Scientific Visualization, 2025, volume 17, number 2, pages 123 - 136, DOI: 10.26583/sv.17.2.09

Visualization of Methods of Machine Learning. GUI Programming

N.O. Shesterin[1]

National research university "HSE" (Higher School of Economics), Moscow, Russia

[1] ORCID: 0000-0003-2134-8412, nshesterin@gmail.com

Abstract

The technologies of artificial intelligence and machine learning have made a fundamental leap in their capabilities in the last five years. The growth of processing power and the emergence of more and more effective methods of machine learning allows AI to not just solve the most typical tasks associated with the field, such as statistical analysis and optimization of mathematical processes, but also to find new applications in related fields of research, as well as practical applications, including those on the free market, available to the mass consumer. Image generation, audio, animation, self-learning models of control of robotic platforms and virtual mechanical models – these and many more novel applications of the recent years have led to a media-boom around AI and a growing interest from developers and authors from various fields and industries.

That being said, the methods for developing, research, testing, and integration of AI have largely remained unchanged and still require the knowledge of programming languages, machine learning libraries, as well as a deep understanding and experience specifically in the narrow field of AI. This barrier of specialization not only demands inclusion of machine learning specialists in the development process of otherwise trivial computer applications, typical for the field of AI, but also prevents small teams and independent developers from using the latest advances in these technologies without significant monetary and time investments into studying the subject.

I offer a novel solution to this issue in the form of a prototype graphical interface that allows the user without technical education and without the need for knowledge of programming languages to develop and tune various architectures of neural nets and other machine learning methods, methods of unsupervised machine learning, and to test these methods on a wide range of experimental tasks – from mathematical equations to controlling virtual mechanical models in a simulated physical environment. In this article, I give a brief description of its structure and organisation, its fundamental principles of operation, and the capabilities of this GUI.

Keywords: neural net, artificial intelligence, machine learning, block programming, graphic interface.

1. Background

Methods of machine learning and overall, the products of artificial intelligence (AI) nowadays is applied in a wide range of both fields of scientific research and practical applications, including the development of commercial products targeted at the mass consumer with no special skills or prior technical knowledge. The range of applications as well as the capabilities of those methods themselves has been steadily growing in recent years, which can be largely attributed to several key advancements in the fields of deep neural networks, unsupervised learning, adversarial neural nets, and generative algorithms. These new technologies are developing at an increasing speed, and new accomplishments and discoveries are being

made over the course of months or even weeks. Following the emergence of high-efficiency generative algorithms, a media-boom has occurred around AI, image and, more broadly, media-generation, followed by a second boom around LLMs (Large Language Models), has attracted a large number of developers from various fields and industries, often tangentially related to AI and even to IT in general. All of this has made pertinent the task of creating visually representative interfaces for working with AI, similar to block-based programming.

Despite high practical and theoretical relevance, this area is scarcely researched. In this article, I attempt to eliminate this blind spot.

2. State of the industry

Today, machine learning methos and AI solutions are being actively integrated in the media-industry, the movie industry, in the online environment, in the entertainment industry, in the various areas of design, marketing, in the context of scientific research – in biology, medicine, chemistry, genetics, physics, astronomy, robotics, information security, etc. This is happening despite the fact that, until recently, the limitations in processing power and data storage as well as the capabilities of machine learning methods themselves, have prevented AI from getting outside of a narrow range of special applications in the fields of computer modelling, statistical analysis and other largely theoretical areas of scientific and technological exploration. The modern processing capabilities have allowed regular individuals and independent developers to not only utilize, but also train and even develop AI solutions on commonly available mid-range machines or with the use of cloud computing. This, together with an exponentially growing interest in AI in a wide variety of areas of science and tech, has led to an increasingly large number of people working with AI both as users and as developers.

At present time, there exist several interfaces and APIs designed for specialists with minimal experience and even for casual users, that allow the user to generate images, music, animation, deep-fakes, and overall exploit already trained AI models. There are several chat-bots on the market that provide LLM-powered services and allow users to generate texts of large volumes, from prompts, to complete partially written texts. There also exist several software products and services that allow developers to integrate AI solutions into projects in the field of IT with minimal prior experience and without a necessity for specialist education with a focus on AI or data science: services, offered by Google, Amazon, Microsoft, and other companies allow development teams to generate media live, through a text prompt or a prompt in pseudo-code, utilizing AI tools such as Midjourney and Dall-E, often paired with technologies such as ChatGPT and similar LLMs, which also allow for quick production of long-form texts of various styles.

These services offer not just media created by AI but also integration of AI interfaces into web applications and desktop applications with a reliance on cloud computing. However, to develop proprietary or custom AI solutions, to experiment with neural architectures and test various machine learning methods for one's specific task, there is still no recognized industry-leading app or service that would allow non-specialists to work with AI without the need for coding.

Most often, developing custom neural architectures, training neural networks and, of course, developing new methods of machine learning requires familiarity with the Python programming language, libraries of machine learning like TensorFlow, Keras and others. This knowledge and skills are demanded from specialists in areas otherwise unrelated to programming and IT (for example, in astrophysics most calculations and statistical analysis can be performed in programs like Wolfram Alpha or with pseudocode, but to work with AI the knowledge of Python is basically necessary). This not only complicates the training process of specialists in many industries (including in creative fields), but also creates a hard to breach barrier for non-specialists that often lack time and opportunities to go through often paid online-courses, with no real guarantee of quality and relevance of the material being taught.

I can see two possible reasons for the lack of availability of such applications. The first reason is related to the often-overwhelming speed at which new technologies appear in this field – an application that allows the user to work with machine learning (and other AI methods) has to be continuously updated and supported by the developer to maintain relevance and perform competitively with lower-level APIs such as TensorFlow, or in turn has to be designed with a reliance on these APIs, which would make such an application dependent on them to stay functional. The second possible reason lays in the huge variety of architectures and operating principles of various machine learning methods that make a task of creating a universally applicable visualization daunting, if not impossible, at least at the level that would maintain ease of access and a degree of intuitive design apparent to new users.

Over the course of my dissertation, I have developed a prototype of a software solution that creates a graphical user interface which allows the user to develop machine learning solutions and, in general, mathematical systems through the use of block-based programming, with a minimal required prior knowledge of machine learning algorithms and without a need for traditional coding. This solution is based on an independently developed machine learning library in C++ and is thus not reliant on any external software product. The developed software allows the user to not only develop custom neural architectures of commonly used machine learning methods, but to also train them locally, without the need for cloud computing services. This software also allows training of these methods on user-defined tasks and in a simulated physical environment, designed specifically for testing virtual mechanical models, akin to AI-gym and Mujoco.

3. Neural architecture editor

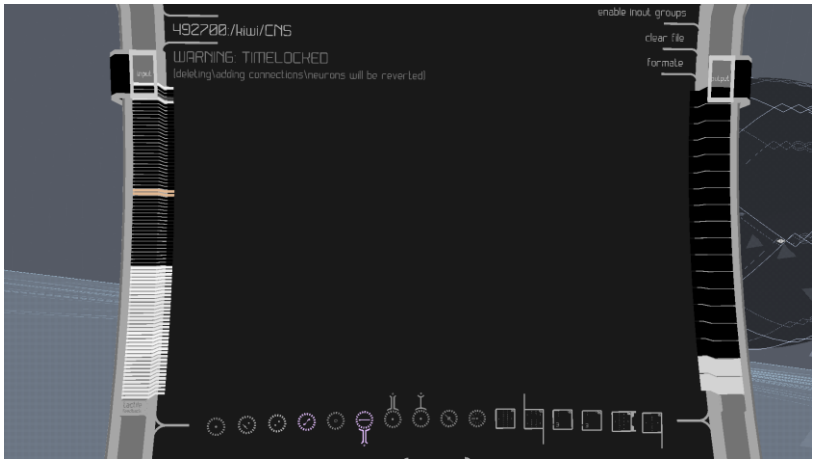

Fig. 1. Neural editor's interface.

The developed software solution allows the user to design neural networks through the use of a graphical interface, and to interact with them through the use of mouse and keyboard. The objects of this interface are formalized and visualized through pictographical icons of two types – "Neurons" and "Encapsulated Learning Units", pictured at the bottom of fig. 1. The first type represents simple mathematical functions: linear, ReLU, leaky ReLU, hyperbolic tangent, a frequency generator, a pseudo-random number generator, a "multiplier neuron", the "memory cell", and the "excitable neuron". I will describe the functions of the latter three kinds of "neurons" further in the text. The second type of objects is called "Encapsulated Learning Unit" and represents an artificial neural network that works through the use of machine learning. At present time, six kinds of this type of object are available to the user: a fully connected neural net that implements several machine learning methods depending on the task at hand, a "combinator" unit that allows the user to join data feeds from several neural nets into one and to split a data feed into two but is otherwise identical to the first kind, a convolutional net, a "memory unit", an "actor" net that implements the DDPG method and

other related unsupervised learning methods of the same core operating principle, and finally – an adversarial net.

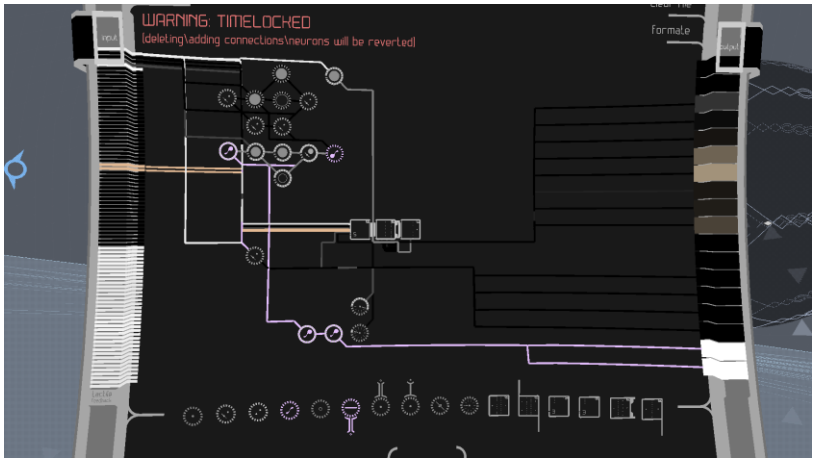

Fig. 2. Editor's workspace.

These objects are placed by the user in the central workspace area of the editor, as pictured on fig. 2, in a free manner and order, as is convenient for the user at the time. The transfer of information between these objects is done through "connections" – visually represented links. For example, a link between two "neurons" creates a sort of "compound" function: a ReLU neuron connected to a TanH neuron will result in a mathematical operation with a result that is identical to (1).

$$f(x) = Tanh(ReLU(x)) \quad (1)$$

When creating a "connection" between a "neuron" and an "Encapsulated Learning Unit", as is pictured on fig. 3, the user can choose between three types of connections that define the "role" that the information will serve when delivered to the neural net from the "neuron", or from the neural net to the "neuron": the first type of connection, the one pictured on fig. 3, sends the output signal of the "neuron" to an input of the neural net. The user can create compound connections and send outputs from several neurons to the neural net in the form of an array of data, or connect several neurons to the same input, which will be identical to adding up their outputs and sending the result of the operation to the same input of the neural net.

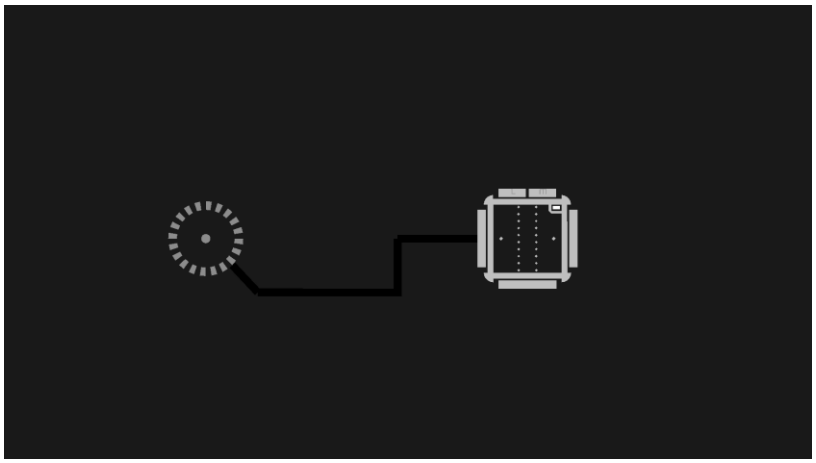
Fig. 3. A "Neuron" and an "Encapsulated Learning Unit" with a connection between them.

The second type of connection interprets the information sent to the neural net as a target value in cases when the type of neural net the user is working with allows for the use of target values. This type of connection acts similarly to the first one – connecting several "neurons" to the same input of a neural net will equate to an addition operation.

The third type of connection is interpreted as an output of the neural net – an output value of the network's output layer will be sent to a connected "neuron". The number of connective elements of this type for an "Encapsulated Learning Unit" (on the right of the icon on fig. 3) is equal to the number of neurons in its output layer, while the number of elements of the first type is equal to the number of neurons in its input layer (on the left of the icon on fig. 3). The second type of elements (at the bottom of the icon on fig. 3) is equal in its number to the third kind of elements for obvious reasons. This organization allows the users to create non-standard configurations of neural nets, for example – unsupervised learning methods with several independent reward values.

In the middle section of the "Encapsulated Learning Unit" icon there is dot-like elements that visualize the neurons of all layers of the neural net (in the middle of the icon on fig. 3). The user can change the number of neurons in each layer through the use of the triangular interactive elements above and below each column of neurons. The user can also change the number of layers with similar triangular elements at the left and at the right of the icon. By the user's choice, these dots can remain greyed-out, or change brightness depending on the activity of the corresponding neuron during online operation at each timestep. This option as well as many others is available in the settings menu of each unit, which allows the user to change all relevant training parameters of the neural net, from learning speed to optimization algorithms. Depending on the kind of unit, additional options may be available – for example, the number of elements in the replay buffer for neural networks that implement the DDPG method or others of the like. I will cover the available settings in the later section of this text.

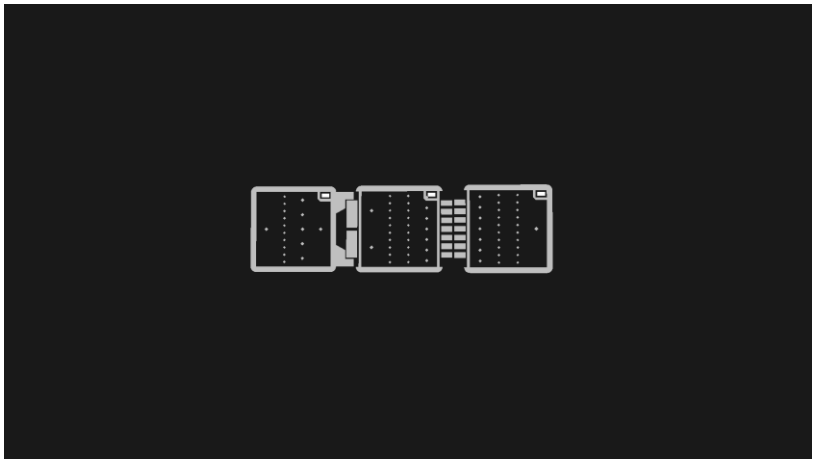

Fig. 4. A chain of "Encapsulated Learning Units".

Just like "neurons", the trainable units can be connected to one another through the use of "connections". In this case, a value or an array of values from the neurons of the output layer of one neural net will be sent as inputs or target values to another. However, trainable units can also be joined together into "chains", as pictured on fig. 4. When placed directly next to each other, several trainable units will share their training process: arrays of data from input to output will be parsed through each unit in the chain in a computationally efficient manner. However, Gradients and other training data will be also propagated back during training, allowing all units in a chain to be trained to perform one task, with each serving different functions depending on its parameters. Gradients, calculated for the input layer of a unit on the right, will be sent to the output layer of a unit on the left. When organized in this manner, the units will have both shared and independent training parameters. Individual units, or sets of layers of this compound neural net, will have their own training speed or can have their training disabled altogether.

The shared parameters, such as for example minibatch sizes, can be set for all units in a chain at once, and will become available depending on the type of chain, which will also define the specific machine learning method that the system implements.

When an “actor” unit is included in the chain, the entire chain switches to implementing DDPG, with all units to the right of the “actor” being treated as parts of the “critic”, and all units to the left of the “actor” as parts of the “actor”, with one or more layers in each unit of the chain. This organizational principle not only allows the user to work with neural nets in a convenient ergonomic manner, but also gives the user an intuitive understanding of how this method operates, and in what role each of its components is going to function.

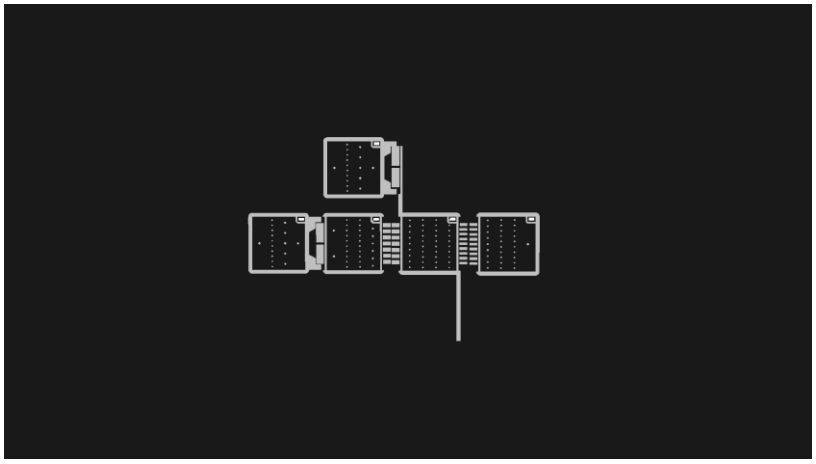

Fig. 5. A chain of units with a joiner unit (third from the left).

When a “combinator” unit is added in the chain, like the one pictured on fig. 5, the values of output layers of two independent chains of neural nets will feed into one data array, while gradients, generated later in the chain will propagate back and split into two data arrays of corresponding sizes that will be sent to the two independent chain segments earlier on in the chain, to the left of the “combinator” unit. A similar principle is applied when two independent chains are connected to the “combinator” at the output – output values of one or two chains will be processed by the hidden layers of the “combinator”, if those layers are included, and then split into two data streams that will be parsed to two chains on the right of the unit, with array dimensions similarly equal to the dimensions of the connected units. This organizational system not only allows the user to create custom neural architectures and experiment with them (for example, one chain to the left of the "combinator” can receive gradients from two independent neural nets with their own target values), but also to implement methods from the DDPG “family” that require several “actors” or “critics” to function in parallel.

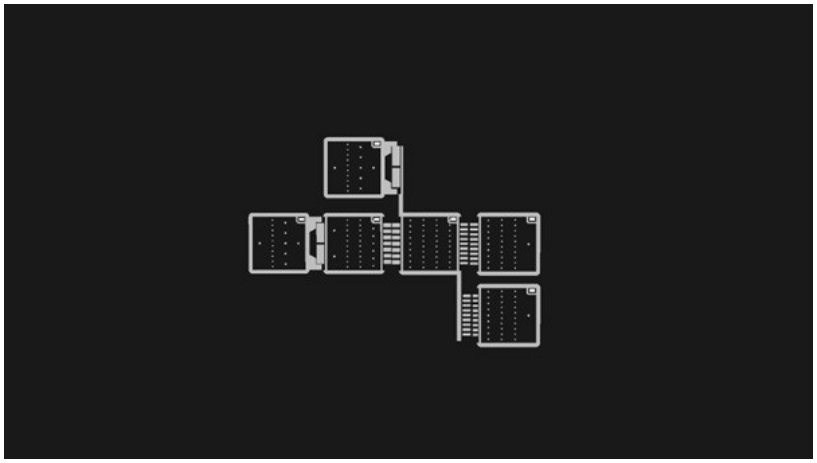

Fig. 6. A chain of units implementing EAC.

For example, the chain pictured on fig. 6 implements EAC (Explorer Actor-Critic) method that utilizes the architecture with two independently trained “critics” and “actors”. When only one chain of actors is connected, the chain switches to implementing TD3 (Temporally De-

layed Deterministic Gradients). This principle of visualization of neural architectures allows the user to intuit, which transformations are applied to the data as it passes through the various stages of the training process, and how the data is exchanged between its compounding neural nets.

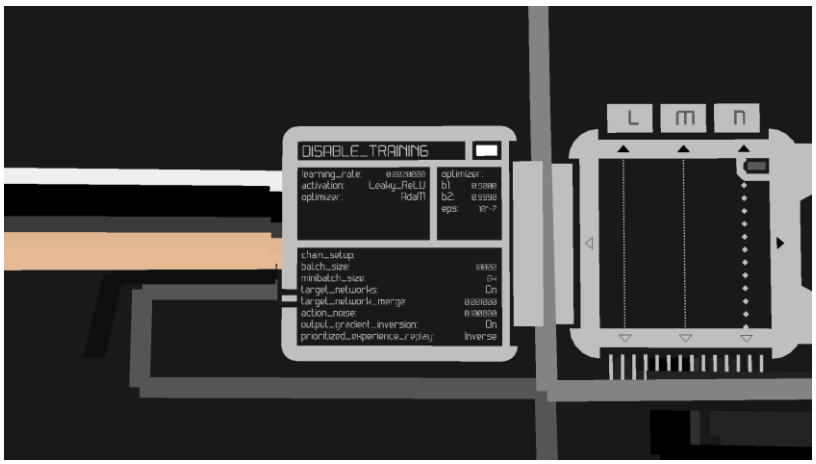

Fig. 7. Settings' menu of an encapsulated unit.

"Encapsulated Learning Units" allow the user to change most of the relevant training parameters of a neural net, as well as to apply a wide variety of optimization methods to the training process (settings menu pictured on fig. 7):

1. Learning rate – defines the speed at which the neural net's trainable parameters, such as weights and biases, are updated at each training step;
2. Activation (the activation function) – defines the activation function for neurons in the hidden layers of the neural net. A range of functions is available – ReLU, SoftMax, TanH, Sigmoid, etc.;
3. Optimizer – applies to the neural net one of the training optimization methods, based on momentum of the changes to the weights and biases of the hidden layers, with the following methods available: Momentum, AdaDelta, AdaGrad, AdaM, NadaM. This option being turned on makes an additional menu section available (fig. 7, top right section of the menu), where the user can edit their parameters;
4. Batch size – parameter, shared by all units in a chain; defines the size of the replay buffer when training methods from the DDPG "family". When the parameter is set to 1, the training is conducted without the replay buffer, in online mode;
5. Minibatch size – parameter, shared by all units in a chain; defines the size of the minibatch during training. When set to 1, the training is conducted in online mode, with a minibatch of one element collected from the last timestep;
6. Target networks – parameter, shared by all units in a chain; a specialized parameter for methods of unsupervised learning – DDPG, TD3 and EAC, adds target networks to each unit in the chain, that over time is adjusted to match the trainable parameters of the main neural net; is used during training to stabilize the "critic's" predictions of the reward value;
7. Target networks merge – parameter, shared by all units in a chain that utilizes target networks; defines the rate of adjustment of the target networks after each training step;
8. Action noise – defines the amplitude of the pseudo-random values added to the "actor" net's output in online mode during training;
9. Output gradient inversion – inverts the gradients of the output layer of a neural net when its absolute value exceeds a certain limit;
10. Prioritized experience replay – enables methods of memory optimization that order the elements of the replay buffer during training of DDPG and the like in such a way that prioritizes elements of the buffer with notable qualities, valuable for training. Three settings are available in this regard: the first one is a standard implementation of the optimization meth-

od and prioritizes the training examples with the highest training error values. The second and third ones are developed by this project and prioritize either the examples with the highest reward values, or the examples with the highest deviation from the average of all reward values.

When working with "Encapsulated Learning Units", the user can also connect "neurons" to specialized connective elements of a trainable unit that allow the incoming signals to influence the training of said unit. These connective elements are situated at the top of the unit's icon (top of the icon on fig. 3) and by default allow external signals to alter the value of the learning rate, the amplitude of the output noise, as well as to block the parameters' update of the neural net of a specific unit.

This software allows the user to implement a wide range of modern machine learning methods, with a selection based on the results of numerous studies (1, 2, 3, 4, 5, 6, 7, 8, 9, 10). The development of the unsupervised learning methods based on DDPG was based on a number of studies by foreign and native researchers (11, 12, 13, 14, 15, 16). The result is a programmatic implementation of DDPG and TD3, that is functionally identical to their typical implementations, but different in the principle of data organization of both the replay buffer and the minibatches, which allows to efficiently store vast amounts of data when working with and storing multiple experimental neural architectures.

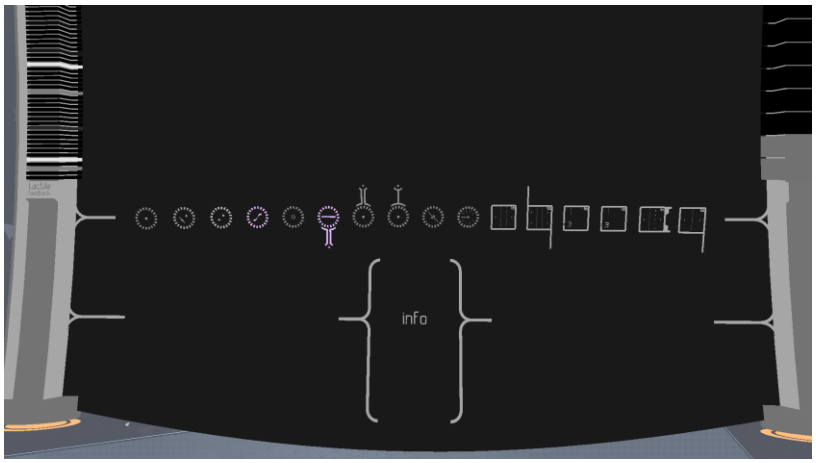

Fig. 8. Menu of object selection.

As can be seen from the list above, my interface not only allows the user access to most parameters relevant to the training process, but also facilitates a high degree of flexibility when designing neural architectures and methods. These parameters are available for all trainable units. There is however two other, specialized trainable units that possess their own unique functions and parameters:

«Memory unit» (thirteenth icon from the left, fig. 8) – a type of unit that saves input data arrays from previous time steps and feeds it to the input layer of its neural net, which allows it to access data from previous time steps and "remember" what happened during a user-defined number of previous states of the training environment;

«Convolutional net» (fourteenth icon from the left, fig. 8) – a convolutional neural net that is typically used when working with data of "multi-layered" structure and complexity (for example, with images). The unit implements a neural net with input layer size several times smaller than the size of the input array. The net's input layer is then applied to each segment of this array, searching for typical patterns (e.g. textures, borders, shapes, etc.);

Both types of units possess an additional parameter that allows the user to set the number of the input segments of the convolutional neural net and the number of past states, stored by the "memory unit".

4. A comparative analysis of the computational efficiency of the software

I have conducted a comparative analysis of my implementation of the DDPG method in this software with an operationally identical from the point of neural architecture and training parameters implementation, made with TensorFlow – a machine learning library. This "control" implementation in Python has been developed by a student of MIT, Steven P. Spielberg (17). His implementation has shown commonly expected results in terms of efficiency and is often cited online as template to start with, for those developing their own AI solutions for unsupervised learning. Both implementations have only been allowed to use regular processing, without utilizing graphical computing. Both implementations have shown identical performance when solving a range of commonly used tasks in the Mujoco testing library, specifically: inverted pendulum, cartpole, Half-cheetah, Reacher3D. The training has been conducted in online mode, with the frequency of the testing environment steps equal to the update frequency of the neural net – one step per update. After 10,000 training steps and after 100,000 training steps, no noticeable difference has been found. However, my implementation has performed slightly better in terms of the number of steps processed in the same amount of time - approximately 12% faster than the "control" TensorFlow implementation. In the future, I plan to expand the functionality of my software implementation and utilize graphical processor computing as well.

5. Experimental functionality of the software

This interface allows the user to implement complex, "compound" mathematical functions through the use of "neurons" and thusly construct tasks for the neural nets to solve and train against. In the future, an expansion of these capabilities is planned with the addition of "compound units" – graphical representations of mathematical and logic chains of "neurons", compiled in a single unit that occupies a single position on the editor's grid and has several input and output panels, akin to "Encapsulated Learning Units". To facilitate training of unsupervised learning methods, an experimental virtual environment has been developed and integrated in this software, with a physics simulation based on Newtonian Mechanics. This environment allows the user to test unsupervised learning systems on typical virtual mechanical models with rigid bodies as well as models with soft, point-cloud based bodies, designed by the user for custom tasks.

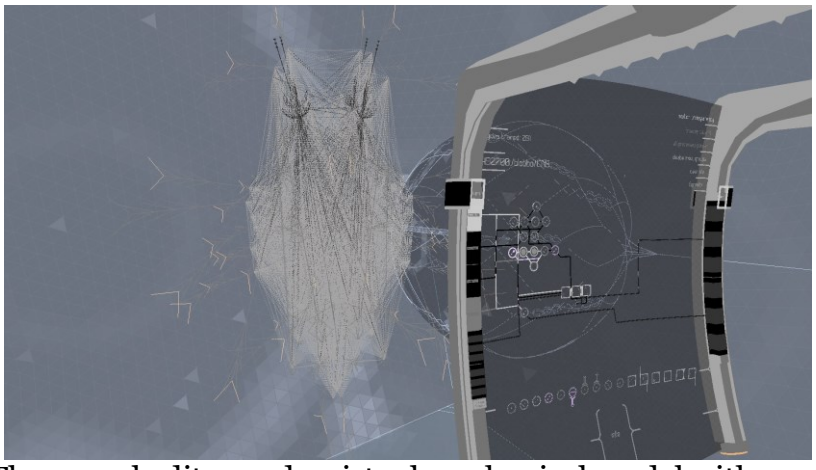

Fig. 9. The neural editor and a virtual mechanical model with a soft body.

When working with virtual models (Fig. 9, on the left) that interact with the simulated physical environment, input and output values that receive data from virtual "censors" and send data to virtual actuators, are visualized in the interface as two sets of panels that act akin

to those of “Encapsulated learning units” – input panels on the left, output panels on the right of the main workspace of the editor (the editor is pictured on the right of fig. 9).

These virtual mechanical models allow the user to implement both typical experiments for unsupervised learning methods (pendulum, inverted pendulum, cartpole, etc.) from training scenarios from databases like Mujoco, as well as tasks involving unique mechanical models with soft bodies based on point cloud structures.

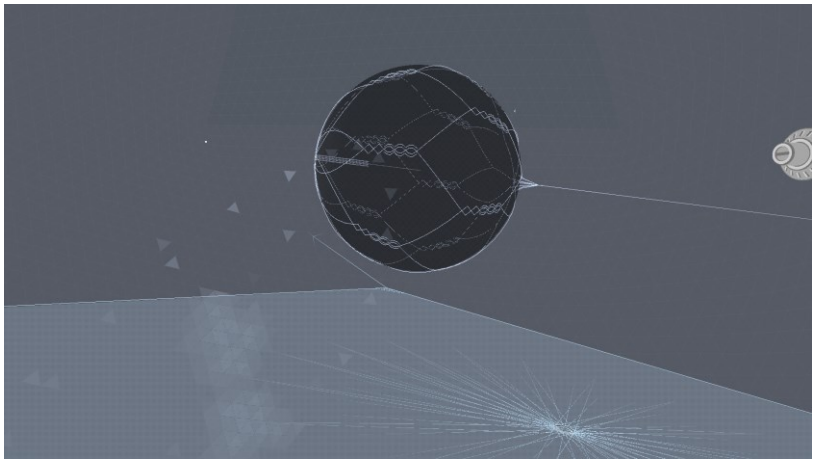

Fig. 10. Geometric primitives of the training environment.

This environment also allows the user to create geometric primitives – spheres (Fig 10, at the top of the image), surfaces (Fig 10, at the bottom) and their combinations. The development of such involved and versatile environment for experimentation is due to, in part, the software being based on a custom machine learning library, which means that it cannot interface with the standardized testing environments, such as Mujoco. This limitation can be characterized as a detriment, however the discontinuation of support for environments like AI-gym and Mujoco in recent years has shown that the reliance on third-party software solutions leads to a potentially limited lifespan of proprietary software and all but guarantees that it will eventually become unusable over the course of years or even months. Thusly, basing my software on native integrated solutions when it comes to machine learning and training, I ensure longevity and freedom of access of the software for learning and research.

When developing the experimental virtual environment, I have taken into consideration the results of the studies, focusing on analysis of potential of online learning in environments with a high degree of input noise (18, 19, 20, 21, 22, 23), which is especially relevant when applied to virtual mechanical models with soft bodies. I have also relied on research into technologies of signal propagation (15, 23). The 3-Dimensional visualization of the environment has been developed and will continue being developed, based on the article “Basic principles of data visual models’ construction, by the example of interactive systems for 3D visualization” by A.A. Zakharova and A. Shklyar (24). Through the course of the project, I have also developed a method for synchronized propagation and delivery of signals in a chain of “neurons” and “Encapsulated Learning units” within a single time step of the testing environment.

When working with the testing environment, to observe the tested models, the user can utilize a free-moving camera that is controlled through the use of mouse and keyboard. Several testing models can be loaded and worked on at once, including simultaneous training and editing of both virtual mechanical models and neural architectures. This software implementation can visualize and process a high number of physics objects (up to 64 simultaneously loaded and trained template rigid and semi-rigid models, up to 32 models with soft bodies based on point-cloud structures) and has been designed based on the core principles of effective 3D-visualization of data, described by P. Vasev and S. Porshnev in the article “An Experience of Using Cinemascience Format for 3D Scientific Visualization” (25).

This software allows the user to track, log, review, and compare experimental results with visualizations of the observed reward values of unsupervised learning methods and target values of "Encapsulated Learning Units" through the use of graphs. The visualization of this information is done in accordance with the general principles of scientific visualization of data outlined by D. Manakov and V. Averbukh in their article "Verification of visualization" (26), specifically: an option has been added for normalization of reward values, an option for the creation of splines, of visualizing both the absolute reward and target values and their first derivatives. An option has also been added for the display of the final 50%, 25% and 5% of each training cycle as a way to mitigate the effects of initial noise from environment randomization at the beginning of each training cycle.

6. Formal description of the software.

This software product is being developed in C++, in the development environment Visual Studio (Visual Studio 2019) for Windows PC (Windows 10+, 64bit). To facilitate fast, high bandwidth matrix operations when working with AI and processing the physics simulation, the application uses the Armadillo library. To implement volumetric sound, the app uses the OpenAL sound library. For graphical rendering, the OpenGL library is used. This software is a commercial product and has been registered on Steam distribution platform. Presently, it is the final development stages and is being planned for release in the form of a beta version in the near future.

Below is the list of the functional capabilities of this software:

1. Developing, editing, and testing of logic chains, chains of neural nets through the use of a graphical user interface;
2. Developing complex, modular neural architectures with independent training parameters for different modules with the use of a graphical interface;
3. Training neural nets, including online unsupervised training through the interaction with a testing environment;
4. A testing environment that implements a physics simulation based on Newtonian mechanics – for training DDPG and other like methods;
5. Implementing such methods as DDPG, TD3, EAC for unsupervised training, adversarial and convolutional neural nets. Implementing various user-developed methods of training and neural architectures through the use of the graphical interface and block-based programming with a high degree of freedom of experimentation;
6. Visualization of neural activity of hidden layers, input and output layers of neural nets, of internal states of logic chains, of target and reward values (if applicable);
7. Development of 3D environments from basic geometric primitives for training, options of overseeing the training process with a free-floating virtual camera, controlled with mouse and keyboard;
8. Key-binding custom user-selected keys for various commands;
9. Developing custom mechanical models for testing unsupervised methods, through the use of a built-in editor;
10. Saving and loading files of neural architectures, virtual mechanical models and experimental environments, with an option of automatically generating neural architectures and typical training setups based on built-in templates;
11. Logging, replay, and saving of experimental results – the dynamics of reward values (in graph format), of fulfilling testing criteria, defined by the user.

7. Potential applications of the software

This software visualizes neural architectures and experimental environments; it can be utilized by both specialists from intersecting fields with no experience of working with AI, and regular people unfamiliar with the technology. The software offers a full glossary, a series of instructions and typical tasks for neural nets as well as templates of several common neural

architectures. It also allows for automatic generation of AI solutions for mechanical models with pre-made templates – ones made by the user and by the developer. After finishing this software and it being released on the market, I plan to share the machine learning library at the heart of the project as public domain, which will allow other projects to benefit from its capabilities. This visualization and the included editor of neural architectures is a unique, ergonomic and visually informative solution for representing abstract, often counter-intuitive processes and mathematical methods. At the time of writing, it does not have competitive alternatives on the market.

8. Relevance of the study

At present time, most scientific achievements and technologically impressive results in the field of Artificial Intelligence are being made by large foreign and international corporations and corporate research teams. My software implementation and other projects like it can aid in "democratization" of this field of study and provide an opportunity for participation, integration, and development in the market niche occupied by these enterprises to small teams and independent developers, which among other things will allow for more active integration of AI technologies into the creative fields (media generation, development of independent AI-agents for in-game videogame environments, etc.) by studios with little financial and labor resources.

9. Conclusion

Artificial intelligence has been firmly integrated into many fields of scientific research, into the creative fields, into many industries, often only distantly related to IT. The number of the fields of application will no doubt continue growing, as will the variety and efficiency of methods of machine learning. At present time, there is no applications available on the market that allow a casual user or a specialist from an IT-tangential field a degree of depth and flexibility of development comparable to that of low-level machine learning libraries and APIs that require a familiarity with coding.

The developed software solution visualizes neural architectures, math functions, and allows the user to work with them, design custom architectures through the use of a minimalistic ergonomic graphical user interface. This software implementation offers several most commonly used methods of machine learning, including ones developed in recent months, such as EAC. In terms of processing speed, the software has shown competitive results when compared to a "template" TensorFlow implementation, with a slight advantage.

The developed visualization can be utilized as a tool for training specialists, non-specialists, hobbyists, as well as for experimentation and the development of novel neural architectures and methods of control of virtual mechanical models. This implementation as well as others like it can assist in democratization and integration of AI technologies into various fields of scientific study and digital industries.

References

1. Deisenroth M., Rasmussen C.E. A model-based and data-efficient approach to policy search // Proceedings of the 28th International Conference on machine learning, 2011, pp. 465–472.E

2. Foster D.J., Matthew A. Reverse replay of behavioural sequences in hippocampal place cells during the awake state // Nature, Vol. 440, 2006, pp. 680–683.

3. Glascher J., Nathaniel D, Peter D. States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning // Neuron, Vol. 66, 2010, pp. 585–595.

4. Van Hasselt H. Double Q-learning // Advances in Neural Information Processing Systems, 2010, pp. 2613–2621.

5. Hinton G. E. To recognize shapes, first learn to generate images // Progress in brain research, Vol. 1652007, 2015, pp. 535–547.
6. Koutnık J., Schmidhuber J., Faustino G. Online evolution of deep convolutional network for vision-based reinforcement learning // From Animals to Animats, Vol. 13, 2014. pp. 260–269.
7. Koutnık J., Schmidhuber J., Faustino G. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning // Proceedings of the 2014 conference on Genetic and evolutionary computation, 2014, pp. 541–548.
8. Krizhevsky A., Sutskever I., Hinton G.E. Imagenet classification with deep convolutional neural networks // Advances in neural information processing systems, 2012, pp. 1097–1105.
9. Larochelle H., Murray I. The neural autoregressive distribution estimator // AISTATS, Vol. 6, 2011, p. 622.
10. Heess N., Wayne Gr., Silver D. Learning continuous control policies by stochastic value gradients // Advances in Neural Information Processing Systems, 2015. pp. 2926–2934.
11. Wawrzynski P., Tanwani A.K. Autonomous reinforcement learning with experience replay // Neural Networks, Vol. 41, 2013. pp. 156–167.
12. Wawrzynski P. Real-time reinforcement learning by sequential actor–critics and experience replay // Neural Networks, Vol. 22, 2009, pp. 1484–1497.
13. Watkins Chr., Peter D. Q-learning // Machine learning, Vol. 8, 1992. pp. 279–292.
14. Hafner R. Riedmiller Martin Reinforcement learning in feedback control // Machine learning. Vol. 84, 2011, pp. 137–169.
15. Lecun Yann, Bottou L., Bengio Y. Gradient-based learning applied to document recognition // Proceedings of the IEEE, Vol. 86, 1998. pp. 2278–2324.
16. Spielberg P.S. Continuous control with deep reinforcement learning - Deep Deterministic Policy Gradient (DDPG) algorithm implemented in OpenAI Gym environments // Github. (https://github.com/stevenpjg/ddpg-aigym)
17. Schulman J., Mnih V., Kavukcuoglu K., Silver D. Gradient estimation using stochastic computation graphs // Advances in Neural Information Processing Systems, 2015, pp. 3510–3522.
18. Van Seijen Harm, Sutton R. Planning by prioritized sweeping with small backups // Proceedings of The 30th International Conference on Machine Learning, 2013. pp. 361–369.
19. Sun Yi, Ring M., Schmidhuber J. Incremental basis construction from temporal difference error // Proceedings of the 28th International Conference on Machine Learning, 2011, pp. 481–488.
20. Todorov E., Erez T., Tassa Y.. A physics engine for model-based control // Intelligent Robots and Systems, 2012, pp. 5026– 5033.
21. Uhlenbeck G.E. Ornstein L.S. On the theory of the brownian motion // Physical review. Vol. 36, 1930, pp. 47-51.
22. Wawrzynski P. Control policy with autocorrelated noise in reinforcement learning for robotics // International Journal of Machine Learning and Computing, Vol. 5, 2015. pp. 91–95.
23. Guo Xiaoxiao, Satinder S., Lee H. Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning // Advances in Neural Information Processing Systems. Vol. 27, 2014. pp. 3338–3346.
24. Zakharova A., Shklyar A. Basic principles of data visual models consruction, by the example of interactive systems for 3D visualization// Scientific visualization. Vol. 6, 2014. 2. pp. 62–73.
25. Vasev P., Porshnev S., Forghani M., Manakov D., Bakhterev M., Starodubtsev. I. An Experience of Using Cinemascience Format for 3D Scientific Visualization // // Scientific visualization. . Vol. 13, 2021. 4, pp. 127–143.
26. Manakov D., Averbukh V. Verification of visualization // // Scientific visualization. Vol. 8, 2016. 1. pp. 58–94.