UDC 004.8
DOI: 10.20535/SRIT.2308-8893.2024.4.06

ON THE EVOLUTION OF RECURRENT NEURAL SYSTEMS

G.S. ABRAMOV. I.V. GUSHCHIN, T.O. SIRENKA

Abstract. The evolution of neural network architectures, first of the recurrent type and then with the use of attention technology, is considered. It shows how the approaches changed and how the developers' experience was enriched. It is important that the neural networks themselves learn to understand the developers' intentions and actually correct errors and flaws in technologies and architectures. Using new active elements instead of neurons expanded the scope of connectionist networks. It led to the emergence of new structures — Kolmogorov–Arnold Networks (KANs), which may become serious competitors to networks with artificial neurons.

Keywords: recurrent neural networks, transformer technology, KANs.

INTRODUCTION

In modern programming there are three degrees of formalization. The first is codes, the second is languages, for example, the most common language for neural network developers is Python. Third, these are libraries that, in addition to data and dictionaries, have a large range of technologies. You actually turn to such technology, mark it in the code, enter the necessary data and it does everything itself. Instead of hundreds of lines of the program, there are dozens left. Moreover, the complexity of the program only increases taking into account libraries. Humanity is increasingly moving into the category of users, since less and less attention is paid to basic formal primary knowledge and descriptions, and people are already using derivatives, such auxiliary structures for describing knowledge. Only a few people are interested in the basics of science, but without such people progress will stop. This article is an example of the efforts of such smart and ambitious people, inventors looking for new opportunities.

For example, this is a matrix recording, a vector with a large number of components is immediately supplied to the network input — a whole set of queries, transformations in the network array occur in matrix form, while fortunately it is linear. The next aspect is the use of the attention mechanism that arose in developed recurrent networks.

Parallel calculations of vectors and matrices due to advanced CUDA technologies on video cards, as well as the use of matrix notation itself, are essential. speeds up calculations. Each of these methods: 1) matrix notation; 2) parallel computing on video cards; 3) the attention mechanism speeds up the work of

modern artificial neural networks by approximately an order of magnitude. It is not surprising that all this has made it possible to move from human-controlled machine learning to deep learning, which is implemented by the network itself, which has acquired new qualities.

However, the very structure of connectionist networks, that is, networks consisting of many active elements with a set of free parameters that allow it to learn or learn, can develop towards a computation graph in the most general sense. These are KAN networks (Kolmogorov–Arnold Networks), and other networks of this type may yet appear.

But it is interesting to consider how people thought when creating language models. These models first took the form of recurrent networks, and then, when the attention mechanism appeared there, networks with "transformer" technology. But the most important are the methods and methods of creating new devices and technologies. *This work is dedicated to this urgent problem*.

Usually, people have a set of data — the history of processes and want to predict the future. Formally, we are talking about a known distribution of probabilities $P(x_t \mid x_{t-1},...x_1)$ on these data, an estimate of the conditional expectation $\mathrm{E}[(x_t \mid x_{t-1},...x_1)]$ (here conditional means that the expectation of the value $x_{t-1},...x_{t-\tau}$) should be found if the conditions for the appearance of the values before) are met. Linear regression is usually used for this. usually only an undetermined number of these previous quantities is confusing, although this problem can be solved by choosing a window τ — the length of this sequence of data (this is attributed to Markov models, for example — orders that take into account the sequence $x_{t-1},...x_{t-\tau}$). If it is possible to somehow summarize the previous data and this summary is marked as $h_i = g(h_{i-1}, x_{t-1})$, then it is possible to enter into the previous forms of description of the forecast $\hat{x}_t$, i.e. $\hat{x}_t = P(x_t \mid h_i)$. Here, a summary appears in the description, which in recurrent models of neural systems is formed by the network itself, and therefore this summary is often called a hidden description (probably because the network does not give users a clear view).

CLASSIC RECURRENT NETWORKS

The idea of the original linguistic classical recurrent networks[1] (RNN) (their recurrent nature is that they constantly use what was known before) is to step by step supplement the text with the most likely next word. At each step, the output h_t, which depends on the inputs x_{t-1} x_{t-2}, … x_{t-m} at the previous steps[2], is calculated. Since it is not desirable for users to find an explicit view h_t, it is easier to call it a hidden description. Later data values in these models depend on earlier ones. Architecturally, a recurrent neural network is a chain of repeating modules. Dictionaries began to be used — embeddings, which represent words in vector

[1] The active use of such architectures tends to be attributed to S. Hochreiter and his colleagues in the early 90s of the last century.

[2] To select the influence of previous words on the following ones (selection of internal memory with a state vector s_t), gate architectures are used, for example, Long Short-Term Memory (LSTM) and gate recurrent unit (Gated Recurrent Unit - GRU). In the practice of creating recurrent LSTM networks [1; 2], blocks were used to improve the transfer of information from previous iterations of recurrent RNN networks.

form, and the distance between vectors depends on how often these words correlate with each other in texts (sets of which are often called corpora).

In recurrent networks of the classical type, the probability and frequency of a pair-triple of neighbouring words from the dictionary was found, and the integral probability of the sentence or phrase maximized $p(x_1,...x_T)$ by the network was formed, by expanding it into a product of conditional densities from left to right, applying the chain rule of probability:

$$P(x_1,...x_T) = P(x_1)\prod_{t=2}^{T} P(x_t \mid x_{t-1},...x_1). \quad (1)$$

You can find the conditional probability for the entire depth of memory $P(x_t \mid x_{t-1},...x_{\tau>1})$, or $P(x_t \mid x_{t-1})$ the length of the corpus of words (1), as well as Markov approximations of different depths of memory , and even , then the probability of the entire sentence, phrase, or corpus will be according to the choice of Markov models, for example τ — orders that take into account the sequence $x_{t-1},...x_{t-\tau}$):

$$P(x_1,...x_T) = P(x_1)\prod_{t=2}^{T} P(x_t \mid x_{t-1},...x_1),$$

$$P(x_1,...x_T) = P(x_1)\prod_{t=2}^{T} P(x_t \mid x_{t-1},...x_{\tau>1}), \quad (2)$$

and even

$$P(x_1,...x_T) = P(x_1)\prod_{t=2}^{T} P(x_t \mid x_{t-1}).$$

Here, respectively, the length of the data sequence T, τ and the length equal to one are chosen.

By selecting words from the dictionary, the network searches for the maximum conditional probability of individual parts of the corpus and the entire corpus. The main task of the classical recurrent network is to generate text, to select words that are most similar to previous phrases and sentences. Further development takes into account a certain summary of past calculations, replacing

$$P(x_t \mid x_{t-1},...x_1) \to P(x_t \mid h_t),$$

and updating the form of these hidden (from developers) states $h_i = g(h_{i-1}, x_{t-1})$. These models were also called latent autoregressive models (see, for example, [3]).

RECURRENT NETWORKS WITH ENCODER AND DECODER FOR TRANSLATION

More modern recurrent networks are even bidirectional (they remove the problem of using only previous data), still form sentences sequentially word by word with the maximum of first local (digram probability, for example), then integral maximum conditional probability (1) or (2), but now they have an encoder and a decoder, which are capable of forming an initially poor-quality translation between language A (encoder) and language B (decoder) when using dictionaries. A phrase in language A is presented to the encoder. A phrase in language B is formed from the dictionary in the decoder.

The translation procedure: 1. The probability of finding a pair of words next to each other is estimated (from previous training of the network). 2. The frequency of appearance of this pair in the studied samples is estimated. 3. The overall probability of a phrase or text fragment is formed.

In the encoder, sequences of hidden states $\vec{h}_t = f(\vec{x}_t, \vec{h}_{t-1})$ are formed, hidden from people, and collected into a context vector $\vec{c} = q(\vec{h}_1,...,\vec{h}_T)$ for language A, which is presented in the encoder. In the decoder for language B, the output sequence $(y_1,...,y_T)$ for each time step t' (we use *t'* in contrast to the time steps of the input sequence of the encoder *t*), the decoder assigns a predicted probability to each possible word (token) occurring at step *t'*+1 determined by the previous tokens in the target object $(y_1,...,y_{t'})$ for language B and adds the context vector, i.e.

$$P(y_{t'+1} \mid y_1,..., y_{t'}, \vec{c}) . \quad (4)$$

Prediction of the next lexeme t'+1 in the target sequence: the RNN decoder takes the target marker (the marker here is the *y* value *y*) of the previous step, the hidden state of the RNN from the previous time step $h_{t'-1}$, the context vector $\vec{c}$ as input, and translates them into the hidden state at the current time step $h_{t'}$. Already in this description, it is clear that even the developers do not know how exactly this was done, but they almost understand the structure and nature of the transformations. The revolutionary action at this step of evolution was the use of two networks — encoder and decoder, which are respectively connected to dictionaries of different languages.

RECURRENT NETWORKS WITH ATTENTION MECHANISM

Using the Bahdanau attention mechanism (https://d21.ai/) allows you to use the information obtained not only from the last hidden state, but also from any hidden state h_t of the encoder for any iteration *t* (runs from 1 to m). With the help of the attention mechanism (accepts the hidden states of the encoder h_i and the hidden state of the decoder $\vec{s}_{t'-1}$ creates a weighted estimate *s* from the sum of the states of the encoder) "focusing" of the decoder on certain hidden states of the encoder is achieved. In cases of machine translation, this capability helps the decoder predict which hidden states of the encoder, given the output of certain words in language A, should be paid more attention to when translating this word into language B.

The attention mechanism was first used in the Seq2seq[3] (sequence-to-sequence) network for machine translation (Machine Translation — MT) [4].

The layer of the attention mechanism is a single-layer neural network, which is fed not the final hidden value, but all such values h_i (t =1, ...*m*), as well as the hidden value of the decoder $\vec{s}_{t'-1}$ at its previous step (iteration). The output of the attention layer is the value of the vector ***s*** (score). This will actually be the weight of the hidden value h_i. Softmax is used to normalize *s*. Then $e = \text{soft}\max(\boldsymbol{s})$. Now the context vector takes the form

[3] In fact, Seq2seq technology has changed the nature of the recurrent network of the previous type.

$$c = \sum_{i=1}^{m} e_i h_i .$$

Thus, the result of the work of the attention layer is the context vector ***c***, which is constantly changing during the calculation process and includes information about all hidden states of the encoder weighted by attention. Transferring a constantly corrected context vector to the decoder improves, as practice has shown, the quality of translation due to changing the context of the encoder and decoder. The main idea of the attention mechanism is that instead of storing a state that summarizes the encoder's original sentence, the network dynamically updates it as a function of the original text (encoder hidden states h_i) as well as the translation text that has already been generated (hidden states decoder $\vec{s}_{t'-1}$). This gives a new context vector ***c***, which is updated after any decoding step t'. The main thing is that already at this stage of the development of neural language models, researchers stop understanding the meaning of transformations of vectors describing sequences. It is argued that "models of attention provide "interpretability", although what exactly the weights of attention mean ... remains a nebulous topic of research".

Then they found an opportunity to use new developments of the attention mechanism used in the "Transformer" technology to form the context vector. In this variant, the context vector c is the result of the combination of attention (the layer of the attention mechanism is then not needed):

$$c_{t'} = \sum_{t=1}^{T} \alpha(s_{t'-1}, h_i) h_i ,$$

used here as a query $S_{t'-1}$, and h_i as a key, and as a value in the terms and designations of the "Transformer" technology.

"TRANSFORMER" TECHNOLOGY

In the "Transformer" technology (see, for example, [5–7]), which already replaces all previous translation systems, the attention mechanism allows you to abandon the recurrent mechanism of forming phrases and corpora "from word to word" and exclude LSTM and GRU blocks, now the sentence is considered all at once. Therefore, the principle of recurrence is no longer needed.

Now all the hidden states of the encoder $h(t)$ are passed to the decoder, which forms the attention weights for the initial sequence. During token prediction, if not all input tokens are relevant (fit), the model considers more of the input sequence that is considered relevant to the current prediction. So to speak, he focuses his attention on them.

With the emergence of the attention mechanism, there is a need for coding, which replaces the numbering of words (tokens). It is possible to enter trigonometric functions — modes (for example, with the wave number $k = 2\pi / L$) on the body L (the number of sentences and words), the multiplication of which does not yield the absolute value of vectors from the interval (0–1). The matrix that numbers the tokens is added to the matrices used in calculations. The need to use this matrix is due to the fact that it is necessary to restore the sequence formation procedure, which was previously automatically implemented in the recurrent scheme.

Next, we denote $D \overset{def}{=} \{(\vec{k}_1, \vec{v}_1),...,(\vec{k}_m, \vec{v}_m)\}$ the database *m* of tuples of keys $\vec{k}_i$ and values $\vec{v}_i$. In addition, we denote $\vec{q}$ as request[4] . This approach, it is believed, helps to form the principles of creating the attention mechanism of the "Transformer" technology

$$Attention\ (\vec{q}, D) \overset{def}{=} \sum_{s=1}^{m} \alpha(\vec{q}, \vec{k}_i)\vec{v}_i,$$

where $\alpha(\vec{q}, \vec{k}_i)$ are the scalar weights of attention. All values of hidden states are multiplied by these weights, and form a weighted sum of values. Note that the scalar weights are chosen quite phenomenologically, using, for example, the most famous Gaussian kernel [8], which describes the characteristic dimensions of the distances between words

$$\alpha(\vec{q}, \vec{k}_i) = \vec{q}^T \vec{k}_i / d\ . \tag{3}$$

Note that attention weights still need to be normalized. We can simplify this with the softmax operation:

$$\alpha(\vec{q}, \vec{k}_i) \to \text{soft}\max(\alpha(\vec{q}, \vec{k}_i)) = \frac{\exp(\vec{q}^T \vec{k}_i / d)}{\sum_j \exp(\vec{q}^T \vec{k}_j / d)}.$$

In this way, it was possible to move to a more effective analysis of sentences both individually and in texts (corpus). However, some problems remained, the solution of which led to the appearance of important mechanisms not only for coding, but also, importantly, for improving the style and quality of translation.

AUXILIARY MECHANISMS OF TRANSLATION

Construction of multi-head attention

Vectors corresponding to text elements are divided into several fragments, which are treated in the same way as whole vectors. This approach, where each of the H_i outputs of the attention pool is a head, was made largely to use parallel computing, which was considered more productive. So far, mathematicians are thinking about the correctness of such an approach, practical results have already shown its effectiveness.

In practice, with the same set of requests, keys and values, it is possible to divide different ranges of changes and enter different subspaces of the representation of requests, keys and values. Actually divide the vectors into parts. To this end, instead of performing a single attention merge, queries, keys, and values can be transformed into a set of queries, keys, and values served in parallel. Such a design is called multi-headed, where each of the H_i outputs of the attention pool is a head [7].

In addition, the researchers discovered that, just as in the case of using several encoders and decoders, each such calculation channel is independently filled with a different meaning, and the network creates these so-called ranges and subspaces in a form that is sometimes incomprehensible to developers.

[4] Key, request, value — this is the structure that seemed more understandable to developers. It is not a fact that such a representation will be preserved in the future.

Given a query $\vec{q} \in \mathrm{R}^{d_q}$, a key $\vec{k} \in \mathrm{R}^{d_k}$, and a value $\vec{v} \in \mathrm{R}^{d_v}$, each head with attention H_i is calculated as

$$\vec{H}_i = f(\vec{W}_i^q \vec{q}, \vec{W}_i^k \vec{k}, \vec{W}_i^v \vec{v}) \in \mathrm{R}^{p_v},$$

where $\vec{W}_i^q \in \mathrm{R}^{p_q \times d_q}$, $\vec{W}_i^k \in \mathrm{R}^{p_k \times d_k}$, $\vec{W}_i^v \in \mathrm{R}^{p_v \times d_v}$ are input parameters R^p, R^d — show the dimensionality of vectors and matrices and *f* is an attention pool, such as additive attention. It is surprising, but such an action, initiated by too determined developers of neural networks, does not lead to nonsense, but gives quite reasonable results. How it works out in the network still needs serious research.

Self-attention

In addition to the attention used between the encoder and the decoder, each of them needs so-called self-attention, or internal attention. This is practically the same as the classic recurrent network, but in a form that has already become the basis for the Transformer technology. This self-attention now works differently [7], and elsewhere is described as a model of internal attention [9].

The same elements of input or output sequences alternately play the role of queries, keys, and values.

The authors of many works give an example.

Thus, when translating the sentence “Student is studying a transformer”, the word “Student” is the first query, and the key is “studying”. The scalar product of the corresponding vectors of the hidden representation gives the attention score of this pair, which will then be multiplied by the value, i.e., the vector representation of the word “learns”. In the next passage, the query will be the word “learns”, and the key may be the word “transformer”.

As a result, according to expression (3), an attention score will be formed for all request/key pairs, by which all value vectors of the input sequence will be multiplied. The encoder context vector will now be first multiplied by these self-attention weights, and then sent to the decoder. And in the decoder, even after all transformations, it is rational to use self-discipline to avoid inconsistency of the translation with the basics of this language. Self-attention allows you to rework a faithful but not very literary text into a rather attractive and more acceptable one for the reader.

In this mechanism, a query is the name of what needs to be found. Keys are signatures on folders and blocks in the middle of the filing cabinet. Having found the appropriate folder, we can get it and find out the content — the value vector. But in the case of internal attention, one must look for not one value, but a significant number of values from a set of folders. Multiplying the query vector by each of the key vectors will give us the coefficients for each folder (technically: a scalar value followed by a softmax function, i.e. converting this value into a unit interval that makes sense of probability). By adding up all the values with their coefficients, you can get the result of internalattention.

CONCLUSIONS

The process of developing neural networks continues and looks like a strange search method, more intuitive than strictly logical. If at first neural networks were

created by neurophysiologists who understood how the brain works, then mathematicians joined this process, but they were not really listened to.

However, the rapid development of computing systems, advances in parallel computing (see, for example, [10]), and a significant amount of memory have made it possible to more boldly form neural network architectures and technologies. And technologies appeared on the scene, people who were more focused on the technical development of networks. They created such complex and large systems that a different approach to their understanding and presentation was needed. It turned out that the initiative in the development of neural networks is already moving to the neural networks themselves, which are capable of correcting the defects and weaknesses of people's technological innovations, independently finding methods of correcting weak human decisions. An illustration is the creation of the Transformer technology, which is quite inaccurately made by humans, but the neural network itself found methods to correct inaccuracies and inaccuracies and demonstrated a remarkable ability to present users with the result they desired.

The development of neural and similar networks with active elements did not stop there. In fact, Tsybenko's theorem (Universal approximation theorem), which allows approximating any continuous function with a set of neurons with activation functions and a significant number of inputs and outputs, can be used for more general networks with active elements. The main thing is to be able to make the necessary functional connection between the inputs and outputs of the network, which is possible if there is an opportunity to teach the network.

Therefore, it is not surprising that the idea and the first attempts to create a network appeared, where the active elements are spline functions (multiple-polynomial functions that can consist of different polynomials at different segments) [11]. For each spline, more polynomial coefficients need to be introduced, so the new network created from them — KANs (Kolmogorov-Arnold Networks), which very boldly uses the theorem of these famous mathematicians, needs more parameters than exist in networks based on artificial neurons (there the parameters are weights and displacement).

However, it turned out that much fewer layers could then be used. You will have to teach these polynomial functions and this seems to be easier, but it will take longer, and increasing the coefficients of the polynomials will even improve the capabilities of such a network. Such networks are more suitable for solving problems in mathematics. Against the background of such innovations, the achievements of the developers of "transformer" technology no longer seem so significant, especially since mathematicians did not see the mathematical rigor in its architecture.

Even the limitation associated with the problem of using processing on GPUs also turned out to be a solvable problem. Such modified networks were called ReLU-KAN [11], they turned out to be faster than expected and more accurate, which was a pleasant surprise. All these hopes of the developers were confirmed by the practice of using these networks.

In conclusion, it can be noted that in general, the creation of networks with active elements, with customizable connections — such a computational graph — can be implemented in different ways, the main thing is that there are free parameters for its appropriate optimization and the possibility of using parallel computing to speed up learning and use. Although it should be understood that the

amount of internal memory and the complexity of the tasks will still require a large number of active elements and network parameters.

REFERENCES

1. S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.
2. J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling.* 2014. Available: https://arxiv.org/pdf/1412.3555
3. I.V. Gushchin, O.V. Kirychok, and V.M. Kuklin, *Introduction to the methods of organization and optimization of neural networks: a study guide.* Kh.: KhNU named after V. N. Karazin, 2021, 152 p.
4. E. Charniak, *Introduction to deep learning*. Massachusetts: The MIT Press Cambridge, 2019, 192 p.
5. D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by joint learning to align and translate.* 2014. Available: https://arxiv.org/abs/1409.0473
6. I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *International Conference on Machine Learning*, pp. 1139–1147, 2013.
7. A. Vaswani et al., "Attention is all you need," *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.
8. E.A. Nadaraya, "On estimating regression," *Theory of Probability & its Applications*, 9(1), pp. 141–142, 1964. doi: https://doi.org/10.1137/1109020
9. A.P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, *A decomposable attention model for natural language inference*. 2016. Available: https://arxiv.org/ pdf/1606.01933
10. V. Gushchin, V.M. Kuklin, O.V. Mishin, and O.V. Pryimak, *Modeling of physical processes using CUDA technology*. Kh.: V.N. Karazin KhNU, 2017, 116 p.
11. Z. Liu et al., *KAN: Kolmogorov-Arnold Networks*. 2024. doi: https://doi.org/ 10.48550/arXiv.2404.19756

Received 01.03.2024

INFORMATION ON THE ARTICLE

Gennadii S. Abramov, ORCID: 0000-0003-0333-8819, Kherson State Maritime Academy, Ukraine, e-mail: gennadabra@gmail.com

Ivan V. Gushchin, ORCID: 0000-0002-1917-716X, Kharkiv National University named after V.N. Karazin, Ukraine, e-mail: i.v.gushchin@karazin.ua

Tetiana O. Sirenka, Kharkiv National University named after V.N. Karazin, Ukraine

ПРО ЕВОЛЮЦІЮ РЕКУРЕНТНИХ НЕЙРОННИХ СИСТЕМ / Г.С. Абрамов, І.В. Гущин, Т.О. Сіренька

Анотація. Розглянуто еволюцію нейромережевих архітектур, спочатку рекурентного типу, а потім із використанням технології уваги. Показано, як змінювалися підходи та збагачувався досвід розробників. Важливо, що нейронні мережі самі навчилися розуміти наміри розробників і фактично виправляли помилки та недоліки в технологіях і архітектурах. Використання нових активних елементів замість нейронів розширило сферу застосування конекціоністських мереж і призвело до появи нових структур — мережі Колмогорова–Арнольда (KAN), які можуть стати серйозними конкурентами мереж зі штучними нейронами.

Ключові слова: рекурентні нейронні мережі, технологія трансформер, KAN.