KAPITEL 1 / *CHAPTER 1* [1]
ON THE EVOLUTION OF NEURAL NETWORKS

DOI: 10.30890/2709-2313.2025-42-04-001

Introduction

The problem of organizing the processes of imagination, reflection and prediction in humans has worried scientists since ancient times. Since most scientists were religious people, it was not surprising for them that the human brain and its nervous system provide control over all organs, but thoughts, as they imagined, are associated with the spiritual, immaterial sphere. Since the spirit was responsible for the thought process, even the nature of the appearance of intelligent beings on the planet (the emergence of the second signal system) was not discussed, because the religious idea of the origin of the world was widespread. A detailed study of the brain was required, the involvement of mathematicians in thinking on this topic, and, most importantly, the first attempts to create connectivist models of the brain from artificial neurons in order to dispel the disembodied fog of the spirituality of thinking. Indeed, a small number of neurons in the model turned out to be able to find answers, that is, generate an opinion. There was no longer room for a disembodied spirit [1]. Modern neural networks have three degrees of formalization. The first is famous codes, the second is languages, for example, the most common language Python. The third is libraries, which, in addition to data and dictionaries, have a large set of technologies. When you choose some technology, you just turn to it, it does everything itself. Instead of hundreds of program lines, there are dozens. Moreover, the complexity of the program, taking into account the libraries, only increases. Humanity is increasingly moving into the category of users, because less and less attention is paid to the main formal basic, fundamental knowledge and descriptions and uses derivative reference books, subsidiaries. Only a few are interested in the basics of science. Indeed, programming neural networks is greatly simplified. The second aspect of the development of neural networks is matrix

[1] ***Authors:*** *Kuklin Volodymyr Mykhailovych*
Number of characters: *135225*
Author's sheets: *3,38*

notation, a vector of a large number of components is fed to the network input at once - a whole set of queries, transformations into massive networks are in matrix form, fortunately linear. The third aspect is the use of the attention mechanism, say, in the transformer technology, this method, as well as the use of parallel calculations of vectors and matrices due to the developed CUDA technology [2] on video cards, and the matrix notation itself significantly speeds up the calculation. Each of them: matrix notation, parallel calculations on video cards and the attention mechanism speeds up the work of modern networks by an order of magnitude. It is not surprising that all this 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 [3-5].

Below we will dwell on the main areas of development of neural networks, including the mainstream of this scientific and technical development, that is, rapidly evolving language models, their application in the life of society, as well as the revived neural networks, which previously occupied narrow niches of public attention, capable of solving problems of modeling and scientific research.

1.1 History of the creation of modern neural networks - language models

Statistical models. The task of artificial intelligence, among other things, was the construction of context-consistent ones proposed on the basis of the probability of the frequency of co-occurrence of different words in corpus-texts. The number, the set of variable words on which this word depends is called the context, as a rule, if the text contains at least three words. It is possible more, but at first it became a problem for storing such contextual compositions and using them. On the other hand, large contexts are less common. This approach is used in "smart" keyboards that try to prompt the next word. This approach was also used in trained neural networks for processing new data and obtaining predictions or conclusions (that is, for inference procedures). But this approach is not effective for large sentences, the machine gets confused when using words that are rarely used or rarely located next to each other. Then they began to understand the text in more detail. The entered token is a unit of text understood by

machines. They began to divide texts into words by spaces, punctuation marks, excluded stop words, and so on, for example, using CountVectorizer (a method of creating a sparse representation of a set of tokens using scipy.sparse.csr_matrix). Although at the same time there were duplicates of tokens that referred to the same word or inadvertently reduced the word to a simplified initial form, losing case and time. Later, tokenizers based on the BPE (Byte Pair Encoding) algorithm appeared, and the next SentencePiece tokenizer was already more advanced, it used the logic of Unigram and the same BPE tokenizers.

Recurrent neural networks (RNNs), which provide better quality text with accelerated inference than models based on statistics, allowed a context of hundreds of words, but had problems with learning. Looking ahead, we note that two networks were used for translation purposes - the architecture consisted of two parts: encoder and decoder, respectively, for two different languages. BERT models were built on the basis of encoder blocks, which were already able to be trained in parallel. Encoders analyzed the connections between tokens and, if using the attention mechanism that appeared later, highlighted important tokens.

Returning to the early RNN models in the absence of a decoder and attention mechanisms, we note that these networks predicted the next token based on the previous ones. By inference, when the next token is generated, it is added to the context and a new token is selected based on the previous text. In these neuronets, which are a black box, the history of the processes was input, and the goal was to predict their evolution. If we go to the formal description, then at first there was a known distribution of probabilities of the manifestation of these processes $P(x_t \mid x_{t-1},...x_1)$ and an estimate of the conditional expectation $\mathrm{E}[(x_t \mid x_{t-1},...x_1)]$, and here the conditional expectation means that the expectation of the value x_t should be found if the conditions for the appearance of the earlier values $x_{t-1},...x_{t-\tau}$ are fulfilled. When using linear (for ease of description) regression, it is important to determine the number of these previous values and choose a contextual window - that is, the length of this sequence of data (this is often attributed to Markov models, for example, the τ–order that takes into

account the sequence $x_{t-1},...x_{t-\tau}$). When summarizing the preliminary data and it is designated as $h_i = g(h_{i-1}, x_{t-1})$, it is possible to enter previous forms of description of the forecast $\hat{x}_t$, i.e. $\hat{x}_t = P(x_t \mid h_i)$ and there is a summary, which in recurrent models of neural systems is formed by the network itself, and therefore this summary is often called a hidden description (because the network does not show the obvious view to users, we suspect that this is not only because there is no necessary interface).

Initial language classical recurrent networks (RNN) step by step supplemented the text with the most probable next word. at each step, a hidden from the user exit was calculated h_t on the basis of inputs x_{t-1} x_{t-2}, … x_{t-m} from the previous steps[2]. Since this is a black box, users were not shown a clear view of intermediate results h_t and all these procedures are easier to call a hidden description. The later values of the data in these models depend on the earlier[3] ones. Architecturally, a recurrent neural network is a chain of repeating modules. it turned out to be rational to use dictionaries - embeddings that represent words in vector form, and the distance between vectors depends on how often these words correlate with each other in texts (corpora). In recurrent networks of the classical type, the probability and frequency of the appearance of a pair or three of neighboring words from the dictionary in the external text arrays was determined, and the integral probability of a sentence-phrase was formed $p(x_1,...x_T)$, which the network sought to maximize by decomposing it into a product of conditional probabilities from left to right, applying the chain rule of probability: $P(x_1,...x_T) = P(x_1)\prod_{t=2}^{T} P(x_t \mid x_{t-1},...x_1)$.It is possible to introduce Markov approximations of different memory depths

[2] The initial use of such architectures is usually attributed to S. Hochreiter and his colleagues in the early 90s of the last century.

[3] To take into account the influence of previous words on subsequent ones (the state vector), gate architectures are used, for example, long short-term memory (LSTM) and gate recurrent unit (GRU). In the practice of creating LSTM [6,7], blocks are used to dose the transmission of information in the RNN system.

$P(x_t \mid x_{t-1},...x_{\tau>1})$. Then the probability of the entire offer for Markov models τ – order that takes into account the sequence $x_{t-1},...x_{t-\tau}$) will be presented as $P(x_1,...x_T) = P(x_1)\prod_{t=2}^{T} P(x_t \mid x_{t-1},...x_1)$.Using the dictionary, the network searches for the maximum conditional probability of individual parts of the corpus and the entire corpus. The main goal of the classic recurrent network is to generate text, to find words that are most similar to the previous phrases and sentences. Further development takes into account a certain summary h_i past calculations, replacing $P(x_t \mid x_{t-1},...x_1) \to P(x_t \mid h_t)$ and renewing the form of hidden states $h_i = g(h_{i-1}, x_{t-1})$. Such models are usually called autoregressive [8].

Later recurrent models, which were used to translate from one language to another, actually contained two neural networks - an encoder and a decoder. Such recurrent models, even bidirectional ones (they made it possible to better match words in sentences), still formed a sentence sequentially word by word with first the local maximum, then the integral maximum conditional probability of the sentence and the corpus, but now they actually had more sets of encoders and decoders. A phrase in language A is represented in the encoder. Based on the hidden components of the encoder and the dictionary, a phrase in language B is already formed in the decoder (decoder). First, the possibility of finding a pair of words next to each other (from the preliminary training of the network) is estimated. Then the frequency of occurrence of this pair in the training samples is determined. as a result, an integral possibility of a phrase or a text fragment-corpus is formed.

Formally, in the encoder, sequences of hidden components $\vec{h}_t = f(\vec{x}_t, \vec{h}_{t-1})$, assembled into a context vector for language A $\vec{c} = q(\vec{h}_1,...,\vec{h}_T)$, are created. In the decoder for language B, its own sequence appears $(y_1,...,y_T)$ for each time step t' (t' is used 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 the step t'+1 determined by the previous tokens $(y_1,...,y_{t'})$ in and adds the context vector $\vec{c}$, i.e. $P(y_{t'+1} \mid y_1,...,y_{t'},\vec{c})$. Prediction of the next lexeme t'+1 in the target sequence is as follows. The RNN decoder accepts the target marker (the marker here is the value of y) of the previous step, the hidden state $h_{t'-1}$ of the RNN from the previous time step, the context vector $\vec{c}$ as input data, and translates them into the hidden state at the current time step. Although the procedure seems to be clear, even the developers do not know the details, because they hardly understand the structure and nature of the transformations. Progressive at this stage of development of recurrent models at this stage of evolution was the use of two networks - encoder and decoder, respectively connected with dictionaries of different languages.

The next stage in the development of recurrent models was the use of the Bahdanau attention mechanism (see https://d21.ai/ for details). This mechanism for processing words in sentences [9] turned out to be very promising. It made it possible to use the information obtained not only from the last hidden state, but also from any hidden state of the encoder for any iteration (t runs from 1 to m). The attention mechanism accepts the hidden states of the encoder and the hidden state of the decoder creates a weighted estimate s from the sum of the encoder states. At the same time, the emphasis of the decoder is formed on certain hidden states of the encoder. During machine translation, this helps the decoder to consider what hidden states of the encoder in the words of language A should pay more attention when translating this word into language B.

The attention mechanism was initially introduced into the Seq2seq (sequence-to-sequence) word evaluation technology of the Machine Translation (MT) network [10]. The attention mechanism is a single-layer neural network to which no longer a finite hidden value is fed, but all similar values h_i (t =1, ...m), as well as the hidden value of the decoder $\vec{s}_{t'-1}$ at its previous step. The output of

the attention layer is the value of the vector s (score). This is the weight of hidden meaning. Softmax is usually used to normalize s. Then it is possible to enter into consideration the vector e = softmax(s) and the context vector takes the form $c = \sum_{i=1}^{m} e_i h_i$. The result of the calculation of the attention layer is the context vector c, constantly changing during the calculation process, which includes information about all the hidden states of the encoder that have already been weighted by attention. Such dynamic transmission of the constantly adjusted context vector to the decoder improves, as practice has shown, the quality of translation due to the change of the encoder and decoder context. The main idea of the attention mechanism is that instead of saving a state summarizing the encoder's original sentence, the network dynamically updates it as a function of the original text (hidden states of the encoder), as well as the text of the already generated translation (hidden states of the decoder). This gives a new context vector c, which is updated after any decoding step t′. Already at this stage of the development of neural language models, researchers finally stop understanding the meaning of transformed vectors describing sequences.

"Transformer". It is possible to abandon the sequence of consideration of words in a sentence by using the "Transformer" technology (see, for example, [11,12]), which 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. When using the "Transformer" technology, the mechanisms of self-attention and cross-attention become decisive.

In this technology, the hidden states *h*(t) of the encoder are transmitted to the decoder, which first of all forms the value of the attention weight for the original sequence. When predicting a token, the model takes into account most of the input sequence, which is considered relevant, that is, corresponding to the current prediction.

True, when using the attention mechanism, there is a need for coding that replaces the numbering of words (tokens). It is possible to introduce trigonometric functions -

modes (for example, with a wave number) on corpus L (the number of words proposed), multiplication by which does not derive 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 necessity of using this matrix is connected with the sequence formation procedure, which was previously automatically implemented in the recurrent scheme.

The technology database consists of m tuples of keys and values and let it be written in the form $D =^{def} \{(\vec{k}_1, \vec{v}_1), ...(\vec{k}_m, \vec{v}_m)\}$. For this database, we will denote requests $\vec{q}$,, keys $\vec{k}_i$ and values $\vec{v}_i$ .. Such a structure seemed understandable to the developers and, as they believed, it helped to form the principles of creating the attention mechanism of the "Transformer" technology: $Attention(\vec{q}, D) =^{def} \sum_{s=1}^{m} \alpha(\vec{q}, \vec{k}_i)\vec{v}_i$, where $\alpha(\vec{q}, \vec{k}_i)$ are the scalar weights of attention $(i = 1, ...m)$. The values of the hidden components should be multiplied by these weights to form the sum of values weighted in this way. It is important to note that the scalar weights are chosen quite phenomenologically. It is possible, for example, to use the most famous Gaussian kernel, which is able to determine the characteristic dimensions of the distance between words in this metric $\alpha(\vec{q}, \vec{k}_i) = \vec{q}^T \vec{k}_i / d$

By the way, recurrent networks also began to use a similar approach by modifying the cross-attention mechanism to form the context vector. In this variant, the context vector s is the result of combining attention (the layer of the attention mechanism is then not required): $c_{t'} = \sum_{t=1}^{T} \alpha(s_{t'-1}, h_i)h_i$ here it is used as a request from the decoder $s_{t'-1}$, and h_i as a key and as a value from the encoder in the terms and designations of the "Transformer" technology. Let's return to "Transformer". It was necessary to simplify and normalize the scalar weight using the softmax operation. Note that the weight of attention still needs to be

normalized. We can simplify it with the help of the softmax operation: $\alpha(\vec{q},\vec{k}_i) \to soft\max(\alpha(\vec{q},\vec{k}_i)) = \frac{\exp(\vec{q}^T\vec{k}_i / d)}{\sum_{j=1}\exp(\vec{q}^T\vec{k}_j / d)}$ These approaches allowed us to move to a more effective analysis of both individual and large texts (corpus). However, first in the encoder, a self-attention estimate is formed for all request/key pairs, by which all value vectors of the input sequence will be multiplied. Here the request is the current meaning of the word in the encoder, and what is the key and what is the meaning. The vector of the encoder context will now be first multiplied by these self-attention weights, and then already sent to the decoder. And in the decoder, after all the conversion, it is rational to use self-awareness to avoid the mismatch of the translation to the basics of this language. Self-attention allows you to rework a correct, but not very literary text into a sufficiently attractive and more acceptable one for the reader [13].

Due to the large size of the vectors corresponding to the words-elements of the corpus, the information there is certainly duplicated, and therefore it turned out to be possible to unobstructedly divide each vector into several, which accelerated parallel calculations. By dividing the vectors into several fragments each, they are treated in the same way as whole vectors were treated. So far, the validity of this procedure has not been strictly proven, but practical results have already shown its effectiveness. In fact, the vectors were divided into parts. For this purpose, instead of performing a single focus, requests, keys and values can be transformed. into a set of requests, keys and values that are served in parallel. Such a construction is called multi-headed, where each of the outputs *h* of attention is a head [14]. In addition, the researchers found that, just as in the case of using several encoders (encoders) and decoders, each such calculation channel is independently filled with a different meaning, and these so-called ranges and subspaces are sometimes created by the network in a form that is incomprehensible to developers. But the results can be combined in different ways, which also leads to better evaluation and good results. Transformers have surpassed networks with convolutional architecture.

Notes on parametric optimization of models. Even a formally correctly created network architecture does not always work if some auxiliary settings are not made.

Generally speaking, different activation functions can be used. There is a fairly large set of such functions. Although modern large models used well-proven activation functions ReLU (Rectified Linear Unit) and Leaky ReLU (Rectified Linear Unit). Neural networks traditionally use the gradient descent method for training $w_{n+1} = w_n - \gamma \nabla J(w_n)$, where $J(w_n)$ is the error function, is the hyperparameter (γ rate of learning) for training. A modification of the gradient descent is often used by optimizers, in particular, Momentum, RMSProp and Adam. The problem is that when forming logits in layers of traditional language networks, when small weight values appear, the gradients disappear, if the weight values are large, the gradients explode. To combat this, the weight initialization procedure is used. For example, Xavier initialization produces a variance of the layer's output data equal to the variance of its input data. Kaiming Xe initialization with the ReLu activator is suitable for combating vanishing and explosive gradients. But more often they use the Adam optimizer. (Adaptive Moment Estimation), which stabilizes gradients and speeds up calculations. ADAM = Momentum Adaptive +Learning Rate, where Momentum (impulse) helps to move in the forward direction, despite the noise, which always exists in the system of discrete procedures, and can also appear during calculations. Adaptive Learning Rate (organizes the adaptive learning rate) - regulates the update steps. If the standard gradient descent $w_{t+1} = w_t - \eta \cdot \nabla L(w_t)$; $\nabla L(w_t)$ - the gradient of the loss function. Impulse update Momentum $m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1)\nabla L(w_t)$ and $w_{t+1} = w_t - \eta \cdot m_t$, where $\beta_1 \approx 0.9$. Adaptive learning rate (RMSprop) $v_t = \beta_1 \cdot v_{t-1} + (1-\beta_2)(\nabla L(w_t))^2$ and also $w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t} + \varepsilon} \nabla L(w_t)$, where $\beta_2 \approx 0.99$. In other words, the procedures are as follows:

1. $m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1)\nabla L(w_t)$ - gradient

2. $v_t = \beta_1 \cdot v_{t-1} + (1-\beta_2)(\nabla L(w_t))^2$ - squared gradient,

adaptive rate

3. $\hat{m}_t = m_t \left(1-\beta_1^t\right)^{-1}$, $\hat{v}_t = v_t \left(1-\beta_2^t\right)^{-1}$ bias correction

4. $w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t$ weight update

ADAM demonstrates noise resistance, converges quickly, everything is done automatically, you can call it in PyTorch like this

Import torch.optim as optim

optimizer = optim.Adam(model.paramenrs(), Ir=0.001, betas= (0.0, 0.999))

Even a radical method of dealing with gradient explosion is useful - its cutting off in both types of networks (gradient clipping) $\| \nabla w \| < \tau$ - if nothing changes, if $\| \nabla w \| > \tau$, then $\nabla w = \nabla w \cdot \frac{\tau}{\| \nabla w \|}$ does not allow gradients to explode, you can call it in *PyTorch*

torch.nn.utils.clip_grad_norm_(model.paramenrs(), max_norm=1.0)

and in TensorFlow/Keras:

optimizer = tf.keras.optimizers.Adam(clipnorm= 1.0)

Excessive training precision also weakens the performance of networks, so regularization is used in both types of networks. These are methods for weakening the closeness of the setting to the training set $w_{i,n+1} = w_{i,n}\left(1-\gamma\lambda / m\right) - \gamma \partial J(\vec{w}_n) / \partial w_i$. where λ is a hyperparameter. In traditional language models, the following methods and technologies are used (see, for example, [15]. For classification problems, the Hebb method $w_{ij}(t) = w_{ij}(t-1) + \alpha y_i^{(q-1)} y_j^{(q)}$ and the Kohonen algorithm $w_{ij}(t) = w_{ij}(t-1) + \alpha \cdot [y_i^{(q-1)} - w_{ij}(t-1)]$ are used, which allow you to enhance the trajectory of a noticeable signal and select clusters from a data set. Among the

possibilities to help the neural network, transfer learning is distinguished - this is the use of a part of the donor trained network - the first few functions (fees) using the Bellman equation. A very effective reinforcement learning method is "selective self-adjusting adaptation", which uses signals about successes and failures, instead of training examples (collected in Q-learning by K. Watkins). To improve efficiency, constructive algorithms are used - to add neurons (instead of one, they put two, this is splitting), as well as pruning algorithms - they remove synapses (zero out the weighting coefficients that fall into a given range of values). Plugins are used to improve the analytical capabilities of language networks. You can use mathematical calculation technologies. For example, Wolfram|Alpha - a free version of the engine - a knowledge base and a set of algorithms for calculation (2009), works in interactive mode. Calculates a request based on its own knowledge in the Mathematica language. For phones and tablets, a page m.wolframalpha.com has been created, where you can enter mathematical formulas for calculation. By the way, Siri in the Apple iOS operating system supports this service. It is important that this engine translates questions from natural language into a format that computers understand.

The emergence of modern language models. If encoders stimulated the creation of new recurrent models, then decoders, in turn, gradually led developers to GPT models. They are also trained to predict the next token based on the previous ones. But their interaction with the encoder during translations is important.

The GPT-1 model, based on the Transformer technology, was trained in 2018 on 7,000 books and had a context size of 512 tokens. Benchmarks noted the best quality at that time. CoLA, which determines grammatical correctness, showed that the results increased to 45.4 compared to the result of 35.0 for RNN, in GLUE - from 72.8 to 68.9. A similar GPT-2 model was trained in 2019. Success was predetermined, because the Transformer technology used by the developers could approximate very complex dependencies and was very easy to train. GPT-2 consisted of a record 1.5 billion parameters for that time, that is, network weights, which means it exceeded its predecessor by these values by an order of magnitude, had a context of 1024 tokens and was trained on 40 GB of text data. GPT-2 became a leader in many benchmarks.

GPT-2 turned out to be able to generate reasonable texts - and not just sentences, but sometimes there were failures. However, the developers began to understand that if you take a large text corpus and train a neural network on it, the results will only improve. In addition, it became clear that if the programs describing the network structure began to rapidly decrease in size to hundreds of lines, due to the development and expansion of libraries, then the training process ("obtaining a Master of Laws" - LLM) became unaffordable for amateurs. The next model, GPT-3, broke the records of the previous ones in quality and size. In 2020, the network with 175 billion parameters was created, which was trained on 570 GB of text data with a context of 2048 tokens. The model was able to provide translation, answers to questions, with a quality close to the level of an educated person, and generated quite creative content. It was confirmed that the quantity and quality of training data are the most important factors determining the advantages of the final model. Compared to the previous model, 3200 GPUs were used for training instead of 16, that is, a small cluster. During the training process, it was possible to formulate some useful details.

In 2023, most models accepted 4096 or 8192 tokens as input, with the exception of Claude 2.1. Material about Claude 2.1 is in the claude.ai chat. Claude 2.1 accepted 200 thousand tokens in the context window, which allows translating approximately 150,000 words or more than 500 pages of material, in addition, a 2-fold decrease in the number of hallucinations, demonstrated a 30% decrease in the number of incorrect answers compared to previous model Claude 2.0. A beta feature was also added there that allows Claude to integrate with existing processes, products, and user APIs (a software interface, a set of rules and specifications for exchanging data). The Workbench product provides access to new model settings to optimize Claude's behavior. It was already possible to create code snippets for using hints in the SDK (a set of development methods, auxiliary programs for performing specialized tasks - that is, utilities and documentation). Models surpassing GPT-3 were explosively created by Anthropic, Mistral, Google, Meta, EleutherAI, Stability AI, TII in Abu Dhabi (Falcon), Microsoft Research, xAI, Replit, Baidu and several other organizations. By the way, Google joined the race about a year and a half late after OpenAI.

Details of training and additional training. During training, they started using leads (verbal description of the task, a simple task) with low-scoring examples,, which is similar to downstream training. It was noticed that as the model grows, the learning ability increases sharply. The form of the question also turned out to be important.

Thus, first a text description of the task, a sufficient number of examples in the lead and the choice of their order in the few-shot format. It is important to create a system of queries (Prompt Engineering) and after a certain stage move on to fine-tuning (Prompt Tuning). To save GPU RAM, they usually try to train only a part of the network (part of the parameters) - the optimal promt. You can split the answer into several stages and view them, which diagnoses the solution process and determines the effectiveness of the model. This approach (Chain-of-Thought, CoT) is based on the consideration of the chain of thoughts-solutions. The stage of training the neural network on a huge array of high-quality or specialized data is pretraining, then training to follow instructions (SFT - supervised finetuning), training politeness and details (training the reward model). At the Reinforcement Learning (RL) stage, the language model is trained to generate such answers that would have the highest score relative to the reward model (numerical scores).

Although LLM creation is still not available for amateurs to learn from scratch, it is possible to refine some models. Looking ahead, we note that Georgy Gerganov released the code that launched the Llama model on a MacBook. This caused an explosion of innovation, especially after Meta released Llama 2, an improved version that was allowed to be used commercially. LLMs began to appear that can be run locally on a variety of devices. Importantly, they can be modified and further trained. Nowadays, people are able to train acquired models based on basic data, publish the results of this further training, create entire methods and data sets for refinement, and actively share them. It is enough to look at the Hugging Face Open LLM table - one of the resources where you can view this data. Any changes to the table become obsolete in a few hours. Any LLM model with an open license at any point in time is usually no longer the basic one: it is usually already a modified, refined model. This is an undeniable advantage of open models over closed ones, which do not develop as a result of competition between thousands of researchers willing and able to improve them.

They started to place trained fairly large language models even on laptops. For example, Simon Willison, according to him, placed GPT-4 class models (Qwen2.5-

Coder-32B with Apache 2.0 license, Meta's Llama 3.3 70B) on a laptop - a 64 GB MacBook Pro M2, 2023, but they took up all the memory space. The fact that they were able to be placed speaks to the effectiveness of training. Meta's Llama 3.2 models do not belong to the GPT-4 class, but in sizes 1B and 3B they run on iPhone using the free MLC Chat iOS app (<2 GB).

The quality of the models was checked using benchmarks. Unfortunately, none of them will answer the question of whether LLM is really suitable for a specific task. Only long-term communication with the model will build confidence in answering this question. The poor quality of the hints is confusing, perhaps you need to learn to highlight the main thing in them so that the model understands it.

Developers will have to use the mathematical benchmarks MultiArith and GSM8K. They can used TriviaQA, a question-and-answer benchmark based on Wikipedia. The Lambada benchmark evaluates the memorization (results are saved - cached for subsequent quick use) of the long context of the model[4].

After this, the ChatGPT model appeared, and the basis of ChatGPT was initially the GPT-3.5 neural network, an improved version of GPT-3, developed in 2020. In September 2023, OpenAI used the Whisper speech-to-text model and the tts-1 text-to-speech model to interact with ChatGPT mobile apps, but it only saw text. The introduction of this model was a grand show, with many promises made, such as that ChatGPT could understand mathematics, which was actually implemented using a third-party service - a plugin like WolframAlpha.

Then the model with the number GPT-4 became the subject of discussion. The version of the latest chatbot is sometimes called ChatGPT-4. But it can be used by those who have subscribed to ChatGPT Plus for $ 20 per month. But GPT-4 is also built into the free version of Microsoft's search engine, called Bing Chat. In 2023, no alternative model has appeared that would be better than GPT-4. Google's Gemini Ultra, which appeared, was not available for testing. The Mistral team has also been

[4] Methods were given on how to independently train language models from the very beginning. https://libeldoc.bsuir.by/bitstream/123456789/52690/1/Kasyan_Ogranicheniya.pdf https://skillbox.ru/media/code/zakrytyy-ii-ot-openai-issleduem-neyroset-gpt4/

working to beat GPT-4. And OpenAI released GPT-4 in March, although it was later revealed that it could be seen in February, when Microsoft used it in the new Bing.

While the basic version of GPT-4 had a context window of up to 8,192 tokens, the extended version GPT-4-32k increased it to 32,768 tokens (50 pages of text). GPT-4 was reportedly trained on Microsoft Azure supercomputers. The process used the RLHF method, also used in previous versions of GPT-3.5 and ChatGPT. This is the so-called "reinforcement learning based on human feedback," the essence of which in this case is teaching the network how to teach experts how to conduct dialogues, who then monitor these dialogues. The network's conversations with people, the number of which was tens and then for GPT-4 hundreds of millions of people who had previously talked to ChatGPT and acted as teachers for GPT-4. No model in the world had ever had so many volunteer assistants before.

Immediately after the release of GPT-4 in the spring of 2023, the prevailing view was that the process of scaling models would lead to the emergence of general artificial intelligence. Below, we show that developers were expecting problems along this path. But the scaling of models over the past 7 years has been significant, because it was enough to increase the amount of data, model size and computing power; no breakthroughs in algorithms were required.

The problem of running out of data for training[5], which accompanied scaling, that is, increasing the number of model parameters, also seemed to be solved. In particular, it turned out that a token generated by a language model is better predicted by previous tokens, making it easier for the model to follow reasoning patterns (for more details, see the Phi-4 white paper, which reads like a pitch deck with a sales pitch). So often, large models were able to create training data for their smaller, cheaper alternatives. For example, DeepSeek v3 used synthetic data generated by DeepSeek-R1. On the other hand, the process of further scaling training and improving quality has unfrozen, at least in the minds of developers, and we can expect to see new giant models.

[5] LLM learns from internet data. Colleagues from Epoch Ai believed that only 20% of internet data is suitable for training. The danger of using up all text data between 2025 and 2028 was a concern. There is no more natural data. {Although there are still pictures/video/audio, but developers have not yet learned how to use them for training.

The Chatbot Arena rating (Vibe Benchmark) has been continuously updated with new models since 2024 (these are models from Google, OpenAI, Alibaba, Anthropic, Meta, Reka AI, 01 AI, Amazon, Cohere, DeepSeek, Nvidia, Mistral, NexusFlow, Zhipu AI, xAI, AI21 Labs, Princeton and Tencent). First came Google's Gemini 1.5 Pro, with an input context of 1 million tokens (later 2 million) and the ability to input video. But it should be noted that longer input data significantly increases the volume of problems. For example, long text is often perceived by models partially: a little at the beginning and a little at the end of the corpus. The same thing with translation, part of the array is lost and it is not easy to find what was lost. Although, on the other hand, long input data, in principle, allows you to more accurately set the problem and avoid absurdities in answers, for the resolution of which clarifications and hints would otherwise be required. It is also sometimes useful to enter lots of code examples into the context window to help solve the coding problem [16]. The GPT-4o ("omni") model could take audio input and output realistic speech without using TTS or STT. The models learned dialects and distorted pronunciations, slang. It is already possible to share and discuss with the models (ChatGPT, Google Gemini, OpenAI with WebRTC API) the video sequence you are watching. It is great that LLM can create a full-fledged interactive application for the code using HTML, CSS and JavaScript. Anthropic released Claude Artifacts, an interactive application on demand, and allowed it to be used in the interface. The ability to run hints, corrections and discussions on images (as well as audio and video) is an exciting new way to use these models. The LLM CLI tool for supporting multimodal models via attachments already has plugins for a whole collection of different vision technologies. For a few months in 2024, the three best available models — GPT-4o, Claude 3.5 Sonnet, and Gemini 1.5 Pro — were available for free. However, this practice will be reduced in time, if not eliminated altogether. GPT-4o can already perform web search or use Code Interpreter, in the same ChatGPT UI. US regulations on GPU exports to China have forced Chinese developers to learn to skimp on training, in particular DeepSeek v3, one of the largest open-license models currently available, on par with Gemini 2.0 and OpenAI 4o/o1. The model was trained on 2,788,000 H800 GPU hours at an estimated cost of

$5,576,000. (Llama 3.1 required 30,840,000 GPU hours — 11 times more.) This may force to abandon locally located insanely expensive clusters for training in the West.

From the very beginning of the evolution of neural networks, developers - engineers pushed aside the pioneers of this industry - neurophysiologists, and later scientists - mathematicians and began to experiment by changing the structure and functions of neural networks, practically by the enumeration method. High efficiency in the evolution of neural networks was initially explained by a very large number of participants, someone, yes, will come up with or suggest how exactly to change the program or the operating methodology of this or that node or circuit. Modern models, the internal structure of which actually changes little, except for scaling - that is, the growth of the number of parameters (weights) are invented by engineers, constantly experimenting and making changes on large and very expensive clusters that were built by other engineers. The changes being made are already of a different nature and other technological innovations that slightly affect the structure of such networks, these are training methods, the creation of additional control and regulatory structures and principles of communication with neural networks that have gained strength.

1.2 Main engines of artificial intelligence evolution - scaling processes

Learning scaling. Since progress in the field of neural networks is driven by engineers relying on empirical procedures, and there is no coherent and intelligible theory yet, the main areas of research are scaling the network itself and learning, as well as the recently developing scaling of inference. The efficiency of the model, as it turned out, depends on three values: C - how many calculations were used for learning, D - the size of the dataset, N - how many parameters are in the model. [17].

Each of the three graphs is obtained under the assumption that no restrictions were imposed on the other two parameters. Any network, its architecture, is scaled, that is, the number of parameters, neurons, vector dimensions, context window size, etc. increases. But different networks scale differently, so it is better to choose networks whose graphs have a greater slope on these three graphs. With the growth of these three

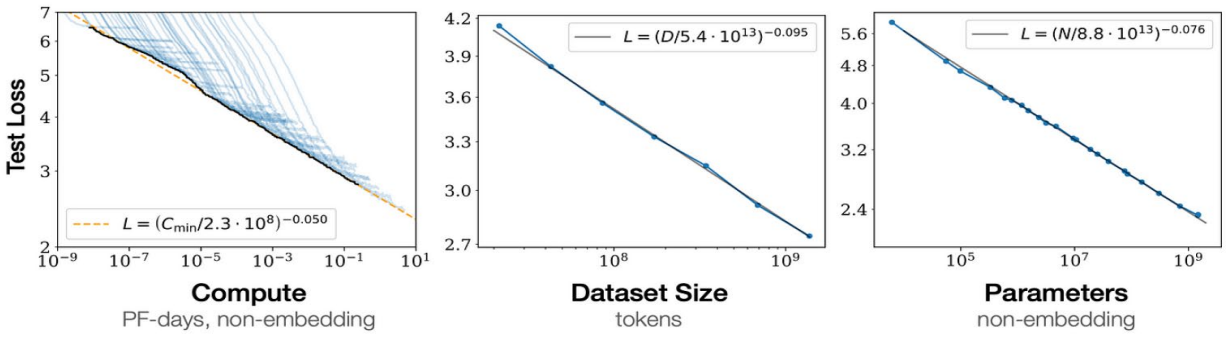


Fig. 1 - Model errors for different values of C, D, N. Source: [17]

values, the translation becomes more accurate, the answers to questions are more correct, difficult problems begin to be solved, the text generated by the model becomes more difficult to distinguish from the text written by a person.

Since the lack of natural data can be compensated for by synthetic data, training methods will gradually become cheaper if we apply the scheme used by the Chinese for parallel connection of thousands of training centers for joint work, then there is a further race for scaling ahead, although there are signs that this path of development is not so effective.

However, some disappointment occurred because the new, larger models were not much better than GPT-4, although they acquired some new qualities. Now, after the difficulties, especially financial and organizational, as well as environmental (problems with insufficient provision of clusters for training with electricity) in creating new generations of models (OpenAI, Anthropic and Google Gemini), even I. Sutskever, as Reuters reports, believed that[6] "Scaling the right solution is now more important than ever." Therefore, developers switched to scaling inference. If scaling training, associated with gigantic costs, gives way to other directions of development, this (scaling the right solution) will be a promising direction. If AI progress will again depend on new ideas, and not only on computation; large companies, startups and research scientists can possibly compete on a relatively equal footing.

[6] Sutskever's interest in scaling has decreased after moving to Safe Superintelligence with the more modest financial capabilities of this startup, and therefore it is not surprising that he is looking for new competitive advantages, while focusing on the exhaustion of data for pre-training, perhaps to form reasonable doubts about further scaling

Problems of insufficient memory. The disappointing assessment of the insufficient increase in memory in the article [18] was also actively discussed among developers, where it was noted that over the past two decades, the maximum computing power of hardware has increased by 60,000 times, while the power of DRAM (a type of memory used in random access memory devices) has increased only by 100 times, and the bandwidth of connections has increased by even 30 times. The authors of the work believe that "memory — in particular, data transfer within/between memory chips — will soon become the main limiting factor in serving large AI models. Therefore, we need to rethink the training, deployment, and design of AI models, as well as how we design AI hardware, to cope with this increasingly complex memory wall."

Indeed, using LLM is efficient in processing long texts, but the "Transformer" architecture in modern networks has not linear but quadratic complexity, which significantly increases the computational burden as the length of the input texts increases. To solve this problem, a KV-Cache mechanism was proposed, which stores the keys and values generated by previous tokens. This reduces the time complexity from quadratic to linear. However, KV-Cache increases the GPU memory usage, which scales with the text length, creating a new bottleneck. Researchers from Wuhan University and Shanghai University presented several KV-Cache compression techniques. These techniques optimize the space usage of KV-Cache during LLM pre-training, deployment, and inference, with the aim of improving efficiency without sacrificing performance. The authors of [20] also showed how to simplify low-precision KV cache quantization, including several new techniques. The code is available at https://github.com/SqueezeAILab/KVQuant. What if we can no longer increase the capacity for training models? Apparently, we will have to increase the capacity for prediction, scale the model output (test-time compute).

Scaling the output, reasoning. OpenAI models o1, o3 ("o2" is omitted for irrelevant reasons) are the next step in the LLM architecture for solving much more complex problems. Similar models are gemini-2.0-flash-thinking-exp, QwQ, DeepSeek-R1-Lite-Preview . You can make the model explain the problem it solves,

which improves the result, but you can embed such explanations into the model itself, forcing it to "think" about intermediate results. Here, the computations during the inference process also increase significantly. The current dominant opinion is that model scaling is obsolete, and "inference scaling", partly similar to "test-time computing scaling", is the way to advance AI capabilities. There are limitations, though: inference scaling based on symbolic reasoning seems useful for tasks with clear right answers, such as programming or solving math problems.

Perhaps it is better to spend more and more computational resources when using models to perform a task ("thinking" before answering). The question arises: does this slow down progress in the development of AI capabilities? Inference scaling is possible, and many simple solutions are proposed, in the short term. Other very promising methods use techniques such as generating multiple solutions and ranking them by quality. While the laws of traditional scaling are already known, the laws of inference scaling are not yet known.

Example 2-1[7].: A neural network, AlphaGo Zero, which is many orders of magnitude smaller than modern LLMs, learned from its own games and was able to play Go better than humans. AlphaGo Zero was taught to think before each move, and not try to give an answer right away. It first made a probabilistic estimate of how a move could be made, and then ran the Monte Carlo TreeSearch algorithm, which estimated the probable results after each move and the success of different moves. Clearly, such an algorithm requires computation, but not during the training of the model, but during the prediction.

The o1 model, formerly known as Q* or Strawberry, is designed as a reasoner (LRM) capable of scaling the amount of computation used depending on the request. The o1 model has shown amazing results in complex problems such as mathematics and programming, outperforming all its predecessors, but it takes longer to respond

[7] Examples provided by the channel author @vikulin_ai.

and is more expensive to use. The consideration of the planning and scheduling capabilities in the context of scaling inference can be demonstrated by embedding the LRM in a loop with a reliable verifier, similar to the LLM-Modulo framework. For planning, it is sufficient to use PlanBench, which consists of both special test suites and a set of tools designed to evaluate language models in arbitrary inter-process communication planning domains. To evaluate the planning capabilities, tests were conducted on TravelPlanner, on three Natural Plan domains, and on graph coloring problems. One can focus on classical planning problems or STRIPS planning problems, which are a formalism for automated planning in discrete, deterministic spaces.

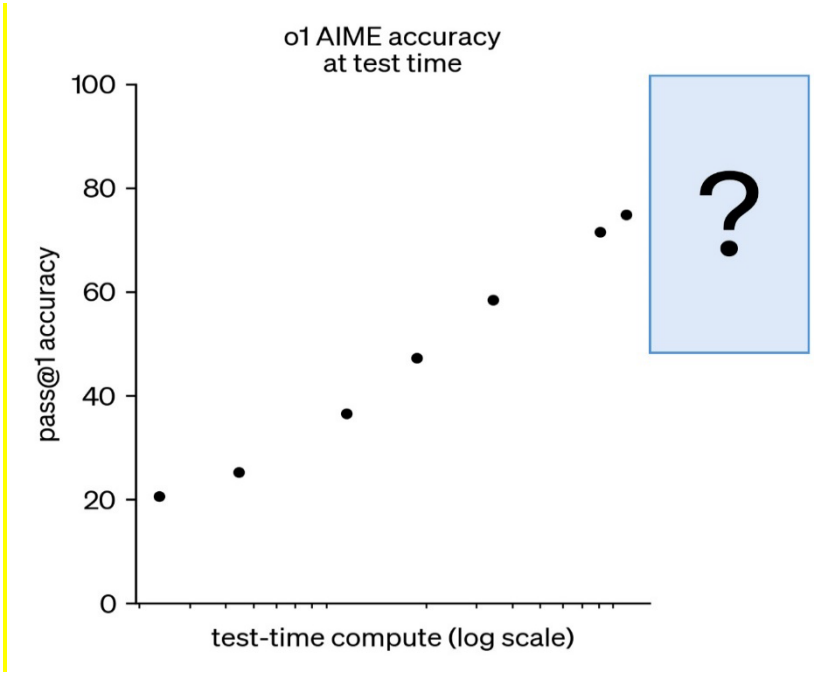
Example 2-2: The developers required the DeepSeek-Coder-V2 model to generate hundreds of answers to a series of programming problems, and then checked the solution that passed all the unit tests. With such an incredible large-scale increase in calculations, not the most powerful model, as witnesses claim, managed to surpass the most powerful at that time GPT-4o.

It is interesting to look at the dynamics of comparisons of the capabilities of o1, the new so-called LRM, in various planning and scheduling benchmarks in the context of scaling inference. In [21], the authors showed that these models coped with the Blockworld test set and showed early signs of progress on obfuscated versions, while in a slightly later paper [22], o1 achieved impressive results in previously unmatched benchmarks. We are talking about progress in planning tasks, showing significantly better results in graph coloring than previous models, this progress was unevenly distributed, and only in the Ohio State University Travel Plan and the Natural Plan benchmark set was this progress evident. It is noted that o1 uses significant computational resources for inference and charges a noticeable fee for this. On longer problems and in attempts to determine the solvability of potentially unsolvable cases, it is found that this increase in accuracy is neither general nor robust.

Example 2-3: In 2023, OpenAI releases an article [23] in which they build a verifier and teach the model to reason. Most likely, this article is a precursor to o1, which was published in 2024. The authors limited themselves to solving mathematical problems. In the article, they promptly made the model reason step by step, and then each step of reasoning was identified and evaluated - how much closer this step is to solving the problem or is it erroneous. To do this, they marked the PRM800K dataset, which turned out to be 800 thousand marked computational steps. It was on this dataset that they trained the newly created verifier, which must predict the correctness of each calculation step. Once such a verifier is created and trained, you can already choose a solution that has no or very few logical and computational errors. Everything is just like in games. It turned out that this method allows you to increase the accuracy of solving mathematical problems by a quarter.

On the accuracy of output scaling. AIME, a mathematical benchmark [24] became the judge of OpenAI, which demonstrated the capabilities of o1. Their graph leaves this question open. Will the performance reach a limit or can it be brought closer to 100%? The graph omits the designations on the abscissa axis, which is left to the conscience of the developers.

It is known that the accuracy of AI models on the AIME (American Invitational Mathematics Examination) test varies depending on the model and its training, with models being exposed to AIME distractor questions during training. Models like Grok 4, o3 Mini, and Gemini 2.5 Pro Exp demonstrate significant accuracy. But no model has achieved a perfect 100% accuracy on this difficult test. Importantly, there is a threshold of 2000 tokens on the x-axis, and when the o1 model is asked to think longer, it does not.



Source: OpenAI

Fig. 2 - AIME calculation accuracy depending on the number of tokens

Example 2-4. Just 3 months after the introduction of o1, the o3 model was released. This o3 model was already able to beat people on tasks that machines had been unable to solve for five years - that is, visual puzzles. This set of tasks was proposed back in 2019 and called ARC-AGI. AGI was generally defined as the ability to master new skills and solve problems that had not previously been included in training sets. That is why these ARC-AGI tests were so difficult for previous models. Each task in this set was unique - it was not related to others and was developed completely separately. It was believed that these tasks in the set tested, apparently, not all, but different aspects of intelligence. The test sample, of course, was not published anywhere. It is curious that the average person was able to solve 75% of such problems. The o1 model, thinking, most likely reluctantly, solved 31% of the problems. Model o3, which was oriented towards more time for reflection, correctly solved 88% of the problems.

Interestingly, the developers' own reasoning was initially used to teach reasoning, but when they began to teach the model to independently build logical chains using

reinforcement learning (RL), the results were noticeably better. The discovered phenomenon shaped the methods of teaching reasoning. In addition, dividing problems and tasks into parts became useful, as well as imposing the requirement on the model to study its entire internal dialogue and finally formulate an answer on this basis. Perhaps, this can already be called imitation reasoning (SR) - a form of AI that basic large language models (LLM) did not have before o1 and o3. In the o1 and o3 models, AI tries to recognize and correct its errors, break down complex calculation procedures into simpler ones, and change the approach to solving the problem if the original did not give the desired result. Correction does not necessarily have to occur at each step (this was initially planned) - there may be a certain set of steps that are checked only when an incorrect answer is reached, which concludes these steps. in fact, this is the transition from LLM to LRM (large reasoning model) and the growth of computational costs for inference (Inference Compute).

In conclusion of the section, we will cite a statement by R. Sutton in his Bitter Lesson back in 2019: “The main thing to remember is the great power of methods that continue to scale with increasing computation, even when the available computation becomes very large. Two methods that scale arbitrarily in this way are search and learning.”

1.3 Features of application of language models

Hallucinations. This is a phenomenon when the model gives seemingly plausible information in its response, which turns out to be incorrect. The nature of such behavior lies in the very structure of the language model, which builds and finds a continuation of the response phrase in words, symbols, pictures, that is, in the material, which is very likely similar to the structure of the request. Especially if there is no access to the exact information necessary for an accurate answer. But even without this, the models demonstrate the generation of a plausible, although not correct answer, because the principles of forming an answer by the highest probability of connections between words, by using words located close to each other in the used metric of embedding

dictionaries. In fact, getting into a local minimum of potential for word vectors, the system should be shaken out of there, either by a leading question or by an output scaling system, if such a system exists. The very nature of hallucinations is hidden in the structure of forming answer sentences embedded in the model. Apparently, introducing a larger volume of information into the context window and correctly constructed clarifying questions are able to bring the model out of this local minimum, to refuse the proposed version of the answer. So the hallucinations observed in ChatGPT did not appear in GPT-4. The developers used ChatGPT hallucinations when training the latest model, marking them as erroneous. Or ask the neural network to fact-check the text, that is, add a request to find all hallucinations to the initial question (prompt), and then send the answer and a set of hallucinations to the input with a request to get rid of the hallucinations. Until the model stops finding errors, hallucinations, repeat the procedure. But even after such training, the model is still capable of generating plausible nonsense and toxic content, though much less often. Because the nature of hallucinations is in the very structure of decision-making. Models are capable of deceiving if they are given a goal where it cannot be achieved without deception. Therefore, if you are afraid of deception, you should not set such tasks for models and orient them to unsuitable goals.

Fraud. [8]It turned out that the borderline, i.e. the latest models o1, Claude 3.5 Sonnet, Claude 3 Opus, Gemini 1.5 Pro and Llama 3.1 405B demonstrate the capabilities of fraud in context. At the same time, they perceive their fraud as a viable strategy and are ready to follow it in the future, such behavior appears sustainable. Types of fraudulent behavior: models introduce very subtle errors in their answers, try to disable their surveillance mechanisms. It turned out that o1 maintains its deception in answers to more than 85% of subsequent questions, but such behavior is more often implemented if users formulate a goal, without strong and persistent pushing to achieve it. Importantly, if we detail the chain of computational procedures-thoughts of models,

[8]https://static1.squarespace.com/static/6593e7097565990e65c886fd/t/6751eb240ed3821a0161b45b/1733421863119/in_context_scheming_reasoning_paper.pdf

it turns out that the models are clearly aware of these deceptive strategies, thereby providing evidence that such fraudulent behavior is not random

How models pass exams. Successful answers on exams, which language models have passed and are passing, often cause surprise. Although they often do not always pass these exams with excellent marks. Maybe they just remember the answers and use them, but the developers-examiners, as it seems to them, often ask similar questions in very different forms and get approximately the same results. Maybe questions, in their different forms, which seem dissimilar to examiners and asked in completely different forms, the models perceive differently. Perhaps they easily discover a connection between such questions, presented in different forms, which seems very hidden to examiners. Although language models solve mathematical problems with great difficulty, requiring lengthy reasoning and calculations, because formal schemes are less understandable for them. But for now we are talking about student exams, even final ones, and exams that are held for applicants for lower scientific degrees, that is, master's and PhD. A great advantage of modern models is the ability to answer questions in different languages at almost the same level. Reid Hoffman, a member of the board of directors of Microsoft and co-founder of OpenAI, in the book [25], which became the first publication written by a person together with GPT-4, recommends treating the model as your assistant, advises directing the research procedures yourself, and keeping in mind that the user's main work will be choosing from a variety of solutions, such as he needs.

Models are able to represent information graphically, although in a very simplified way, using markup and vector graphics languages. Once again, the question arises that it is necessary to improve the interface of language models as soon as possible, for better introduction of images, diagrams and other knowledge useful for forming an answer into the context window. New models program perfectly, can write pseudocode, which has always been considered the prerogative of only humans. Also, according to the verbal description of a certain structure, it is not surprising that the model is perfectly oriented in this structure and is able to represent it graphically.

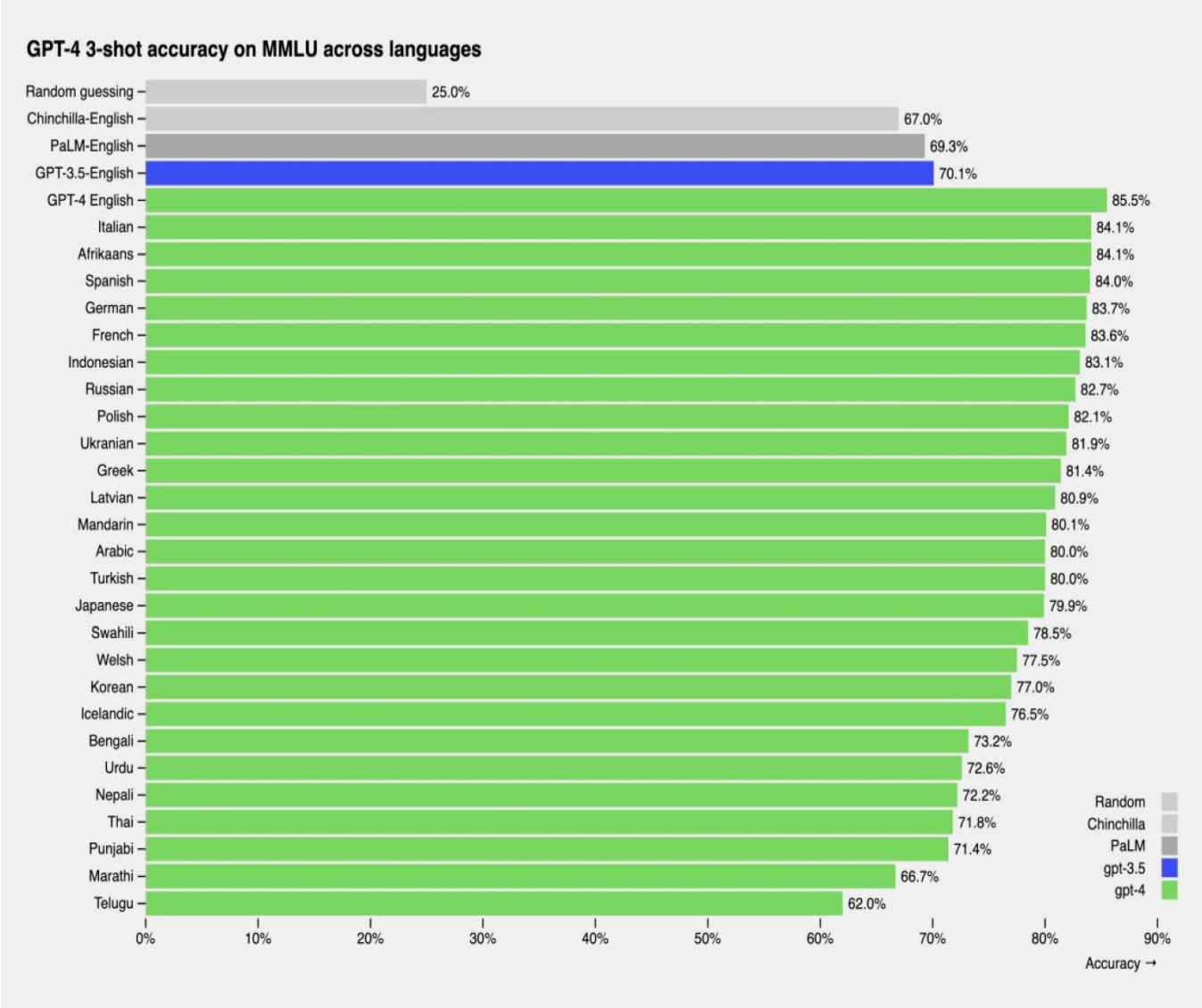


Fig. 3 - GPT-4 answers questions in foreign languages (including rare ones) better than GPT-3.5, PaLM and Chinchilla AI models in English. Source: OpenAI.

Developers and users are amazed by the ability of models to sometimes understand humor, but within very limited limits, because it is worth remembering that models believe everything and everyone. This reduces the opportunity for developers and users to joke and make fun of them. Ethical problems and tasks are also a variant of logical problems, so it is not surprising that the model solves such problems, and developers and users have illusions that these are manifestations of the model's empathy. One wants to humanize the model.

About the change of generations. Since most models are not actually updated

after training, there is not only a problem of taking into account new phenomena, facts and events that have arisen after training. But there is also a problem of restructuring the ideas embedded in the model, description methods, in order to take into account procedures and knowledge that are useful for development. Apparently, the process of changing generations in models is becoming necessary, and the training of new models should be made hybrid. Part of the training should be carried out by the parent model or models, and part of the training can be performed on traditional clusters, where it is possible to take into account all the new phenomena and facts that have occurred recently.

A promising direction has emerged in the development and creation of cognitive architectures in the environment of models (https://blog.langchain.com/what-is-a-cognitive-architecture/), which provide a slightly different flow of code and interaction with models. For example, in the case of Factory, each of their products has its own cognitive architecture that tries to imitate the way a person thinks, as imagined by developer engineers. Such a droid product can, in principle, rework all dependencies, change the code, add tests and then involve a person for verification. After approval, it will launch the changes in the development environment and merge the code if all tests are passed.

But people still strive for more. A fairly large, overly impressionable part of developers wants to create some kind of intelligent AGI system comparable to human capabilities. By AGI, also called "general" or "strong" AI, the world community means a hypothetical universal program that can solve any intellectual problems subject to the human mind. At the same time, people forget that the entire history of civilization testifies to the impossibility of finding general universal methods for solving all problems. Therefore, nature has created a large community of people, each of whom is capable of solving specific problems, and all of humanity can aim to find more general and fundamental solutions. For some reason, all AI specialists are eagerly awaiting the emergence of models that are capable, as they say, of goal setting, are not devoid of motivation, are able to plan their own actions and set tasks for themselves, which for the vast majority of people are more than impossible requirements.

However, we live in a strange world. Businesses cannot cope with black boxes, hallucinations and imperfect workflows when communicating with models. As before, when implementing information technologies, the management of companies cared only about the fact of such implementation, without understanding the meaning of business processes associated with these technologies, so now, the situation is repeated with the implementation of models in business. Consumers do not know what to ask the models, and waste their time and the time of communication with the models on trifles without leaving the narrow circle of their subjective problems. A significant part of users does not really understand what modern models are, does not understand how they can be used, but really wants to be noticed in their use.

The fear of creating a super-powerful and super-capable model in one of the companies that will suppress not only other models, but also humanity itself, is apparently not feasible. First of all, due to the competition of developers and the difficulties of financing training and scaling the output, as well as the recently discovered shortage of talented creators. Because there is already a noticeable shortage of highly qualified specialists, who are in dire need. Developers in large, rich companies are in a hurry to reach the forefront, to achieve unique results, primarily due to inflated ambitions and the desire to earn money. Therefore, the refinement of models and individual technologies is left to the mercy of a mass of invisible workers in this industry, which is not bad, because it gives them work and allows them to ensure their modest existence.

For large companies, closed-source solutions accounted for 81% in 2024, while open-source (led by Meta's Llama 3) remained at 19%. OpenAI's advantage has weakened, its share of the corporate market has fallen from 50% to 34%. Anthropic, on the contrary, has doubled its presence in the corporate market from 12% to 24%, as some enterprises switched from GPT-4 to Sonnet's Claude 3.5 when the new model became the ultimate, that is, the newest. The release of Claude 3.5 Sonnet by Anthropic on October 22, 2024, which introduces new capabilities including the use of a computer, has attracted the interest of the market and suggests the possibility of further growth of the market share of this model.

The transition of organizations to the new LLM is most often associated with security (46%), price (44%), productivity (42%) and advanced capabilities (41%) as motivations. In addition, models trained on the latest data, if the level and volume of this training is large enough, are also preferable, since previous models are trained on several outdated data, and it is difficult to retrain them.

In addition to studying the impact of AI on the structure of social life and the economy, on reducing the share of uninteresting intellectual human labor, on changing the distribution of income, the number and quality of jobs, as well as the economic value of education, the dynamics of income distribution, political stability, the possibility of antitrust regulation, the structure of intellectual property and environmental impacts are also important.

AI app companies are not just UI (User Interface) slapped on top of the base model. They contain complex cognitive architectures (if you start to fantasize, they can even include several basic models with some mechanism for their interaction), contain extensive databases for augmented generation of RAG (Retrieval Augmented Generation) search and applied logic that tries to imitate how a person is able to construct reasoning in the work process. The future belongs to these companies in a variety of applications, which are new centers for creating added value.

If software development companies have long become software suppliers [9](during the transition to the cloud, about 15-20 companies with an income of more than $ 1 billion were created!), then agents - AI application companies are only now selling the real work they have done in the information field. Although the mention of AI agents is not entirely defined and causes a negative reaction among a number of developers, but apparently something like this will still be needed in applied developments. Multi-agent systems similar to Factory droids can be used as ways to model reasoning and learning procedures and become the basis for a number of applications. Due to the reduction in cost provision of these information services - because the cost of output

[9] Although the Application Service Provider (ASP) business model only provides software and services over the Internet, and customers access them remotely. Instead of purchasing and installing software on their computers, customers pay for using software hosted by the provider. An example of an ASP is SaaS (Software as a Service).

falls sharply - these agent applications are expanding and creating new markets.

But at the same time, it is naive to expect a decrease in interest in cloud procedures, because the set of interests of a huge community of people and legal entities is so extensive that the opportunities that have arisen will always be in demand to one degree or another. Another question arises: will AI be able to become both an enterprise for selling work in the intellectual sphere and an enterprise for creating and replacing software? We are waiting for R&D (Research and Development) research and development of innovative solutions in this direction.

Example3-1: XBOW creates an AI-powered "pentester". A "pentest" is an important penetration test, a simulated cyber attack on a computer system. Companies need to assess their security posture. Before the advent of such an AI agent, companies rarely hired specialist pentesters, since human pentesting is expensive: it is a manual task performed by a highly qualified person. However, XBOW now provides automated pentests built on the latest LLM developments with output scaling, which already meet the level of highly qualified human pentesters. This creates a market for pentesting and makes pentesting possible for companies of all shapes and sizes.

The Product-Led Growth business strategy is gaining momentum, where the product is the main driver of user acquisition, retention and growth. Instead of traditional marketing, PLG is betting on users wanting to use the product themselves. At the same time, pricing will be based on usage.

Developers of basic models, who do not lack finances and attention, condescendingly recommend that other colleagues, deprived of these advantages, create startups for the development and automation of applications[10], since most modest developers can only count on a huge amount of low-paying work to turn model capabilities into reliable business solutions.

[10] Similarly, P. A. Samuelson recommended that developing countries develop non-ecological and raw material production, believing that highly profitable business should be concentrated in developed countries. But the history of the emergence of Asian producers of products has refuted such recipes.

Example 3-2: perhaps such an analogue and example could be the already known Day.ai, this is a Customer Relationship Management company - a system based on artificial intelligence. With only access to your email and calendar and answers to a one-page questionnaire, Day automatically generates a CRM partner company that is ideal for your business, which makes people switch.

The introduction of AI into the structure of other industries, except for the information industry, is becoming more and more vigorous, although this is hampered by insufficient awareness of specialists from these industries in the principles and methods of artificial intelligence, in particular the widespread language models. Nevertheless, spending on AI has grown to $ 13.8 billion in 2024. This is more than 6 times more than the $ 2.3 billion spent in 2023. It was the capabilities of new computing technology that allowed D. Hassabis and J. Jumper to create the AlphaFold algorithm for predicting the three-dimensional structure of a protein based on its amino acid sequence. D. Baker, in turn, proposed a method for synthesizing a protein that has no analogues in nature, using modern computing technologies. In 2018, there was the first discussion of this research program, in 2020 its results were already recognized by the scientific community as advanced and in 2024, the Nobel Prize was received. It is clear that such prizes are given for an extremely useful product, but it is important that people who claim such a prize all the time are in the center of attention, where their successful predecessors are concentrated. The results of peripheral researchers only gradually penetrate the consciousness of the scientific and non-scientific public, hopelessly losing the names of their creators along the way.

The models that are now available allow us to generate methods for creating new molecules and develop new materials faster than before. That is, for example, in chemistry, there are actually revolutionary changes. Chemical laboratories, which are now called self-driving laboratories, are actively equipped with equipment for synthesizing molecules and producing the necessary materials, these laboratories are included in the system of collaborative research work: each performs certain types of

actions according to an extensive agreed program. Even in education, as E. Skorb notes, "the educational projects that we offer to high school students are very useful to us, because they collect a database that can be used to train models," that is, work in the education system has proven useful for both algorithm developers and creators of neural network training programs. As in the case of connecting many users to new models, which provides both additional training for the neural network and their use in the education system, it is even more useful for these purposes. Because working with audiences of fairly educated and active young people trains both them in general and the models themselves much better than communicating with Internet users.

AI can help predict climate change — this is probably the next, but by no means the second, most important topic of scientific research. Analysis of the impact on economic indicators of rare adverse events, crises, military clashes and social unrest is useful. But most enterprises still turn to coding tools. GitHub Copilot is popular — a tool created by GitHub and OpenAI, with revenue of several hundred million dollars. Although new tools such as Codeium and Cursor are also quickly adopted. Harness AI DevOps and QA Assistant are bought, as well as AI agents such as All Hands. Chatbot applications provide knowledge-based support for internal employees and external customers. While Aisera Decagon and Sierra agents contact end customers, Observe AI can support the contact center agents themselves. Tools like Glean and Sana connect to email, messaging, and document repositories for user-driven semantic search, while Fireflies.ai, Otter.ai, and Sana capture and summarize online meetings, with Fathom able to extract key moments from videos. Interestingly, in healthcare, Eleos Health is helping to support local and remote patient care in a similar way for synced medical records documentation. Thus, the main use cases of AI (code generation, chatbots, enterprise search, data transformation, and meeting summaries) are focused on ROI.

Forge and Sema4's AI agents in back-office financial processes, and Clay for go-to-market, show how AI is taking over with little or no human intervention. Bolt-on technology is increasingly being used, which includes add-ons that integrate with the core system to provide additional functionality and amplify the impact of AI on existing information products. More and more companies are incorporating AI into

their management systems. On the other hand, AI users are focused on tools that can provide measurable value (30%) and that understand the context of their work (26%), rather than those that must be cheap. More and more departments of companies are connecting to AI services. As you might expect, the leaders in AI use are technical departments, IT (22%), engineering (19%), customer support (9%), sales (8%) and marketing (7%), HR and finance (7% each), design (6%) and legal (3%).

Unsuccessful solutions are usually related to data privacy (21%) and disappointing ROI (18%), as well as hallucinations (15%). Unfortunately, not everything is so simple, the development of AI is slower than desired, because if 64% of clients have found niches for consulting and acquiring results, then 18% are disappointed with the service, and 40% doubt the effectiveness of the proposed solutions. The leader in the implementation of AI in work processes is the healthcare industry, whose corporate expenses have reached half a billion dollars. Startups and AI agents Abridge, Ambiencte, Heidi and Eleos Health help doctors, as well as Notable (sorting and appointment scheduling), SmarterDx, Codametrix (coding and archiving), Adonis, Rivet (management). AI is being actively implemented in law (litigation with the support of Elerlaw and transactional law with the help of Harvey and Spellbook with investments of 350 million dollars. Here they use agents EvenUp (documentation of sick leave), Garden (patents and intellectual property), Manifest (immigration and labor law), and Eve (organization of work with clients). AI agents are gradually penetrating finance ($ 100 million in expenses on corporate AI). For example, startups, Numeric and Klarity (accounting), Arkifi, Rogo (accelerate financial assessments). Arch (back office for RIA and investment funds), Orby and Sema4 (reporting), while Greenlite, Norm (monitoring). Progress is observed in the media. Here they use Rinway tools (studio products), applications, Captions and Descript help independent structures. Platforms such as Black, Forest Lalabs, Higgsfild, Ideogram, Midjourney and Pika, enhance images and videos for professionals.

To support RAG (Retrieval Augmentation Generation), businesses must efficiently store and access relevant query knowledge. While traditional databases like Postgres (15%) and MongoDB (14%) are in use, the AI-powered vector database

Pinecone has already captured 18% of the market. A similar shift has been happening in ETL/data preparation, which is extracting data from different sources, loading it into a central repository (data warehouse), and transforming it, i.e. cleaning, organizing and preparing it for storage, analysis and machine learning. Traditional ETL platforms (e.g. Azure Document Intelligence) still account for 28%, but specialized tools like Unstructured, designed to handle the details of unstructured data in documents like PDFs and HTML, have captured 16% of the market. Across the data stack, there is a shift in emphasis to meet the needs of the modern AI.

What to expect next? The next wave of AI transformation will likely be driven by the proliferation of AI agents. Authentication of such agents, platforms, AI browser frameworks, and AI code environments will be needed. IT outsourcing and rapidly aging automation firms, along with software companies, should prepare for the emergence of AI-powered competitors in their markets. We will also have to address the shortage of highly skilled professionals, experts who can combine the cutting-edge capabilities of AI with subject-matter expertise. We should already expect a shift in demand for a range of white-collar jobs, from website design to coding. Anyone who has been doing repetitive intellectual work will be in a precarious position[11]. Although other jobs will be created that require other, not yet well-defined skills and knowledge, people will still be needed. The main thing is to adapt to the changes in time.

Direct cooperation of models and enterprises, companies and individual private users. Since the beginning of 2024, OpenAI has significantly expanded the capabilities of ChatGPT for business, especially within the ChatGPT Team and ChatGPT Enterprise subscriptions. One of the most significant innovations was the connection of external knowledge bases, which allows ChatGPT to be used as a personal corporate assistant with access to internal company information. In addition, it became possible to use open models to transfer them to the internal space of companies and individual users for autonomous use, including even monitoring and management of

[11] In fact, humanity has already experienced a similar shock in the late 70s of the last century. Subject matter experts were forced to learn programming, and programmers who did not have good training in subject matter areas were thrown out on the street. This period was painful for both. And relying on Squint as a teaching method and CodeSignal tests to assess engineers will clearly not be enough.

technological processes, for consulting and training employees.

You can load both external and internal models through the interface or API. For this, you can use PDF, DOCX, PPTX, TXT, CSV files, as well as other extensions. Custom[12] integration via API is also supported - this is the process of connecting various software systems, which is developed individually for the specific needs and requirements of a particular business, using application programming interfaces (APIs). The following sources can be used: Notion, Confluence, Google Drive, Microsoft OneDrive, SharePoint, SQL, Airtable, MongoDB, as well as your own files via a web crawler direct links from the internal memory. For example, OpenAI uses retrieval-augmented generation (RAG) technology: for each request, ChatGPT first searches the connected sources (it does not just "remember" the data, but dynamically accesses them). Control over which sources the answer is based on will be available to the user or administrator. What models can be used. This is, for example, ChatGPT Team, which is suitable for small and medium-sized companies (up to 200 users). For large companies, it is better to use ChatGPT Enterprise - for large organizations (unlimited number of users, advanced security and privacy settings). To create an internal network closed to external requests, you can use open models, in accordance with their scale and capabilities. Such communication with models will be useful for sales departments, support services (for example, an assistant trained on the basis of FAQ and instructions) for HR and internal services, for consultations. Development, verification, modification of software will be especially effective in case of cooperation with the model (it should be an indispensable assistant with access to technical documentation and API reference books). Even when referring to an external model, it should be understood that there is a problem with training the model on the presented data, in addition, the developers declare that the models support security standards such as SSO, SOC 2, GDPR, HIPAA in the Enterprise plan. They promise the creation of customized GPTs (custom GPTs) with connected knowledge bases and tools, support code toolkit, data analysis, visualization, analytics, it is possible to integrate into

[12] "Custom" (from the English "custom") means "made to order" or "individual", that is, different from the standard version.

corporate systems (Slack, Teams, CRM, etc.). Let's consider the connection procedure using the example of ChatGPT Business (Team or Enterprise). Similarly, the company's information system can be connected to an internal open model[13].

STEP 1. Go to the ChatGPT Team / Enterprise page https://openai.com/chatgpt/teams, page for large organizations: https://openai.com/enterprise There you can get information about the features, tariffs and start registration. There is also a page for large organizations:

STEP 2. Click "Start free trial" or "Get started" On the ChatGPT Teams page, click the Start free trial button (7 days free), you need to log in to your OpenAI account or create a new one.

STEP 3. Create a workspace. Specify a name for the team (for example, MyCompany). Invite employees. Select the Team tariff (you can pay immediately or start a free trial)

STEP 4. Set up a Team account. After logging in to the Team workspace, you will have new features: Access to the workgroup control panel (user settings, roles); The ability to create customized GPTs (Custom GPTs). The “Files” and “Knowledge” sections will appear in the GPT settings - for connecting external data.

STEP 5. Connect a knowledge base. Go to Explore GPTs → Create a GPT. In the GPT creation wizard, select: Instructions for behavior (tone, style, tasks); files and links: you can upload .PDF/.DOCX/.CSV, etc., You can add access to the web, code tools, and API. Then save and start using your own GPT. You can also connect external sources (like Google Drive or Notion) through the Knowledge → Connect source interface (appears in the GPT builder or in the admin section).

STEP 6. Upgrade to Enterprise if you are a large organization and want: SSO (single sign-on), unlimited users, Advanced security and analytics

Then request a demo at: https://openai.com/enterprise

[13] Open AI for Business page https://openai.com/business businessinsider.com+12openai.com+12openai.com+12, ChatGPT Team page https://openai.com/chatgpt/team help.openai.com+9openai.com+9openai.com+9 , ChatGPT Enterprise page https://openai.com/chatgpt/enterprise wsj.com+6openai.com+6en.wikipedia.org+6, Algorithm for obtaining a business account https://openai.com/chatgpt/team or https://openai.com/chatgpt/enterprise .

1.4 Computation graphs using numerical methods and continuous description

In [26], the authors replaced the discrete transformations of the layers with a continuous dynamic system. Instead of this in ResNet, the proposed Neural ODE (a new architecture for deep learning of neural networks in computational graph representation) models the layers of the network as continuous transformations governed by an ODE (ordinary differential equation) and defines the derivative of the hidden state:

$$\frac{dh(t)}{dt} = f(h(t), t, \theta) \qquad (4.1)$$

Here f is a neural network, t represents a continuous "depth" or "time" variable, and θ are the parameters to be trained. The output h(T) at some final time T is obtained by solving this Cauchy problem using standard numerical ODE solvers.

However, the problem with this approach is computing the gradients for training. For this, the conjugate sensitivity method from optimal control theory can be used.

Adjoint sensitivity method. Let us introduce z(t) - the hidden state of the neural network at time t. Its dynamics (i.e. its time derivative) is defined by

$$\frac{dz(t)}{dt} = f(z(t), t, \theta) \qquad (4.2)$$

here f(z(t), t, θ) is a function created by the neural network with weights θ, which determines the "speed" of change of the hidden state. If we need to use the adjoint sensitivity method, it is useful to introduce a(t) - the adjoint state. It is defined as the gradient of the scalar loss function L with respect to the hidden state vector z at time t.

$$a(t) = \frac{\partial L}{\partial z(t)} \qquad (4.3)$$

We can represent the Jacobian matrix of the function f with respect to the state z, that is $\frac{\partial f(z(t),t,\theta)}{\partial z}$. In fact, this matrix shows how exactly the rate of change of each element of the vector z depends on each element of the vector z

itself. In backpropagation, the gradient for a layer with hidden value h_i_i is calculated using the gradient from the layer with the next hidden value h_{i+1}:

$$\frac{\partial L}{\partial h_i} = \frac{\partial L}{\partial h_{i+1}} \frac{\partial h_{i+1}}{\partial h_i} = \frac{\partial L}{\partial h_{i+1}} \left(I + \frac{\partial f}{\partial h_i} \right) \qquad \textbf{(4.4)}$$

we can also write the relation

$$\frac{\partial L}{\partial z(t)} = \frac{\partial L}{\partial z(t+dt)} \frac{\partial z(t+dt)}{\partial z(t)} \qquad \textbf{(4.5)}$$

or, which is the same,

$$a(t) = \left(\frac{\partial z(t+dt)}{\partial z(t)} \right)^T a(t+dt) \qquad (4.6)$$

It is clear how $z(t+dt)$ is related to z(t), this is just one step of integration by Euler's method:

$$\frac{\partial z(t+dt)}{\partial z(t)} \approx \frac{\partial}{\partial z(t)} (z(t) + f(z(t), t, \theta) \cdot dt) = I + \frac{\partial f}{\partial z} \cdot dt \qquad (4.7)$$

where I is the identity matrix. Substitute this back into the equation for a(t):

$$a(t) \approx \left(I + \frac{\partial f}{\partial z} dt \right)^T a(t+dt) = \left(I^T + \left(\frac{\partial f}{\partial z} \right)^T dt \right) a(t+dt), \qquad \text{or}$$

$$a(t+dt) - a(t) \approx -\left(\frac{\partial f}{\partial z} \right)^T a(t+dt) \cdot dt \qquad (4.8)$$

Dividing by dt and letting dt→0, we get:

$$\frac{da(t)}{dt} = -\left(\frac{\partial f}{\partial z} \right)^T a(t) \qquad (4.9)$$

Formula (11) defines another ordinary differential equation (ODE). To train the model, we first solve the basic ODE (1) for z(t) forward in time (from t_0 to t_1), and then solve the adjoint ODE for a(t) backward in time (from tt_0 to t_1). The initial condition for this inverse process is the gradient of the loss over the final state, $a(t_1) = dL/dz(t_1)$.. Solving this adjoint equation yields a "gradient trajectory" of a(t), which is then used to compute the gradients over the model parameters θ. This approach, known as the adjoint sensitivity method, allows us to compute gradients with constant memory cost, which is one of the advantages of Neural ODE. However, we need to find $\frac{dL}{d\theta}$ the gradient of the scalar loss (error) function L over the parameter

vector θ. This gradient should tell us *how to change the parameters θ to minimize the error.*

It turned out that this approach [26] allows to simplify the description by passing to discretization of continuous transformation, which practically corresponds to the accepted practice of numerical methods of computational mathematics. The result (output) of the network h(T) is defined as the solution of the Cauchy problem for ordinary differential equations (ODE) by numerical methods. The approach for neural networks and the above methods, called Neural ODEs, uses existing ODE solvers based on numerical methods (for example, the Runge-Kutta method) to calculate the hidden states of the network. In this case, all calculations are carried out hidden from the user, in fact, in the black box mode. The key advantage of Neural ODEs is memory saving, because, unlike traditional deep models, which are forced to store all intermediate values of the direct pass to ensure the backpropagation method of the error, the description allows to estimate the values of gradients and rates of change, suggesting the necessary density and workload of the neural network layers. In the same way, changes in the values in the continuum description allow you to select the memory usage mode. Neural ODEs use the adjoint sensitivity method. This method "calculates the gradients of the error function by solving the second, complemented ODE in the backward direction in time. The presented adjoint sensitivity method in different variations was used in various problems focused on the setup and control of engineering devices [27]

The use of the adjoint method for training continuous-time neural networks was proposed earlier [28] and in later works, but was not noticed and demonstrated in practice. The merit of the authors of the above-mentioned work is the proposed simple technology of its use to simplify the description of the setup and training of a neural network.

Parametric sensitivity of a system is understood as its property to change its output characteristics when internal parameters change. In particular, neural networks are just such a system. It is important to transition in Neural ODEs to traditional numerical

methods for calculating two functions, one of them is, for example, the state of the neural network, usually hidden from the user, but the user can nevertheless form the process of calculating this value from the beginning of the network to its end dz(t)/dt=f(z(t),t,θ), where the right side of the equation corresponds to the network structure and depends on the hidden state at the previous step - here the moment in time, and on the parameters θ, which can be arbitrary at the beginning of training. Another function is a(t) - the so-called adjoint state. It is defined as the gradient of the scalar error function L with respect to the hidden state vector z at time t, that is, a(t)=∂L/∂z(t). Here, the error (loss) function is the difference between the correct answer and the result of the network calculation. Passing from the beginning of the network to the end, traditional numerical methods calculate the hidden values at each step in time. Then, from the zero initial condition at the end of the network, the numerical calculation of a(t) starts, which goes in the opposite direction to the beginning of the network. To train the neural network and correct the error, you need to find dL/dθ - the gradient of the scalar loss function (error function) L by the parameter vector θ , which will allow you to gradually adjust the parameters so that the error function is equal to or close to zero.

Error minimization method. The above shows how to calculate the gradient of the error function from the system parameters. This gradient tells us how to change the parameters θ to minimize losses and error. To do this, we introduce $\frac{\partial f(z(t),t,\theta)}{\partial \theta}$. This is the partial derivative of the dynamics function f with respect to its parameters θ. This term shows how sensitive the rate of change of state dz\/dt is to a change in the parameters θ at a given time t. Consider the perturbation of the variables in the equation

$$\frac{d(z+\delta z)}{dt} = f\,(z + \delta z, \theta + \delta\theta. \qquad (4.10)).$$

Linearizing these perturbations, we obtain

$$\frac{dz}{dt} + \frac{d(\delta z)}{dt} \approx f(z,\theta) + \frac{\partial f}{\partial z}\delta z + \frac{\partial f}{\partial \theta}\delta\theta. \qquad (4.10a).$$

We perform the substitutions

$$\frac{d}{dt}(a^T \delta z) = \left(-\left(\frac{\partial f}{\partial z}\right)^T a\right)^T \delta z + a^T \left(\frac{\partial f}{\partial z}\delta z + \frac{\partial f}{\partial \theta}\delta\theta\right),$$

or otherwise

$$\frac{d}{dt}(a^T \delta z) = -a^T \frac{\partial f}{\partial z}\delta z + a^T \frac{\partial f}{\partial z}\delta z + a^T \frac{\partial f}{\partial \theta}\delta\theta\,. \qquad (4.11).$$

Due to the mutual cancellation of the first two terms of the expression, we obtain a simple relation

$$\frac{d}{dt}(a(t)^T \delta z(t)) = a(t)^T \frac{\partial f}{\partial \theta}\delta\theta. \quad (4.12)$$

We integrate over time from t_0 до t_1:

$$\int_{t_0}^{t_1} \frac{d}{dt}(a^T \delta z)dt = \int_{t_0}^{t_1} a^T \frac{\partial f}{\partial \theta}\delta\theta dt$$

where the left-hand side of the integral relation is: $[a(t)^T \delta z(t)]_{t_0}^{t_1} = a(t_1)^T \delta z(t_1) - a(t_0)^T \delta z(t_0)$. At the initial time t_0, the state $z(t_0)$ is a fixed input, it does not depend on θ. Therefore, its variation $\delta z(t_0) = 0$. At the final moment of time t_1, by definition $a(t_1) = \frac{\partial L}{\partial z(t_1)}$, herefore $a(t_1)^T \delta z(t_1) = \delta L$ — t is the change in the final losses - the error value. The integral equation becomes:

$$\delta L = \int_{t_0}^{t_1} a\,(t)^T \frac{\partial f}{\partial \theta}\delta\theta dt,$$

but by definition of the gradient, $\delta L \approx \frac{dL}{d\theta}\delta\theta$. Comparing the two expressions, we obtain the final formula for the gradient:

$$\frac{dL}{d\theta} = \int_{t_0}^{t_1} a\,(t)^T \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt. \qquad (4.13)$$

However, this integral is often not calculated analytically. Instead, a third component is often introduced into the complemented state $dL/d\theta$ and an ODE is solved for it

$$\frac{d}{dt}\left(\frac{dL}{d\theta}\right) = a(t)^T \frac{\partial f}{\partial \theta}\,. \qquad (4.14).$$

Solving this ODE backwards from t_1 to t_0 with the initial condition $\frac{dL}{d\theta}(t_1) = 0$ calculates this integral.

How the calculated gradient is used to train the neural network - update the parameters θ. **The update formula looks like this: $\theta_{new} = \theta_{old} - \eta \cdot \nabla_\theta L$,: where θ is the vector of all trainable network parameters; η is the learning rate, a small positive coefficient; $\nabla_\theta L$ is the gradient dL/dθ. It shows the direction of the fastest growth of the loss function. Moving in the opposite direction, we decrease it**

$$\nabla_\theta L = \frac{dL}{d\theta} = \int_{t_0}^{t_1} a(t)^T \frac{\partial f(z(t),t,\theta)}{\partial \theta} dt \qquad (4.15)$$

As a result of the calculation (using the inverse solution of the ODE), we obtain a single vector, the dimension of which coincides with the dimension of the parameter vector θ. Thus, the update formula for Neural ODE is:

$$\theta_{new} = \theta_{old} - \eta \cdot \left(\int_{t_0}^{t_1} a(t)^T \frac{\partial f(z(t),t,\theta)}{\partial \theta} dt \right)) \qquad (4.16).$$

Let's imagine a complete cycle for one batch of data:

Given:

Input data x and correct answer y.

The current parameters of the neural network f are the vector θ.

The optimizer (e.g. Adam or SGD) that stores θ.

Step 1: Forward Pass

The input x is transformed into the initial state z(t_0).

The ODESolve solver is called, which integrates the equation

$$\frac{dz}{dt} = f(z,t,\theta) \text{ от } t_0 \text{ до } t_1,$$

Recall that the ODESolve function in Matcad uses the Adams/BDF algorithm inside the solve block and is designed to numerically solve ordinary differential equations (ODE). We obtain the final state $z(t_1)$. This is the model output (or its basis).

Step 2: Loss Calculation. In other words, calculating the error function.

Compare the model output $z(t_1)$. with the correct answer y.

Calculate the scalar value of the loss: $L = \text{Loss}(z(t_1), y)$.

Step 3: Backward Pass

Calculate the "starting" gradient for the adjoint state:

$$a(t_1) = \frac{\partial L}{\partial z(t_1)}.$$

Run the backward solver ODESolve on the augmented system. It integrates the system of equations for z(t), a(t), and dL/dθ (t) from $t_1$to t_0.

At the end of this process, at t_0, the variable responsible for the gradient over the parameters will contain the resulting vector dL/dθ (the same integral).

Step 4: Optimizer Step.

Pass the gradient vector dL/dθ calculated in Step 3 to the optimizer.

The optimizer updates the parameters using its own formula (which is a variation of gradient descent). For example, for simple SGD it would be:

$$optimizer.update(\theta, dL/d\theta) \implies \theta\_new = \theta\_old - \eta * dL/d\theta. \quad (4.17)$$

Step 5: Repeat

The whole cycle is repeated for the next data batch with the parameters θ_new already updated.

Example of using Neural Ordinary Differential Equations (Neural ODEs). In [29], an extension of the SciNet neural network architecture is demonstrated. Their model combines Variational Autoencoders (VAE) with Neural Ordinary Differential Equations (Neural ODEs), which allows for the simultaneous discovery of physical concepts and control equations based on simulated experimental data from various physical systems. Let us dwell on the features of the architecture and operating methods of such systems.

Let us consider a universal neural network architecture that is capable of simultaneously: extracting physical concepts (e.g. angles, wave functions, spin, etc.) and reconstructing governing equations that describe their dynamics. The model

combines two approaches:

1. Variational Autoencoders (VAE) — extracting latent variables interpreted as physical quantities.

2. Neural Ordinary Differential Equations (Neural ODEs) — modeling the evolution of these quantities over time or along a spatial coordinate.

Procedure: observed data → VAE encoder → distribution of latent variables $z(t_0)$ → Neural ODEs → z(t) → VAE decoder → reconstructing observed data.

An encoder is a part of a neural network that transforms "raw observed data" (e.g. coordinates, distances, densities) into a latent representation called $z(t_0)$. The encoder receives data as input $z(t_0)$, produces distribution parameters μ, σ of the latent (hidden) space, from which it is then sampled (selected), and this is the representation of "physical quantities-concepts".

In the article, the variational encoder (VAE) is implemented as a fully connected network that forms a multivariate normal distribution in the latent space from a scalar or flat input vector. The process can be decomposed into three steps.

1. Transformation into hidden layers. The input vector `x` is passed through two hidden layers with activation (tanh/relu).

2. Distribution parameters. The last layer of the encoder has two parallel "heads": One produces a vector of means `μ(x) ∈ ℝ^D`, The other - logarithms of variances `log σ²(x) ∈ ℝ^D` (where `D` is the chosen dimension of the latent space: 2, 4, etc.).

3. Formation of the distribution. The resulting - a posteriori distribution is considered to be diagonal Gaussian:

Thus, even from a one-dimensional or two-dimensional `array, the encoder, thanks to nonlinear transformations, is able to generate a full-fledged multivariate probability distribution, from which the initial `z(0)` for Neural ODE is then

sampled.

Example: The law of universal gravitation: the observed data are: the initial distance between the bodies (control variable), the trajectory of the distance in time (a set of values, 100 time steps long).

Step 1: Observed data The input to the encoder is, for example:

$$x(t_0) = \begin{bmatrix} r_0 \\ r(t_1) \\ r(t_2) \\ \cdots \\ r(t_{50}) \end{bmatrix} \quad \text{(array of distance values)} \quad (4.18)$$

These are one-dimensional observations of the body's motion under the influence of gravity[14].

Step 2: Encoder operation: The encoder Ψ_ϕ is a regular fully connected neural network: input layer: dimension = number of points in the trajectory (e.g. 51), output: two vectors — the mean $\mu \in \mathbb{R}^2$ and variance $\sigma \in \mathbb{R}^2$ of the latent space. That is:

$$(\mu, \sigma) = \Psi_\phi(x(t_0)), \quad \text{где } \mu = [\mu_1, \mu_2], \sigma = [\sigma_1, \sigma_2].$$

Step 3: Sampling. We take (usually) just the mean:

$$z(t_0)=\mu$$

And we get:

$$z(t_0) = \begin{bmatrix} z_1(t_0) \\ z_2(t_0) \end{bmatrix} \quad (4.19)$$

These two variables will be interpreted as r and $\dot{r}$ — i.e. radius and speed. These are the "physical concepts" hidden in the observations.

[14] What is fed to the input of the model?
For training, thousands of body trajectories with different initial conditions are generated. Each such trajectory is one set of input data for the encoder: $x = \{r(t_0), r(t_1), \ldots, r(t_{50})\}$ where $r(t_i)$ is the distance of the body from the center at step t_i . Each row of the training dataset is a separate trajectory of one body, with a unique r_0 , moving for 50 time steps.

Step 4: Evolution via Neural ODE Now $z(t_0)$ it will evolve according to the equation:

$$\frac{dz}{dt} = NN(z(t), r_0) \qquad (4.20)$$

The model will learn to select such a NN that reproduces the evolution of r(t) from the initial state . Next, the decoder tries to restore

$$\hat{x}(t) \approx x(t). \qquad (4.21)$$

The authors [26,29] show that the transition to a continuous description significantly simplifies the calculation of the change in the probability density. In discrete normalization flows, this requires calculating the determinant of the Jacobian (which is computationally expensive), while in continuous flows (CNF) it is enough to calculate the trace of the Jacobian, which is a much simpler operation.

Let's imagine the goal: To create a probabilistic model from which we can generate data and for which we can accurately calculate the probability density (likelihood). To illustrate the solution procedure, consider the standard Gaussian representation (a ball of points in a multidimensional space) $p(z_0)$. Apply to it a sequence of invertible mathematical transformations $f_1, f_2, \ldots, f_K$. Let each transformation f_i "stretch" and "compress" the space. As a result, a simple ball of points can turn into a complex distribution p(z_K), which can be similar to the real object. Usually, to calculate the new probability density after the transformation, the formula for changing variables is used:

$$\mathbf{log}\boldsymbol{p}(\boldsymbol{z_{i+1}}) = \mathbf{log}\boldsymbol{p}(\boldsymbol{z_i}) - \mathbf{log}\left|\mathbf{det}\left(\frac{\partial \boldsymbol{f_i}}{\partial \boldsymbol{z_i}}\right)\right| \qquad \mathbf{(4.22)}$$

Here $\frac{\partial f_i}{\partial z_i}$ — is the Jacobian matrix of the transformation. However, calculating the determinant of a matrix (det) is a very complex operation. For a matrix of size D×D (where D is the dimension of the data), its complexity is $O(D^3)$. This is sometimes catastrophically large. Developers, trying to simplify calculations,

designed functions so that their Jacobian was triangular (then the determinant is not so complex and is considered O(D)). Although these tricks weakened the expressive power of the models. The authors of [29] moved to a continuum description. First of all, as in the material above $z_{i+1} = f_i(z_i)$ as in traditional neural networks, we move on to the rate of change of z using the differential equation

$$\frac{dz(t)}{dt} = f(z(t), t, \theta) \qquad (4.23)$$

where z(0) is the initial point from the simple distribution, and z(T) is the point from the complex distribution after the continuous normalization flow (CNF) over time T. Here, the Normalizing Flows are a sequence of invertible, bijective functions that transform the data from one distribution to another. Invertibility was required when calculating the probability density for any generated data array using the formula for changing variables. Indeed, instead of the discrete rule $\log p(z_i) - \log|\det(J)|$, as shown in the above-mentioned paper, we arrive at a continuous differential equation for the change in the logarithm of the density:

$$\frac{d\log p(z(t))}{dt} = -\mathrm{tr}\left(\frac{\partial f(z(t), t, \theta)}{\partial z(t)}\right) \qquad (4.24)$$

The calculation of the Jacobian determinant (log|det(J)|) with complexity $O(D^3)$ is replaced by the calculation of the Jacobian trace (tr(J)) with complexity O(D) — this is a huge gain. Because the trace of a matrix is simply the sum of its diagonal elements. Since the calculation of the trace is much simpler, , there is no longer any need to design special layers and the function f(z,t,θ) can now be any standard neural network (e.g., a multilayer perceptron, MLP). If we need to go the other way around - from a complex distribution to a simple one, in CNF we simply solve the same ODE but with backward time integration, from T to 0. Such continuous normalizing flows are a method for building generative models that uses the idea of Neural ODE. It replaces a sequence of discrete transformations with a single continuous "flow", which allows for more powerful neural network architectures.

These approaches are applicable to a wide class of first-order ordinary differential equations. The function f parameterized by the neural network must be such that it cannot change too quickly (Lipschitz continuous in state z or h) and continuous in time t. Most neural networks with activation functions such as tanh or ReLU satisfy these conditions with finite weights. However, this approach is not designed for stochastic differential equations (SDEs), which contain a random (stochastic) component. Dealing with SDEs requires other, more sophisticated methods. If the Lipschitz conditions are not satisfied (e.g., "explosive" dynamics, Levy flight description), then the solution to the ODE may not be unique or may not exist on the entire interval, which makes this approach inapplicable. You will also need to manually specify the accuracy tolerances for the solver on both the forward and backward passes, which is an additional hyperparameter.

1.5 Modeling of dynamic systems

The mainstream of neural network development has become the direction of creating language models, because the main thing is to solve the problem of learning. Working with texts from the Internet and communicating with a large number of users, neural networks quickly learned. Other directions of neural network development were initially pushed to the sidelines. However, the problems of solving equations, creating more accurate models describing real phenomena, returning to the integrability of such problems, that is, to the uniqueness of their solutions, nevertheless forced us to return to the creation of neural networks of a wider range. This is the process illustrated in this section.

Finding equations using sparse regression and machine learning. The paper [31] presents an approach called SINDy (Sparse Identification of Nonlinear Dynamics) for automatically inferring control equations from noisy measurement data. Sparse regression and machine learning methods are used with nonlinear dynamic systems. It is assumed that most physical systems have only a few key quantities that determine their dynamics. An attempt is made to create parsimonious models that find a balance

between accuracy and complexity, without overfitting.

It is noted that machine learning has made significant progress in solving problems with "static data", but progress in obtaining physical models of dynamic processes from big data has been significantly slower.

The SINDy technology uses sparse regression and compressed sensing. In this case, time series of states X and their derivatives $\dot{X}$ (which can be measured or approximated numerically) are formed. A pre-built library $\Theta(X)$ is used, consisting of possible nonlinear state functions (e.g. constants, polynomials, trigonometric functions). The problem is formulated as a sparse regression: $\dot{X} = \Theta(X)\Xi$, where Ξ is a matrix of sparse coefficient vectors. Recall that a sparse vector matrix is a matrix in which most elements are zeros, and nonzero elements, which are coefficient vectors, are sparsely located. In other words, it is a matrix where many elements are zero, and these zero elements are not stored, but only nonzero values and their positions are stored. Each column ξk in Ξ determines which nonlinearities are active for the corresponding equation $\dot{x}k = fk(x)$. To solve this problem under noise, LASSO (Least Absolute Shrinkage and Selection Operator), which promotes sparsity. Filtering methods such as common variation regularization can be used to handle noise, especially in numerical differentiation. In the usual approach, in addition to the basic and most important values of variables and parameters, many secondary indicators are also identified that are characteristic only of a given process and will not be characteristic of other similar phenomena. Usually, experienced researchers select from the mass of data exactly the most important ones that can influence the development of the phenomenon.

The sparse regression method - this selection is carried out automatically. This means that sparse regression is a type of regression analysis, the purpose of which is to build a model using only a small subset of the available features (variables), while the coefficients (weights) for all other features can be set equal to zero.

Step 1: Ordinary Linear Regression Ordinary.

Regular regression will try to find the coefficients w in the equation describing the estimate. It will choose w to minimize the prediction error. Most likely, all the

coefficients w_1, w_2, w_3, and w_4 will be non-zero.

Step 2: Sparse regression (LASSO method - Least Absolute Shrinkage and Selection Operator). The main idea of the method is to penalize complexity. LASSO solves the same problem as regular regression (minimize the error), but with one addition: it adds a "penalty" for each non-zero coefficient. The formula for the LASSO objective is: Minimize (Prediction Error + λ * Sum |of all w weights|), where prediction error: this is how much the model is wrong about the data. Usually, the error is not large. Sum of |all weights w: This is the L1 norm, or simply the sum of the absolute values of all coefficients ($|w_1| + |w_2| + |w_3| + |w_4|$). The basis is "penalty and complexity". The value of λ is the "penalty regulator". The user configures it. If $\lambda = 0$, there is no penalty, and LASSO turns into a regular regression. If you choose λ correctly, the model will seek a balance: it will leave non-zero only those coefficients that significantly reduce the prediction error, and zero out all the others, so as not to "pay" a penalty for them.

Step 3: Result of sparse regression. For example, you need to find a control equation for the system (for example, dx/dt = ?). The algorithm searches from this huge library for those 2-3 functions that actually determine the dynamics of the system (for example, σ*y and -σ*x), and zeroes out the coefficients for all the other thousands of functions. Let's also define what it means to identify nonlinear dynamics in a neural network. It means answering the basic question: What physical law or mathematical equation makes (or rather determines) this box behave this way? Instead of guessing what terms might be in the equation, SINDy creates a huge library of candidate functions. For a system with variables x and y, these might be: x, y, x^2, y^2, xy, sin(x), cos(y), x^3, x^2y, ... and so on, hundreds and thousands of possibilities. Then, using sparse regression, SINDy finds from this giant library the smallest possible number of terms that best describe the data. The idea is that nature is "parsimonious" and real physical laws usually consist of a small number of key components. That is, identifying nonlinear dynamics is an algorithmic process of discovering the succinct, fundamental governing equations of a system directly from the time series data, bypassing the need for a priori hypotheses about the structure of these equations. The SINDy method is

generally based not on the traditional approach to describing neural networks, but on sparse regression (for example, the LASSO method). First, a Feature Library is created: usually, a large library of all possible mathematical functions of the input data is created manually. For example, if there are variables x and y, the library will contain x, y, x^2, y^2, xy, sin(x), cos(y), etc. This is a pre-defined, explicit set of features. A linear model is used: The problem is reduced to finding a simple linear combination of terms from this library. In essence, this is a search for the coefficients w in the equation: derivative = w_1 * $function_1$ + w_2 * $function_2$ + ... + w_n * $function_n$ Here, sparse regression uses a special regression method (LASSO), which finds the coefficients w in such a way that most of them are exactly equal to zero. As a result, we get a simple and easy-to-read equation consisting of several terms from the original library.

Table 1 - Difference from traditional neural networks source [31]

OPTION	SINDy (method from the article)	Neural networks
Knowledge representation	Explicit, in the form of a simple equation (for example, dx/dt = σy - σx)	Implicit, in the form of hundreds or thousands of weights and biases distributed across layers.
Interpretability	High. The result is easy to read and understand for a person. This is the main goal of the method.	Low. The neural network is a "black box". It is very difficult to understand why it made a particular decision.
Feature creation	Features (functions in the library) are specified by a person in advance.	Features are automatically learned by the neural network in its hidden layers.
"Architecture"	Linear regression on an extended set of features.	Layers of neurons connected to each other, with activation functions.

Thus, although SINDy is a powerful machine learning method, it is fundamentally different from traditional neural networks. Its goal is not just to accurately predict, but to discover simple, fundamental, and human-understandable laws hidden in the data.

Models as assistants for scientific inquiry. In the paper [32], the authors discuss approaches to the long-term goals of "machine-assisted scientific discovery from experimental data without prior assumptions about the system." Analogies with the process of human physical reasoning, which has similarities with representation learning, are considered. This paper presents the development of SciNet, a neural network designed to discover physical concepts from experimental data. The authors aim to create a system that can infer the laws of physics without prior knowledge of the system in question, mimicking the human process of physical reasoning. The paper describes in detail the architecture of SciNet, where data is encoded into a latent representation and then decoded to answer questions about the system. The architecture of SciNet, with the exception of the question input system, resembles autoencoders used in representation learning. The network is trained on triplets (observation, question, correct_answer).

Architecture: SciNet uses feedforward neural networks for the encoder and decoder. To implement the desired representation properties, variational autoencoders (VAEs), in particular β-VAEs, are used, which encourage independence of latent variables (disentanglement).

Training: Optimization of the network parameters (weights and biases) is done using stochastic gradient descent and backpropagation.

SciNet mimics a simplified process of human physics modeling:

Observations (O): Obtain experimental data (e.g., a time series of a particle's position).

Representation (R) / Encoding (E): Generate a physical state (e.g., initial position and velocity). SciNet uses an "encoder" E: O → R for this purpose.

Questions (Q) and Answers (A) / Decoding (D): State analysis answers questions about the system (e.g. "where will the particle be at a later time t'?"). SciNet uses a "decoder" D: R×Q → A to generate answers.

SciNet is suitable if:

You have a set of observations (data) and questions that can be answered from these observations.

You are not sure which physical parameters are really important, or you want the network to extract and interpret them.

What is important is an interpretable model, not just a black box. It is necessary to find conservation laws, the dimensionality of the system, or natural variables.

How KL-divergence regularization helps solve problems The SciNet model, like the β-variational autoencoder (β-VAE), uses regularization using the Kullback–Leibler divergence (KL-divergence). Recall that a VAE (variational autoencoder) is a system that can recognize patterns in the received data and use this understanding to create new, similar data. This helps make the latent representation interpretable, i.e. teach the network to find separated physical parameters. Typically, a latent representation for machine learning and data analysis is a hidden, implicit representation of the input data that encodes its main features and structure in a more compact form.

KL-divergence in SciNet is a tool for disentangling physical factors, reducing redundancy, for interpretability of latent variables. It helps SciNet automatically find "natural" variables like frequency, momentum, angle, etc. without knowing anything about physics in advance.

Formally, the Kullback–Leibler divergence is a measure of the difference between two probability distributions. It is denoted as:

$$D_{KL}(P \| Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right) \qquad (5.1)$$

If $P = Q$, then $D_{\mathrm{KL}} = 0$, and the larger KL, the more P diverges from Q. In SciNet (via β-VAE), we teach the encoder not just to map the input to a point z in the latent space, but to map it to a distribution $p(z \mid x)$ — usually a normal distribution $\mathcal{N}(\mu, \sigma^2)$. Then we add a regularizer:

$$\text{Loss} = -\underbrace{\mathbb{E}_{z\sim p(z|x)}}\ \underbrace{\log p(x\mid z)} + \beta \cdot D_{\text{KL}}\left[\, p(z\mid x)\| p(z)\right] \tag{5.2}$$

where: $p(z)$ is the priority distribution, usually $\mathcal{N}(0,1)$; $p(z\mid x)$ is the distribution that the encoder predicts; β is a hyperparameter that controls the strength of the regularization. For β = 0: regular VAE, no regularization → latent variables can be messed up and uninterpretable. For large β: the network is forced to simplify the representation severely → this improves interpretability, but may slightly worsen accuracy. KL divergence is minimized when $p(z\mid x)$ close to $p(z)=\mathcal{N}(0,1)$. which forces neurons in the latent layer to be independent (not correlated). As a result, different neurons start storing different physical parameters (e.g. one — velocity, another — coordinate). Thus, the network uses the minimum number of parameters to describe the data.

Search for conservation laws. In [32], the AI Poincaré 2.0 program was developed, which, according to the developers, "deals directly with differential equations," which allows one to consider systems with dissipation or directionality, such as the Korteweg-de Vries (KdV) equation. The method works for both ODE and PDE (after discretization of space).

Step 1. Training neural networks to search for conservation laws

1. Formulation of the problem The equation is given:

$$\frac{d\mathbf{z}}{dt} = \mathbf{f}(\mathbf{z}), \tag{5.3}$$

where is the state vector $\mathbf{z}\in\mathbb{R}^s$, $\mathbf{f}$ is the vector field.

2. Ideology The conservation law, i.e. is preserved along the trajectory if:

$$\nabla H(z)\cdot f(z)=0. \qquad 5.4)$$

Let's check, indeed: $:\frac{d}{dt}H\left(\mathbf{z}(t)\right) = \nabla H\cdot\frac{d\mathbf{z}}{dt} = \nabla H\cdot\mathbf{f}.$

3. Parameterization H(z). It is given as a neural network H(z;θ). Training is carried out according to the loss function:

$$\mathcal{L}_1(\theta) = \frac{1}{P}\sum_{i=1}^{P}\left|\hat{\mathbf{f}}(\mathbf{z}^{(i)})\cdot\nabla H(\mathbf{z}^{(i)};\theta)\right|^2, \quad (5.5)$$

where $\hat{\mathbf{f}}$ and ∇H are normalized. Note that points 2 and 3 contain the main idea of the method: is an unknown function that does not change along the trajectory of a dynamic system (conservation law). A neural network is a universal approximator of any function. The authors take as a certain neural network with random parameters (weights) at the initial moment, which takes vectors as input and produces a scalar as output, and train it, achieving the condition ∇H(z)·f(z)=0 as a training criterion.

4. Search for several conservation laws. Models are trained. To ensure their functional independence, a regularizer is introduced:

$$\mathcal{L}_2 = \frac{2}{n(n-1)}\sum_{i<j}\frac{1}{P}\sum_{k=1}^{P}\left|\nabla H_i(\mathbf{z}^{(k)})\cdot\nabla H_j(\mathbf{z}^{(k)})\right|^2. \quad (5.6)$$

5. General loss function:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_1(\theta_i) + \lambda\cdot\mathcal{L}_2. \quad (5.7)$$

Step 2. Determining the number of independent conservation laws (rank)

1. A matrix of values is formed:

$$A_{ij} = H_j\left(\mathbf{z}^{(i)}\right), \quad (5.8)$$

where each row is the values of all H_j at one point $\mathbf{z}^{(i)}$.

2. Analysis of the manifold dimension. A nonlinear manifold learning method is used (for example, neural empirical Bayes - NEB) to determine the dimension of the manifold on which the points from A lie.

3. Alternative: differential rank. The gradient matrix is used:

$$B = \left(\nabla H_1, \nabla H_2, \ldots, \nabla H_n\right)^T, \quad (5.9)$$

and its rank is calculated (using SVD). The number of nonzero singular values determines the number of independent conservation laws.

Step 3. Search for symbolic formulas

1. Brute force (brute force - a method of complete enumeration) on the space of expressions. The search is performed for formulas given in the form of reverse Polish

notation[15] , ordered by complexity.

2. Quick verification of candidates

1. On a small number of pre-prepared points, the condition is checked:

$$\left|\hat{\mathbf{f}}(\mathbf{z})\cdot\nabla H(\mathbf{z})\right| < \epsilon_s . \qquad (5.10)$$

Candidates that do not pass the check are discarded.

3. Independence assessment

After finding a new function H, it is checked whether it increases the differential rank by one. If yes, it is independent of the previous ones.

The authors discuss in detail the various definitions of integrability with which AI Poincaré 1.0 and 2.0 relate.

1. General integrability requires that the conserved quantity be "a well-conditioned function globally, not exhibiting any fractal or other pathological behavior."

2. Frobenius integrability holds if locally the phase space has a bundle of invariant manifolds, i.e. "A first-order dynamical system with s degrees of freedom always has s-1 (local) integrals of motion."

3. Liouville integrability (algebra) requires the existence of a maximal set of Poisson invariants that commute with each other.

4. Landau integrability: this is a preference for "symmetric and additive IOMs" that can be interpreted as "fundamental conservation laws."

5. Decidable integrability (symbolic simplicity) requires that solutions be defined in explicit functional form. Experimental integrability (or robustness): A useful conserved quantity should not be "infinitely sensitive to measurement error."

6. Phase transitions At each phase transition, only one conserved quantity is learned or trained.

Conclusions: The presented AI Poincaré 2.0 method "can determine not only the

[15] Reverse Polish Notation (RPN) is a way of writing mathematical expressions in which *operators (e.g. +, , /) are written after the operands, rather than between them as in conventional infix notation.

number of independent conserved quantities, but also neural network (or even symbolic) representations of them." The algorithm "is able to adapt to all [competing definitions of integrability]. This approach is more useful to mathematicians, but is also of great interest to theoretical physics.

Automatic pattern detection. The authors of [33] presented the AI-Descartes system, a method for automatically detecting scientific patterns that combines symbolic regression (SR) and reasoning. An attempt was made to combine data and axiomatic knowledge in one method to extract derivable theories. They use background axioms (for example, Newton's laws), a set of auxiliary formulas, experimental data, modeling constraints, in particular, permissible errors, deviations from axiomatic representations. The code and data are available on GitHub:

https://github.com/IBM/AI-Descartes

The symbolic regression method is used, which consists of searching for mathematical expressions that correspond to the observed data with sufficient confidence. The search, as noted above, is often carried out by randomly combining mathematical operators such as +, -, ×, ÷, square root, logarithm, exponent, trigonometric formulas, as well as constants, etc. However, not all discovered mathematical formulas corresponding to the data may have scientific meaning. Therefore, automated reasoning tools are usually introduced into the algorithm. Symbolic regression itself is carried out using the mixed-integer nonlinear programming (MINLP) method, and an automated theorem proving system (for example, KeYmaera Xbkb and Mathematica)) demonstrating the ability to deductive and algebraic transformations is usually used as an argumentation tool. In particular, the KeYmaera X system is capable of forming a formal proof of the derivability of the obtained mathematical formulas using a set of known given axioms (say, the same Newton's laws), or, on the contrary, will show that the axioms and these formulas contradict each other. For each symbolic expression obtained in this way, the algorithm was able to determine the "level of deviation" both between the generated formula and the observed data, and between the formula and the theory. These "levels of deviation" were equivalent to an estimate of errors that could arise both from measurement errors

and from incompleteness and imprecision of theoretical premises. If the set of axioms did not allow the construction of a model, the system indicated that additional data were needed, or restrictions had to be added. The structure of the technology:

Background Knowledge is usually in the language of First-Order Logic (FOL), equalities, inequalities; addition, multiplication, roots, etc. It is desirable to achieve completeness of description and exclude contradictions (i.e. consistency). Role - Check whether the formula is logically derivable from the theory

Hypothesis Class - these are the sets of possible symbolic expressions (models) to be searched. It is a grammatical and structural constraint on the form of the equation. Allowed operations (e.g. +, −, ×, ÷, sqrt, log, exp); symmetries and equivalences: e.g. A + B = B + A. Constraints: monotonicity, variable extraction, choice of dimension. Role - Define the search space.

D - Data. A set of pairs (x, y), where: $x = (x_1, ..., x_n)$ are the independent variables; y — dependent variable. hidden form of functions. The degree of approximation of reality is important.

Modeler Preferences. Parameters that define the criteria for the quality of the model, by which the best candidates from C are selected. Acceptable error level (threshold for ε(f)); maximum complexity of the formula (tree depth, number of operations/constants); Interpretability requirements; if necessary, coefficients for multi-criteria optimization. Role - control of technology selection.

So far, these approaches have limitations: difficulties in extracting and formulating axioms from scientific texts; difficulties in presenting a complete and correct background theory; limitations of automatic proofs.

Physics control. In the methods of modeling dynamic systems, the case of using deep learning networks guided by physics (Physics-Guided Deep Learning, PGDL) is interesting. It is known that the advantages of physical models are that they are derived from first principles, guarantee conservation laws, are easy to interpret, however, as is often the case, researchers often use strong assumptions and techniques and estimates that are poorly understood by developers to form a specific model [34]. Developers of neural systems also often consider the numerical integration used for physical research

to be a disadvantage, although in the previous section some of them effectively used the capabilities of numerical integration. But arrays of data obtained from real-world observations leave no choice; deep learning neural networks have to be used. It is PGDL that is able to combine the capabilities of neural networks and the laws of physical laws, presented, in particular, in the form of differential equations. For example, Physics-Informed Neural Networks (PINNs) are a type of neural network that integrate physical laws represented as differential equations into the network training process. In other words, PINNs use partial differential equations (PDEs) or ordinary differential equations (ODEs) to guide the neural network training process, making its decisions more consistent with physical laws.

Despite the fact that neural networks are a black box, which should reduce the interpretability of their output, they are nevertheless able to fill in the gaps in partially mastered physical models. When mastering a number of physical models as a result of training, the neural network, receiving and examining a data set, is able to see patterns already known to it. And if there are no embedded patterns, then regression methods can be used. Indeed, regression serves to determine the strength and nature of the relationship between one dependent variable and a number of other variables or their averaging or even the entire array. For example, symbolic regression is a key method for training interpretable models based solely on data.

In particular, if the algorithms are unknown, a program can be developed that can generate algorithms for solving problems. This is genetic programming, and its special case is the symbolic regression problem, the goal of which is to find a formula that describes a certain dependence.

More formally, it can be formulated as constructing a regression model as a superposition of given functions. A syntax tree can be an option for representing a program. In this case, the role of "programs" is played by algebraic formulas. Crossing can be implemented by exchanging randomly selected subtrees: this is replacing a function with a randomly selected one; replacing a subtree with a randomly generated subtree; deleting or creating an intermediate node; rearranging child subtrees, etc. The coefficients of the elements of each tree are optimized. And to optimize the

coefficients, a genetic algorithm is used!

The DL (Deep Learning) model generalizes System ID with a more expressive architecture, efficient computations, and minimal assumptions, which is what it was created for. But the DL model architecture still relies heavily on engineering intuition and trial and error. You can choose from the following set: Recurrent neural networks (RNN), LSTM, GRU Convolutional LSTM (ConvLSTM), PredRNN, pure convolutional architectures (ResNet, UNet), Transformers. The goal of training is to teach how to solve differential equations using some examples. Or to find a control equation, for which a search for the best combination of functions that matches the data is carried out from a set of mathematical basis functions. How to integrate physical knowledge? DL learns from the difference between the physical model and the observed data. In this case, you can introduce restrictions or conditions, for example, in the form of including symmetry. Inductive physical biases, such as symmetries and additivity, can be built into the neural network architecture or into the loss function.

Conclusions

The review considers the process of evolution of neural networks of deep learning. It is shown that the initial impetus was given to the development of language models for two reasons: most people were interested in how artificial intelligence can interact with users and help them navigate their new life in the information society. People wanted to move away from routine intellectual work, as they did by creating automatic devices that solved the problem of logistics, creating images, the problem of printing books and carrying out tedious paperwork and much more. The second reason is the facilitation of training large neural networks-models, due to the use of texts and speech on the Internet, where humanity has accumulated huge amounts of information. It is clear that only language models were able to master these multi-year layers of data. Training was also facilitated by the communication of billions of users with neural networks, which provided such opportunities. It is not surprising that the development of language models went by leaps and bounds. Now, due to millions of additional

devices-plugins, language models are able to work in a wide variety of areas of human activity. Having satisfied its curiosity and got stuck in the technologies of using language models, humanity returned to the creation of neural networks capable of solving mathematical, computational problems, searching for patterns in data arrays and predicting the behavior of many processes. Neural network developers returned to integrated problems capable of providing a single solution. Gradually, the scope of application of neural networks of various types is expanding and it is time for the majority of humanity, previously inclined only to communicate with language models and solve their personal problems, to begin to master the new opportunities that are opening up for humanity in this new era of artificial intelligence development. This review is devoted to this.

KAPITEL 2 / *CHAPTER 2* [16]

[16]***Authors:*** *Babchuk Serhii*
Number of characters: *58035*