since they show excellent results in the task of generating images [7].

2 Data collection

User queries in English from the 2006 AOL search engine were selected as data for training neural networks. Researchers try to avoid using this data in their work, as it

can be considered revealing, but this paper uses only the texts of the requests themselves, without the IDs of users and websites, that is, without using personal information. The initial data are presented in the form shown in Fig. 1.

```
Query   QueryTime       ItemRank        ClickURL
carbol tunnel   2006-03-01 01:01:21
how to install a glue down floor        2006-03-01 07:13:45     2       http://doityourself.com
how to install a glue down floor        2006-03-01 07:13:45     8       http://www.homerenovationguide.com
how to install a glue down floor        2006-03-01 07:13:45     9       http://www.hardwoodinstaller.com
how to install a glue down floor        2006-03-01 07:35:01     20      http://www.ehow.com
how to install a glue down floor        2006-03-01 07:43:50     26      http://www.hoskinghardwood.com
mapquest        2006-03-01 19:40:11     1       http://www.mapquest.com
indian projectile points        2006-03-02 21:12:10
indian projectile points        2006-03-02 21:13:02
indian projectile points        2006-03-02 21:13:03     1       http://www.utexas.edu
indian projectile points        2006-03-02 21:13:03     6       http://www.iath.virginia.edu
indian projectile points        2006-03-02 21:22:40
indian projectile points        2006-03-02 21:22:42
indian projectile points        2006-03-02 21:22:46     16      http://www.mnsu.edu
indian projectile points        2006-03-02 21:22:46     18      http://www.madison.k12.wi.us
```

Fig. 1. Initial training data.

Queries longer than 32 words and erroneous requests containing no information were removed from the corpus. Duplicate queries and queries containing website names have also been removed, as they are not natural language examples. In total, 100 thousand queries were randomly selected for training. Fig. 2 shows examples of data after preprocessing.

```
i can love you like that
breyer traditional horse models
waffle irons
home made structures
dominion a prequel to the exorcist
what is a liability
capstone turbine
danville ill
statetax
old club men ties
piciculture
pagan jewelry
unicoi county memorial hospital
```

Fig. 2. Data after preprocessing.

The queries were separated into character-tokens, each character was assigned with a natural number, the entire corpus was encoded using this dictionary.

3 Recurrent networks

Recurrent neural networks (RNN) are a family of neural networks where the connections between the elements form a directed sequence [8]. They can use their internal memory to process sequences of arbitrary length, and they are also good at identifying the dependencies between tokens.

However, recurrent networks learn slowly, and their ability to memorize long dependencies is limited due to the vanishing gradient problem [9].

Two types of recurrent networks were implemented; they are most often used in the task of generating texts in natural language – Long Short-Term Memory Network [10] and Gated Recurrent Unit [11]. Studies have shown that these types of networks have comparable accuracy, and, depending on the task, one network can be more accurate than the other.

3.1 Long short-term memory networks

Long short-term memory network (LSTM) is a deep learning system that avoids the vanishing and exploding gradient problems [10]. LSTM networks can memorize significantly longer sequences of characters. They use gates, which are internal mechanisms that can control information flow. Fig. 3 shows the standard form of the LSTM cell.

Each cell has 3 gates: input, output and forget gates. The forget gate vector is calculated as:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f),$$

where x_t is the input vector, h_{t-1} is the output vector of the previous cell, σ is sigmoid function, W_f, U_f, b_f are weight matrices and bias vector.

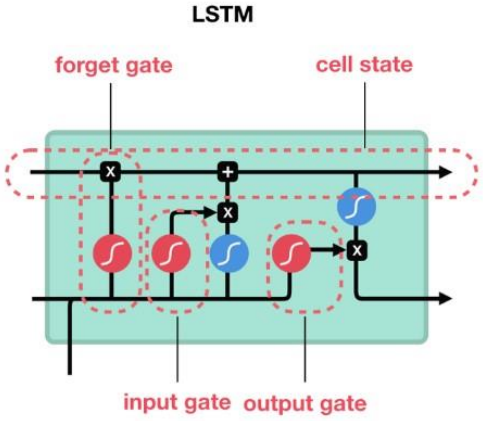

Fig. 3. LSTM cell.

Next, the input gate updates the state of the cell:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i),$$

$$\hat{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c).$$

Then the new value of cell state is calculated:

$$c_t = f_t \circ c_{t-1} + i_t \circ \hat{c}_t,$$

where c_{t-1} is the state of the previous cell. Finally, an output vector decides what the next hidden state should be

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o),$$

$$h_t = o_t \circ \tanh(c_t).$$

The results are passed to the next cell.

3.2 Gated Recurrent Unit

The second implemented model is a network with Gated Recurrent Unit (GRU), which is a new generation of recurrent neural networks, similar to a long short-term memory network [11]. However, compared to LSTM, this type of networks has fewer parameters, and therefore these models are trained faster. The GRU has only 2 gates: update and reset gates. Fig. 4 shows a standard GRU cell.

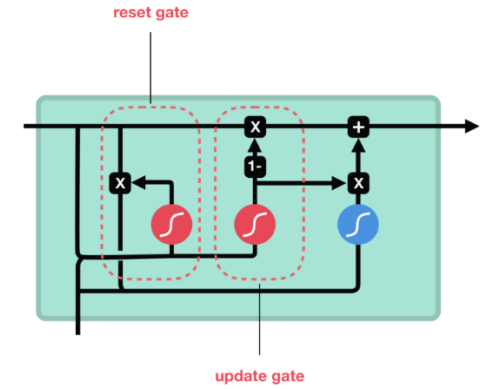

Fig. 4. GRU cell.

Update gate acts as input and forget gates in LSTM and is calculated as:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z).$$

Reset gate is calculated as

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r).$$

Output vector of GRU cell is calculated as

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h).$$

3.3 Attention mechanism in recurrent neural networks

Attention mechanism is a technique used in neural networks to identify dependencies between parts of input and output data [12].

Attention mechanism allows a model to determine the importance of each word for the prediction task by weighing them when creating the text representation. The approach with a single parameter per input channel was used [13]:

$$e_t = h_t w_a,$$

$$a_t = \frac{\exp(e_t)}{\sum_{i=1}^{T} \exp(e_i)},$$

$$v = \sum_{i=1}^{T} a_i h_i.$$

Here h_t is the representation of the word at time step t, w_a is the weight matrix for the attention layer, a_t are the attention importance scores for each time step, v is the representation vector for the text.

4 Implementation of recurrent networks

All neural networks were implemented in Python 3.7 in Google Collab [14], since it has the ability to use GPUs, which significantly decrease the training time of the models. For the implementation of neural networks, we chose the Keras library [15], which is a high-level add-on over TensorFlow. This library greatly simplifies the development of neural networks, since it already has ready-made implementations of the main layers, activation and loss functions. The Adam optimizer (Adaptive Moment Estimation [16]) is used. It is an algorithm in which the learning rate is adjusted for each parameter. Also, the Learning Rate Scheduler function [17] is used as a callback, which allows calculating the learning rate coefficient using a specific function.

The general architecture of the model is shown in Fig. 5.

The preprocessed data was divided into training and validation sets, which constituted 80% and 20% of the corpus, respectively. The training data is fed into the Embedding layer, which converts numbers into vectors that reflect the correspondence between the character sequences and the projections of those sequences. The resulting representations are input to the first LSTM layer (GRU), its output is passed to the second LSTM layer (GRU), and the third in the same way. Next, the output data from the Embedding layer and these three layers are combined and fed to the Attention-WeightedAverage layer. The representation vector obtained from the attention layer is a high-level encoding of the entire text, which is used as input to the final fully connected layer with Softmax activation for classification [13].

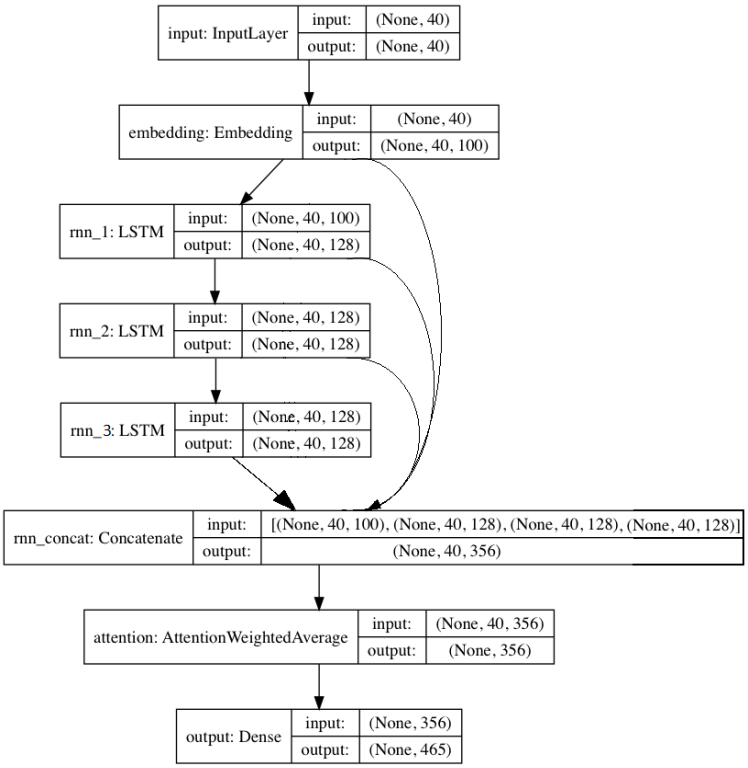

Fig. 5. Architecture of the model.

To check how well the model has trained for a particular epoch, the Categorical Cross-Entropy loss function is calculated as

$$L(y, \hat{y}) = -\sum_{j=0}^{M} \sum_{i=0}^{N} \left(y_{ij} \log\left(\hat{y}_{ij} \right) \right),$$

where $\hat{y}$ are the predicted values.

We conducted experiments with changing the number of LSTM layers (GRU) in the model (2 and 3 layers), as well as adding a Dropout layer after the Embedding layer, which randomly excludes a given number of neurons to prevent network from overfitting and to generalize the model better. Networks with 3 recurrent layers and Dropout performed better.

Bidirectional models of these networks were also trained. Bidirectional recurrent neural network is a model proposed in 1997 by Mike Schuster and Kuldip Paliwal [18], which allows to consider the context of a word not only to the left of it but also to the right of the sequence. A general view of bidirectional neural networks is shown in Fig. 6.

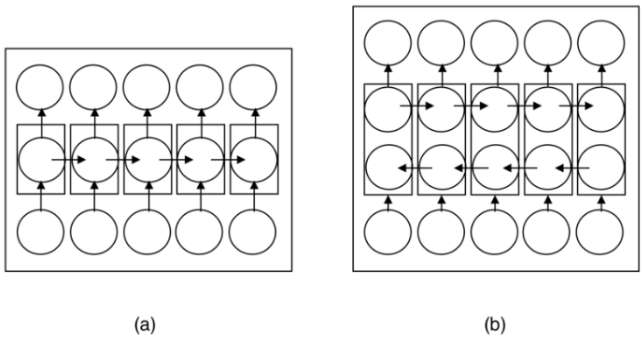

Fig. 6. a) Unidirectional neural network; b) Bidirectional neural network.

In the case of the generating search queries task, the bidirectional model has shown itself to be better than the unidirectional one; the obtained values of the loss function after training the models for 30 epochs are shown in Table 1.

Table 1. Loss function values.

	LSTM	GRU	Bi-LSTM	Bi-GRU
Loss	1,48	1,58	1,30	1,37
Validation Loss	1,6	1,62	1,56	1,57

The value of the loss function decreased on the training data, however, on the validation data, the improvement was less significant, which suggests that the bidirectional model in this task does not learn so well but rather “remembers” the sequences of symbols.

With the help of the implemented model, queries with different "temperatures" were generated. This is a parameter that affects the chance of choosing an unlikely character.

5 Transformer

Transformer is a deep learning model, which was introduced in 2017 [19]. A general view of its architecture is shown in Fig. 7.

Transformers consist of stacks of equal numbers of encoders and decoders. Encoders process input sequences and encode data to show information about them and their characteristics. Decoders do the opposite, they process the information received from the encoder and generate output sequences. All encoders have the same structure and consist of two layers: self-attention and feed-forward neural network. The input sequence being fed into the encoder first passes through the layer of internal attention, which helps the encoder to look at other words in the input sentence while encoding a particular word. The output of this layer is sent to the feedforward neural network. The same network is applied independently to each word. The decoder also contains these

two layers, but in between there is an extra layer of attention that allows the decoder to identify the relevant parts of the input sentence.

Internal attention allows the model to see dependencies between the word being processed and other words in the input sequence, which help to better encode the word.

After all decoders, a fully connected Softmax layer is used, which converts the obtained values into probabilities, from which the largest value is then selected, and the word corresponding to it becomes the output for this time step.

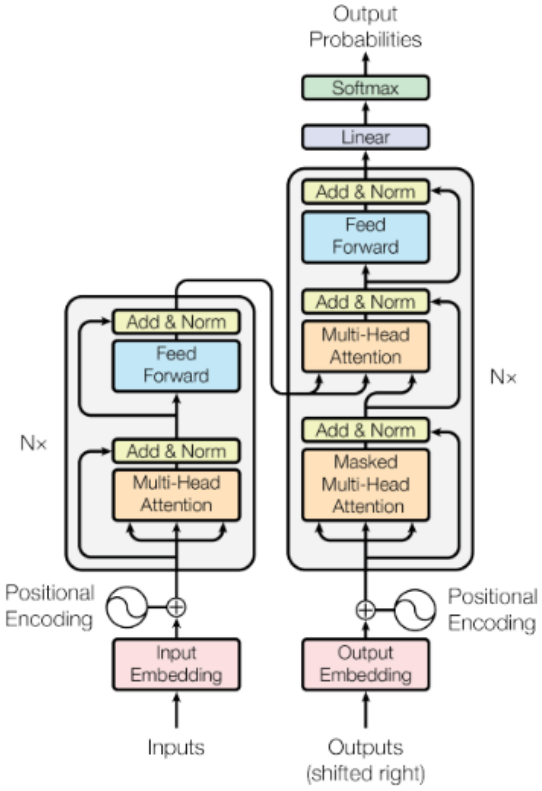

Fig. 7. Transformer architecture.

5.1 GPT-2 architecture

GPT-2 is a large language model based on Transformer, created by the non-profit company OpenAI, with parameters ranging from 117 million to 1.5 billion, trained on a dataset of 8 million web pages [20]. GPT-2 learns with a simple goal: to predict the next word, given all the previous words in some text.

GPT-2 is built using only decoder blocks, which have the same structure as the Transformer model described above.

GPT-2 does not use words as input but tokens obtained using the Byte Pair Encoding (BPE) method. It is a data compression technique in which the most common pairs of

consecutive bytes of words are replaced by bytes that do not appear in those words [23]. This method provides a balance between character and word representations, which allows it to cope with large corpuses of data.

Internal attention in GPT-2 also uses masking, which blocks information from tokens to the right of the position that is being calculated.

5.2 GPT-2 implementation

A medium size GPT-2 model with 345 million parameters was used, consisting of 24 decoder blocks.

The model was further trained using fine-tuning on the corpus of search queries in English which were also used to train recurrent neural networks. Using the resulting model search queries were generated.

We used the model implementation available at https://github.com/nshepperd/gpt-2. The model was trained for 1000 steps.

6 Latent semantic analysis

Latent Semantic Analysis (LSA) is a natural language processing technique for analyzing dependencies between collections of documents and the terms they contain [24].

This method uses a term document matrix that describes the frequency of occurrence of terms in a collection of documents. The elements of such a matrix can be weighted, for example, using TF-IDF: the weight of each element of the matrix is proportional to the number of times the term occurs in each document, and inversely proportional to the number of times the term occurs in all documents in the collection. After compiling the term-document matrix, its singular value decomposition is carried out, i.e. it is represented as $A = USV^T$, where matrices U and V are orthogonal, and S is a diagonal matrix, the values of which are called singular values of matrix A. This expansion reflects the basic structure of dependencies present in the original matrix, allowing to ignore noise [25].

6.1 Implementation of latent semantic analysis

To carry out latent semantic analysis, the gensim library for Python was used [26]. We created a corpus of 10,000 documents containing human-written reference searches. Frequently occurring official words of the English language (prepositions, articles) and words that occur once were then removed from it, since they do not help to calculate the semantic relationship between documents. Using the Dictionary class of the gensim library, a dictionary was created with words and their indices, then using the doc2bow method of this class, all documents were presented in a bag of words format. The TFIDF model was applied to the resulting data corpus, and the LsiModel class performed singular value decomposition. The requests generated using neural networks were tokenized and, using a dictionary created on the reference corpus, transformed into a bag

of words format. Finally, using the MatrixSimilarity class, semantic similarities between these corpuses were calculated using a cosine measure.

7 Results of evaluating generated queries

Comparing each document, in this case a query, with documents from the corpus with real queries, the method returns a value from -1 to 1, reflecting the semantic similarity of the documents. The analysis results are shown in Table 2.

Table 2. Results of latent semantic analysis.

	GRU	LSTM	Fine-tuned GPT-2
Mean	0,0065	0,006	0,0035
Average number of values greater than 0,7	16	14	9
Average number of values greater than 0	4000	4659	2684

The corpus of real queries is varied, so the average value of the result of comparing each generated document with all documents from the reference corpus differs slightly from zero. At the same time, for each query artificially created using GRU and LSTM networks, there are on average 16 and 14 semantically close documents, when the values are greater than 0.7, and for the GPT-2 model, this number is 9 documents. Also, for each request generated by the GPT-2 model, out of 10,000 compared documents, 2684 have a value greater than 0, and for LSTM and GRU networks, 4659 and 4000, respectively. From this, we can conclude that LSTM and GRU used more words semantically similar to words from the training data when generating queries than GPT-2. This makes sense, since the first two models were trained from scratch on the input data, while the main training of the last model took place on a completely different corpus, it was only fine-tuned in order to generate queries suitable for structure. It is also important to take into account that the comparison was carried out with 10 thousand reference queries, although the models were trained on 100 thousand, therefore not all dependencies were taken into account, however, the obtained values are sufficient for analysis.

The analysis results show that the generated queries have similar semantics to the corpus of real user queries, but at the same time they do not repeat them literally, that is, they are new queries in meaning.

The GRU and LSTM networks were trained by characters and could have generated non-existent words, so it was decided to test them. Each word from the queries was checked for existence using a corpus containing more than 466 thousand English words, available at https://github.com/dwyl/english-words. In the queries generated by the GRU network, 141 words out of 4431 were not found, and in the queries of the LSTM

model - 166 out of 4325. The words that were not found contained typos or mistakes in words that the models remembered. Therefore, it may be worth preprocessing the data by correcting typos and errors of this kind. However, queries with typos can be useful depending on the task in which they will be applied. So, for example, when they are used to test a new search engine or to optimize it, they will be more relevant with typos, as they have a greater similarity with real user queries.

Due to the fact that neural networks cannot understand the meaning of a sentence, although they often find the correct dependencies between tokens, an expert (manual) analysis was carried out to assess the quality of the generated search queries.

From the queries generated by each model, 100 queries were randomly selected. It was determined whether each search query makes sense, whether it is similar to a real possible user query. It should be noted that this assessment is subjective. Queries were considered "good" if the words in them were consistent with each other.

The analysis results are shown in Table 3.

Table 3. Results of expert analysis.

	GRU	LSTM	Fine-tuned GPT-2
Good query	72	73	81
Bad query	28	27	19

The table shows that the GRU and LSTM networks showed almost the same results, while GPT-2 is slightly better. During the analysis, it was observed that the GPT-2 model generates shorter queries than the other two models.

The results of the analysis showed that the GRU and LSTM networks have approximately the same quality when solving the task of generating search queries, and the GPT-2 model was worse in automatic analysis, but better in expert judgment. Therefore, this model is better suited for generating search queries, since the significance of the expert judgment is higher than the automatic one, although for more accurate results it is worth carrying out this assessment with the help of other experts.

8 Conclusion

In the course of this work, we researched the leading models used to generate texts in natural language and their ability to solve the task of generating queries for search engines and we conducted their comparative analysis. Two neural networks are fully implemented: a network with a long short-term memory and a network with a gated recurrent unit. The GPT-2 architecture based on the Transformer model was researched; it was also fine-tuned using the corpus of real user requests.

Latent semantic analysis showed that the GPT-2 model performs worse than the other two networks. However, the automatic metrics for evaluating the generated text do not always reflect the quality of the model, since at the moment it is impossible to assess the meaningfulness of the texts using the algorithm. To solve this problem, an

expert analysis of the generated texts was also carried out, according to the results of which the GPT-2 model was better than the other two models. At the same time, the LSTM and GRU networks showed approximately the same quality according to the results of all analyses performed.

Acknowledgments. This work was subsidy of the Russian fund of fundamental research, grant agreement 18-07-00964.

References

1. van Deemter, K., Krahmer, E., Theune, M.: Real vs. template-based natural language generation: a false opposition? (2005) https://wwwhome.ewi.utwente.nl/~theune/PUBS/templates-squib.pdf, last accessed 2020/06/15
2. Xie, Z.: Neural Text Generation: A Practical Guide (2017) https://arxiv.org/pdf/1711.09534.pdf, last accessed 2020/06/15
3. A Comprehensive Guide to Natural Language Generation (2019) https://medium.com/sciforce/a-comprehensive-guide-to-natural-language-generation-dd63a4b6e548, last accessed 2020/06/15
4. Arrington, M.: AOL proudly releases massive amounts of user search data (2006) https://techcrunch.com/2006/08/06/aol-proudly-releases-massive-amounts-of-user-search-data/, last accessed 2020/06/15
5. Reiter, E.: NLG vs Templates: Levels of Sophistication in Generating Text (2016). https://ehudreiter.com/2016/12/18/nlg-vs-templates, last accessed 2020/06/15
6. Gagniuc, P.: Markov Chains: From Theory to Implementation and Experimentation. USA, NJ: John Wiley & Sons (2017).
7. Press, O., Bar A., Bogin B., Berant J., Wold L.: Language Generation with Recurrent Generative Adversarial Networks without Pre-training (2017). https://arxiv.org/pdf/1706.01399.pdf, last accessed 2020/06/15
8. Williams, R., Hinton G., Rumelhart D.: Learning representations by back-propagating errors (1986). http://www.cs.utoronto.ca/~hinton/absps/naturebp.pdf, last accessed 2020/06/15
9. Hochreiter, S., Bengio Y., Frasconi P., Schmidhuber, J.: Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies (2001). https://www.bioinf.jku.at/publications/older/ch7.pdf, last accessed 2020/06/15
10. Hochreiter, S., Schmidhuber, J.: Long-Short Term Memory (1997). http://web.archive.org/web/20150526132154/http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf, last accessed 2020/06/15
11. Heck, J., Salem, F.: Simplified Minimal Gated Unit Variations for Recurrent Neural Networks (2017). https://arxiv.org/abs /1701.03452, last accessed 2020/06/15
12. Bahdanau, D., Cho, K., Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate (2016). https://arxiv.org/pdf/1409.0473.pdf, last accessed 2020/06/15

13. Felbo, B., Mislove, A., Søgaard, A., Rahwan, I., Lehmann, S.: Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm (2017). https://arxiv.org/pdf/1708.00524.pdf, last accessed 2020/06/15
14. Bisong, E.: Google Colaboratory. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform (2019) Apress, Berkeley, CA.
15. Chollet, F.: Keras (2015). https://keras.io, last accessed 2020/06/15
16. Kingma, D., Ba, J. Adam: A Method for Stochastic Optimization (2014). https://arxiv.org/abs/1412.6980, last accessed 2020/06/15
17. Learning Rate Scheduler. https://keras.io/api/callbacks/learning_rate_scheduler/, last accessed 2020/06/15
18. Schuster, M., Paliwal, K.: Bidirectional recurrent neural networks (1997). https://www.researchgate.net/publication/3316656_Bidirectional_recurrent_neural_networks, last accessed 2020/06/15
19. Vaswani, A., Shazeer, N., Parmar, N.: Attention Is All You Need (2017). https://arxiv.org/pdf/1706.03762.pdf, last accessed 2020/06/15
20. Radford, A., Wu, J., Child, R., Luan, D.: Language Models Are Unsupervised Multitask Learners (2018). https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf, last accessed 2020/06/15
21. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018). https://arxiv.org/pdf/1810.04805.pdf, last accessed 2020/06/15
22. Brown, T., Mann, B., Ryder, N., Subbiah, M.: Language Models Are Few-Shot Learners (2019). https://arxiv.org/abs/2005.14165, last accessed 2020/06/15
23. Gage, P.: A New Algorithm for Data Compression (1994). https://www.derczynski.com/papers/archive/BPE_Gage.pdf, last accessed 2020/06/15
14. Deerwester, S., Harshman, R.: Indexing by Latent Semantic Analysis (1987). https://www.cs.bham.ac.uk/ ~pxt/IDA/lsa_ind.pdf, last accessed 2020/06/15
25. Nakov, P.: Getting Better Results with Latent Semantic Indexing (2009). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.6406 &rep=rep1&type=pdf, last accessed 2020/06/15
26. Rehurek, R., Soika, P.: Software Framework for Topic Modelling with Large Corpora (2010). Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. University of Malta.