A Reward Modulated Spiked Timing Depended Plasticity inspired algorithm applied on a MultiLayer Perceptron

Georgios Giannakas*
Dpt. of Computer Science and Biomedical Informatics,
University of Thessaly
Lamia, Greece
ggiannakas@uth.gr

Maria Sapounaki*
Dpt. of Computer Science and Biomedical Informatics,
University of Thessaly
Lamia, Greece
msapounaki@uth.gr

Vasileios Tsoukas*
Dpt. of Computer Science and Biomedical Informatics,
University of Thessaly
Lamia, Greece
vtsoukas@uth.gr

Athanasios Kakarountas*
Dpt. of Computer Science and Biomedical Informatics,
University of Thessaly
Lamia, Greece
kakarountas@uth.gr

ABSTRACT

The creation of a framework in which traditional Machine Learning and neuromorphic algorithms compete to solve a shared Reinforcement Learning environment is presented in this work. In addition, this configuration allows the exploitation of modern and widely-used Machine Learning libraries. The PyTorch framework is used to investigate the expanded capabilities and potential of training an action-critic network pair comprised of specialised units using a custom learning algorithm. The policy and value networks utilised in this context are fully interconnected MultiLayer Perceptrons. The training procedure employs two distinct algorithms: an algorithm inspired by Reward Modulated Spiked Timing Dependent Plasticity and the conventional Back Propagation technique. A comparative evaluation and analysis of the findings is performed.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; **Reinforcement learning**; **Neural networks**; **Bio-inspired approaches**.

KEYWORDS

Machine Learning, Reward Modulated, Spiked Timing Depended Plasticity, Agent, Reinforcement Learning, PyTorch, autograd, Bio & Nature Inspired Computing

ACM Reference Format:
Georgios Giannakas, Maria Sapounaki, Vasileios Tsoukas, and Athanasios Kakarountas. 2023. A Reward Modulated Spiked Timing Depended Plasticity inspired algorithm applied on a MultiLayer Perceptron. In *27th Pan-Hellenic Conference on Progress in Computing and Informatics (PCI 2023), November 24–26, 2023, Lamia, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3635059.3635066

PCI 2023, November 24–26, 2023, Lamia, Greece

ACM ISBN 979-8-4007-1626-3/23/11.
https://doi.org/10.1145/3635059.3635066

1 INTRODUCTION

Currently, Machine Learning (ML) is widely regarded as the most prevalent type of Artificial Intelligence (AI) due to its ability to address a wide variety of challenging issues. As a result, a plethora of tools, frameworks, and algorithms have been developed to improve the efficacy and efficiency of machine learning applications [34]. Reinforcement learning (RL) is a subfield of machine learning that has gained significant popularity. RL aims to optimise the cumulative reward obtained over a sequence of actions, rather than focusing solely on the immediate rewards associated with individual actions. Consequently, its effectiveness is increasingly notable in fields ranging from technology to health-care to playing games [23]. As a result of the demonstrated effectiveness, companies such as OpenAI and DeepMind have focused their research efforts on advancing reinforcement learning. The publication of Deepmind's 'ATARI paper' in 2013 [20], which investigated the application of RL agents in playing Atari games, sparked considerable intrigue and captivated both the general public and the scientific community. The OpenAI Gym, which was introduced in 2016 [6], offers a standardised application programming interface (API) for reinforcement learning (RL) environments, thereby providing substantial assistance to researchers in this field.

An additional subgenre of machine learning (ML) research that is gaining prominence is the investigation of Spiking Neural Networks (SNNs) [33]. The bioplausibility of these entities surpasses that of their traditional counterparts. Hence, their research can provide a deeper understanding of the brain's low power consumption, particularly in contrast to artificial neural networks. Additionally, this can potentially contribute to the advancement of more efficient brain-computer interfaces. In recent years, there has been notable progress in the field of SNN simulation tools and neuromorphic algorithms, facilitating their study and expanding their potential applications. Additionally, there has been a gradual emergence of SNN hardware. The integration of reinforcement learning (RL) and spiking neural networks (SNN) is gradually attracting interest within the academic community. According to Izhikevich [14], this approach has the potential to provide insights into various processes, such as the influence of dopamine on learning. It can also contribute to the development of robotics that are significantly more energy

efficient [5]. Nevertheless, there has been limited advancement in integrating the breakthroughs from both disciplines.

Therefore, a resulting question is how a setup can be constructed where the comparison of classical ML and neuromorphic algorithms is possible on the same virtual environment while the ability to use modern ML libraries is also retained.

In alignment with the aforementioned perspective, the objective is to leverage the expanding capabilities of Pytorch [25] in a manner that provides adequate adaptability, enabling the training of a network comprising of customized units and a specialized learning algorithm. The model should be able to learn a Farama Foundation Gymnasium RL Environment [6]. Therefore, the learning algorithm should be able to communicate with the environment and use its output –especially the rewards for the successful training of the model-.

To validate the efficacy of the proposed methodology, the implemented configuration is assessed through experimentation involving the training of an intelligent agent to successfully solve the CartPole environment within the Gymnasium framework [6]. The learning algorithm uses Proximal Policy Optimization (PPO) [30]. The generated loss function is handled by an Adam optimizer [16] but instead of applying usual Back Propagation (BP), a variant inspired by Reward Modulated Spiked Timing Depended Plasticity (R-STDP) [15] is applied. For simplicity, the policy and value networks are fully connected MultiLayer Perceptrons (MLPs). Each layer however corresponds to a custom torch.nn.module [25] and therefore it can easily be replaced with any type of layer: classical spiking, recursive or hybrid.

BP since its introduction has been the main and most successful optimization algorithm used in artificial neural networks [10]. It is based on the evaluation, through the application of chain rule, of the error signal's derivatives of each layer with respect to the parameters of the layer's units [28]. However as Weiderman points out BP is non-biologically faithfully process [35]. Biological neurons utilize their own group of learning processes and STDP is perhaps the most well documented and understood. It is expressed through the processes of Long Term Potentiation (LTP) and Long Term Depression (LTD) [4] and provides a biological explanation for Hebbian learning [24].

The remainder of this work is organized as follows. Section 2 provides an overview of related models and applications. Section 3 presents an analytical description of the overall methodology. Section 4 reveals the produced results. Finally Section 5 concludes on the work's findings and discusses future plans.

2 RELATED WORK

The subsequent paragraphs provide a concise overview of prior research endeavours, with a particular emphasis on the utilisation of neuromorphic training algorithms in various applications. Specifically Spike Timing Depended Plasticity (STDP) [4] in Supervised Learning (SL) and RL, through the BP of an error or reward signal respectively. A case of SNN training through BP is also examined.

2.1 STPD in SL

A target-reaching navigation system for a mobile vehicle is proposed by Bing et al. based on a R-STDP learning rule and a Leaky Integrate and Fire SNN [5]. In their approach, the weight modification is the product of an STDP component, an annealing learning rate, a local to every synapse reward signal, and a local for every synapse eligibility trace designed to represent the synaptic efficacy. The reward signal is back-propagated to every synapse and has its local value evaluated based on every weight contribution.

Liu et al. proposed a Supervised STDP as an efficient training method for a SNN classifier. The multilayer network uses Integrate and Fire neurons and is trained to successfully classify the MNIST dataset [7]. In the proposed approach, only the first spike of the spike train and its timing carry significant information. The error signal is normally computed as the Jacobian of the loss function relative to the weights. The STDP components take part in this computation as partial derivatives of the output's first spike timing with respect to the weights.

2.2 STDP in RL

An obstacle avoiding navigation system for a mobile robot is developed by Shim and Li [31]. The proposed model utilizes a one hidden layer feed-forward Leaky Integrate and Fire SNN. The network is trained through additive R-STDP. An eligibility trace is introduced which keeps track of the STDP contribution to the weight change. The reward signal at any given moment is global for the whole network. The resulting weight change is dictated by the product of the current weight, the learning rate, the global reward, and the eligibility trace.

Mozafari et al. [21] categorize their work as RL due to the fact that they also utilize R-STDP. However, the proposed SNN network is being trained on image classification. Therefore, it does not seem to be any sparse rewards involved. In a similar approach as Liu et al. [19], Mozafati et al. consider the timing of the first spike to carry all the significant information. A reward-punishment mechanism is proposed and a comparison is carried out between traditional STDP and R-STDP.

2.3 BP in SNNs

Moreover, closely related to the present research is the work of Esser et al. [9]. The researchers use a training none-neuromorphic network and a deployment neuromorphic network. They apply a BP rule on the training network and then they copy the weights' updates on the deployment network. The training network units' output represents the probability of the corresponding deployment network's neuron to spike. In the present paper the authors, in an almost antithetical to the above publication but also similar in the same instance manner, regard the MLP units' output as the probability that a hypothetical, corresponding SNN's neurons spike.

3 METHODOLOGY

In the following paragraphs a detailed description of the constructed setup is provided. Additionally, the parameters and details of the use case are also mentioned and explained. The developed code can be found uploaded on GitHub [11].

3.1 Experimental Setup and RL Environment

All the simulations are conducted on a DELL XPS 15 9570 [12] equipped with an Intel i7-8750H [29] processor and 16GB of RAM.

The graphics card is an NVIDIA GeForce GTX 1050 Ti [17]. All code is written in Python3. Version 3.11.3 64bit [25] is used during all runs. Pytorch version is 2.0.1 with CUDA 11.8 [25].

The used RL enviroment is CartPole [6] and is part of Gymnasium's Classic Control environments. A cart, moving on a line, tries balancing a non-stable horizontal rod under the effect of gravity. Version used is *CartPole-v1*. The implementation is able to use parallel environments. During all runs the algorithm collects data from four parallel synchronized environments. They are all wrapped together as a single gymnasium.vector.SyncVectorEnv subclass [6].

3.2 Networks and Parameters' Update

For the whole subsection indices i and j denote the corresponding input and output units respectively. Index l corresponds to the number of the example in a mini-batch of a population of size n. Index k denotes the layer level of the corresponding unit. Furthermore, a, z, and w representing the signal after activation –application of the sigmoid function, the signal before activation and the synaptic weight respectively. The above notation remains consistent across all equations contained in the present publication.

An Agent is created for the control of the environment. It consists of two actor-critic MLPs [27] networks. Each of their layers comprises 64 fully connected units. Each unit implements the Perceptron model without bias as shown in equations (1) and (2).

$$z_j^k = \sum_i^n w_{ij} \cdot a_i^{k-1} \tag{1}$$

$$a_j^k = \frac{1}{1 + e^{-z_j^k}} \tag{2}$$

The MLPs are implemented with the Pytorch tensor library [25]. The usual implementation of a layer of perceptrons in Pytorch is a sequence of torch.nn.linear and torch.nn.Sigmoid classes. Instead, the perceptron's layer is implemented as a single subclass of torch.nn.Module. This approach actually allows different possible type of unit layers, linear, nonlinear, recursive, neuromorphic to be defined as separate modules.

Through the extending capabilities of Pytorch autograd [25] custom BP, different custom methods can be implemented for each module, offering the opportunity of comparison between them. A similar method and implementation is followed by Liu et al. [19] in their application of STDP on SL. Autograd normally performs the task of generating a Jacobian matrix of the loss function with respect to the weights. torch.autograd.backward computes the gradients of the given tensors with respect to graph leaves as stated in the documentation [25]. This Jacobian is passed by the torch.optim step method to the optimization algorithm which is the one performing the weight updates. In this work, two different methods are implemented.

The first implementation performs the same operations and chain rule differentiation as the intrinsic autograd does, in order to perform usual BP on a MLP [28]. It is included mainly for comparison reasons but also for additional validation –especially during the early phases of development in order to test that the overall process produces consistent results. The underlying chain rule describing the differentiation between layers, for the case that no biases are present, is presented in equations (3), (4) and (5).

$$\frac{\partial a_i^{kl}}{\partial z_i^{kl}} = a_i^{kl} \cdot (1 - a_i^{kl}) \tag{3}$$

$$\frac{\partial z_i^{kl}}{\partial w_{ij}^k} = \sum_l^n \frac{\partial a_i^{kl}}{\partial z_i^{kl}} a_j^{(k-1)l} \tag{4}$$

$$\frac{\partial z_i^{kl}}{\partial a_j^{(k-1)l}} = \sum_l^n \frac{\partial a_i^{kl}}{\partial z_i^{kl}} w_{ij}^{(k-1)} \tag{5}$$

The second implementation generates a matrix with each element corresponding to an appropriate weight correction, but instead of being generated by chain rule differentiation, a different scheme is applied, inspired by R-STDP. STDP is a bio-plausible procedure that is able to introduce a learning mechanism to the biological neuron [4]. When STDP is combined with biological rewarding systems, like the dopamine system, induces R-STDP. R-STDP can explain RL in living organisms as shown and also modeled by Izhikevich [15].

In this formulation the reward signal for the hidden layers is given by equation (6), where r_j^{kl} is the reward assigned to the *jth* neuron of the *kth* hidden layer. This approach is similar to the one previously, suggested by Bing et al. [5] with the difference that in the present case at this stage a separate reward signal is stored for each training example. The reward signal corresponding to every synaptic weight w_{ij}^k will be $r_j^{kl} = r_j^{kl}$. For the outer layer, the reward signal is calculated by Pytorch's backward method applied to the loss of the Proximal Policy Optimization (PPO) algorithm.

$$r_j^{kl} = \frac{\sum_i |w_{ij}^k| \cdot r_i^{(k+1)}}{\sum_i |w_{ij}^k|} \tag{6}$$

The STDP-inspired component of the weight update signal is calculated on the forward pass as described by equation 9. An assumption is made in regard to that the perceptron's output can be viewed as equivalent to the spike rate of an SNN neuron. Then, a conditioned rate for a synapse of a presynaptic neuron to spike, given that its postsynaptic neuron is not spiking at a given trial, can be approximated by equation (7). It is known that this condition between pre-synaptic and post-synaptic neurons is the required condition for LTP –a state under which the synaptic weight becomes stronger [4]. The opposite function LTD resulting in the weakening of the synaptic strength is known to happen when the postsynaptic neuron spikes, given that the presynaptic neuron does not spike. The conditional rate of this event is approximated by equation (8). In equations (7) and (8) intersections are calculated as the average of their minimum and maximum bounds.

$$\Lambda(LTP)_{ij}^{kl} = \frac{max(0, a_i^{(k+1)l} - a_j^{kl}) + min(a_j^{kl}, 1 - a_i^{(k+1)l})}{2 \cdot (1 - a_i^{(k+1)l}) + epsm} \tag{7}$$

$$\Lambda(LTD)_{ij}^{kl} = \frac{max_(0, a_j^{kl} - a_i^{(k+1)l} + min(a_i^{(k+1)l}, 1 - a_j^{kl})}{2 \cdot (1 - a_j^{kl}) + epsm} \tag{8}$$

In order to calculate the STDP component of the weight correction, a few further assumptions are made. Each step is considered

to have a duration of 1 time unit. The two phenomena LTP and LTD are assumed to obey a Poisson distribution. The difference between the chances of each event happening once in one time-step is assumed here as a valid measure of the change in synaptic weight due to STDP. The resulting formula is presented by equation (9).

$$\Delta w(STDP)_{ij}^{kl} = \Lambda(LTP)_{ij}^{kl} \cdot e^{-\Lambda(LTP)_{ij}^{kl}} - \Lambda(LTD)_{ij}^{kl} \cdot e^{-\Lambda(LTD)_{ij}^{kl}} \tag{9}$$

The final weight correction signal, which is passed from autograd to the optimization algorithm, consists of the mean over all the mini-batches training examples of the element-by-element products of the reward times the STDP component. The relative evaluation is demonstrated in equation (10). A negative sign is added in order for the optimizer to be able to solve this as a minimization problem. *epsm* stands for epsilon machine and it is added in order to avoid division with zero terms.

$$\Delta w_{ij}^{k} = -\sum_{l}^{n} \Delta w(STDP)_{ij}^{kl} \cdot r_{j}^{kl} \tag{10}$$

3.3 Training and Optimization

For the purposes of training, an appropriate variation of PPO [30] is applied. The present implementation of the algorithm is heavily based on the one discussed in the blog-post 'The 37 Implementation Details of Proximal Policy Optimization' and implemented in the corresponding GitHub repository by Huang [13]. A clipped loss version is used with Generalized Advantage Estimation. The clipping factor is set at 0.2. This value is in the range suggested by Andrychowicz et al. [2] for similar setups. In their publication, continuous action spaces are used and harder to solve environments but a similar dependence on the clipping factor is guessed in this work's experiments. The factor discount coefficient set to 0.99 while the Generalized Advantage Estimation's hyper-parameter is set to 0.95. The entropy factor equals 0.01. Finally, the factor of the value function component is set to 0.5.

The algorithm is an on-policy algorithm and therefore improves the policy that is used, through constant evaluation. The evaluation is carried out by a critic separate network while the policy is manifested by an actor-network as a probability distribution of choosing any action for any given state. Andrychowicz et al. find in general a better performance in setups with separate networks than in setups of a shared actor-critic network [2] and these is also the approach followed here. An orthogonal weight initialization is performed as recommended by Engstrom et al.[8] and the actor-network is initialized with a distribution of 0 mean and a low std=0.01 as recommended by Andrychowicz et al. [2].

A variation of Adam algorithm [16] is used for the optimization of the parameters -synaptic weights- of both the value and the policy networks. Adam combines the technique of the adaptive individual rates with the idea of the momentum, where instead of the actual gradients a moving average is used. The implemented version is the one supplied by torch.optimAdam [25]. Furthermore, the stabilizing hyperparameter of the algorithm during all the runs, is set to $1.e-5$ instead the preset $1.e-8$.

4 RESULTS

Performance is often hard to track in RL tasks in comparison to other types of learning. This is mainly due to the highly noisy output. Total average episode reward is often chosen as an appropriate metric [20] and the same approach is followed here. All diagrams are generated with the aid of Tensorboard [1]. Tensorboard offers visualization of the measurements conducted during the workflow. Exponential smoothing by a factor of 0.99 is applied to all the graphs in order to reduce the noise and therefore make them easier to read. All diagrams demonstrate a training period of 1 million total steps of the parallelized environment.

4.1 Vanilla BP

As demonstrated in Figure 1 BP in combination with proven solid training and optimization algorithms provides excellent results and converges extremely fast, even if it does not fully solve the environment. Its consistent behavior provides proof that the present setup is valid and functional. The algorithm achieves best performance for a learning rate of $1.e-3$. Learning rate annealing is also applied, as suggested, for optimal fine-tuning of network's parameters by Sutton and Barto [32]. The result is demonstrated in Figure 1. The Steps Per Second (SPS) rate converges to a value of around 850sps while training the networks.

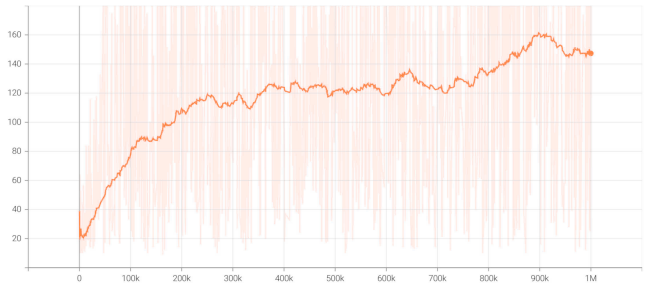

Figure 1: BP Algorithm. Diagram of Episode's Return vs. Global Step

4.2 Hybrid R-STDP Inspired Algorithm

The hybrid algorithm did not manage to solve the environment in the span of 1000000 steps. However, it manages to showcase a clear ability to learn. Under the absence of successful learning, the average of the game stays very close to the value 22 ± 2 –after smoothing- and clearly this is not the case. The best performance, demonstrated in Figure 2, is achieved with a learning rate of $5.6e-3$ and no learning rate annealing. The learning process seems to have been accelerated after about 600k steps. A possible explanation may be that for this learning rule, the networks' initial state is not favorable. SPS converges to approximately 280sps, while training the networks.

5 CONCLUSIONS

One challenge encountered by the researchers pertains to the refinement of parameter initialization and the determination of appropriate hyper-parameter values. RL demonstrates a notable degree

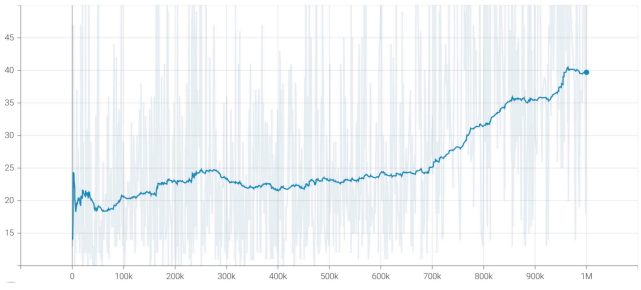

Figure 2: R-STDP Inspired Algorithm. Diagram of Episode's Return vs Global Step

of sensitivity, whereby an inappropriate choice of initialization can potentially disrupt the learning process. Andrychowicz et al. insist on the importance of good policy initialization and good hyperparameters configurations as a complete successful set [2]. In a future project, the additional use of a tuning library like Ray [18] can assist the search for a functional set and achieve an optimization level that in practice is not possible to be attained by guesses and tries.

Early versions of the algorithm prior to the implementation of the custom autograd mechanism were up to two scales of magnitude slower than the later versions that use autograd. To put this in context, prior to the autograd implementation the achieved SPS was consistently under 10sps, while after the autograd implementation, there was an increase in the sps rate, up to the achieved 280sps, after optimization. BP achieved 850sps. Therefore, this proves that normal BP is not only more efficient than the proposed experimental algorithm –for this setup- but also cheaper.

However, there are a few remarks to be made. Spiking is not an intrinsic function of Perceptron. Therefore, LTP and LTD are not intrinsic processes but simulated. This adds to the computational cost. Furthermore, in the case of Perceptron an exact computationally cheap derivative does exist and therefore classic BP is a clear winner. However, in neuromorphic models, exact differentiation is usually impossible [22] and on the contrary spiking is a intrinsic process. Therefore, STDP components can directly be evaluated. Furthermore, with the aid of appropriate traces [26], this can be done in a computationally efficient way, while exact differentiation is not an option.

Another issue is the fact that BP achieves 850sps but this is far behind the approximately 2500sps achieved on the same machine with a similar setup if build-in autograd differentiation is used. The main reason for this is that build-in functions are written in C++ while the custom differentiation functions which are used in this implementation are written in Python3. Therefore, there is a huge speed gap, since C++ is a faster language. Aruoba and Fernández-Villaverde [3] note an approximately 44 times speed difference between the two. However, Libtorch ,a Pytorch version with C++ frontend, offers the option to directly write custom differentiation functions in C++ and add them to torch::autograd, achieving this way similar performance with the build-in functions [25].

The present work manages to demonstrate that a process normally linked to units with temporal dynamics can be applied even in the absence of such behavior. It also hints the high potential of a R-STDP as a process in network training. Moreover, it is the authors' belief that the fusion of different techniques, the combined use of recently developed tools in RL with the application of concepts inspired by neuromorphic algorithms, is still an 'uncharted' field and of great potential for further research. Many tools, such as training algorithms, optimization algorithms, and network structures have been developed for either classic ML or neuromorphic computing. Therefore, hybrid implementations and their capabilities are highly unexplored.

The present approach enables the researchers to easily implement different kinds of setups and apply them on the same problem. The behavior of different neuromorphic spiking models, under variations of R-STDP and gradient-based algorithms, can be examined and their performance can be compared. Similarities and differences between parameter updates performed with different methods and under the use of an appropriate metric might be useful to be examined. The option of implementing autograd extensions directly in C++ is very appealing and a worthwhile endeavor that should be pursued. Also additional algorithms can be examined through further customization of the optim function of Pytorch [25], which is responsible for the applied optimizer. Other neuromorphic hardware-software setup combinations might be considered too.

ACKNOWLEDGMENTS

This research has been financed by the project "ParICT_CENG: Enhancing ICT research infrastructure in Central Greece to enable processing of Big data from sensor stream, multimedia content, and complex mathematical modeling and simulations" (MIS 5047244) which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund)

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Marcin Andrychowicz, Anton Raichuk, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. 2021. WHAT MATTERS FOR ON-POLICY DEEP ACTOR-CRITIC METHODS? A LARGE-SCALE STUDY. (2021).

[3] S. Borağan Aruoba and Jesús Fernández-Villaverde. 2014. *A Comparison of Programming Languages in Economics.* Working Paper 20263. National Bureau of Economic Research. https://doi.org/10.3386/w20263

[4] G. Q. Bi and M. M. Poo. 1998. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J Neurosci* 18, 24 (Dec. 1998), 10464–10472. https://doi.org/10.1523/JNEUROSCI.18-24-10464.1998

[5] Zhenshan Bing, Ivan Baumann, Zhuangyi Jiang, Kai Huang, Caixia Cai, and Alois Knoll. 2019. Supervised Learning in SNN via Reward-Modulated Spike-Timing-Dependent Plasticity for a Target Reaching Vehicle. *Frontiers in Neurorobotics* 13 (2019). https://www.frontiersin.org/articles/10.3389/fnbot.2019.00018

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[7] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[8] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Ma. 2019. IMPLEMENTATION MATTERS IN DEEP POLICY GRADIENTS: A CASE STUDY ON PPO AND TRPO. (2019).

[9] Steven K. Esser, Rathinakumar Appuswamy, Paul Merolla, John V. Arthur, and Dharmendra S. Modha. 2015. Backpropagation for Energy-Efficient Neuromorphic Computing. In *NIPS*. https://api.semanticscholar.org/CorpusID:16801562

[10] P. Frasconi, Marco Gori, and Alberto Tesi. 2001. Successes And Failures Of Backpropagation: A Theoretical Investigation. (02 2001).

[11] Georgios Giannakas. 2023. *Agent capable of solving reinforcement learning environment through the use of training algorithm based on Hebb's Rule.*

[12] Eric Grevstad. 2018. *Dell XPS 15 (9570) Review.* https://www.pcmag.com/reviews/dell-xps-15-9570

[13] Liben Huang and Xiaohui Qu. 2022. Improving traffic signal control operations using proximal policy optimization. *IET Intelligent Transport Systems* (Oct. 2022), n/a–n/a. https://doi.org/10.1049/itr2.12286

[14] Eugene M. Izhikevich. 2007. Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling. *Cerebral Cortex* 17, 10 (01 2007), 2443–2452. https://doi.org/10.1093/cercor/bhl152 arXiv:https://academic.oup.com/cercor/article-pdf/17/10/2443/894946/bhl152.pdf

[15] E. M. Izhikevich. 2007. Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling. *Cerebral Cortex* 17, 10 (Oct. 2007), 2443–2452. https://doi.org/10.1093/cercor/bhl152

[16] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980 arXiv:1412.6980 [cs].

[17] Richard Leadbetter. 2018. *Nvidia GeForce GTX 1050 Ti review. Pascal on a budget.* https://www.eurogamer.net/digitalfoundry-2016-nvidia-geforce-gtx-1050-ti-review-2

[18] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. 2018. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118* (2018).

[19] Fangxin Liu, Wenbo Zhao, Yongbiao Chen, Zongwu Wang, Tao Yang, and Li Jiang. 2021. SSTDP: Supervised Spike Timing Dependent Plasticity for Efficient Spiking Neural Network Training. *Frontiers in Neuroscience* 15 (2021). https://www.frontiersin.org/articles/10.3389/fnins.2021.756876

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. http://arxiv.org/abs/1312.5602 arXiv:1312.5602 [cs].

[21] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. 2018. First-spike based visual categorization using reward-modulated STDP. *IEEE Trans. Neural Netw. Learning Syst.* 29, 12 (Dec. 2018), 6178–6190. https://doi.org/10.1109/TNNLS.2018.2826721 arXiv:1705.09132 [cs, q-bio].

[22] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. 2019. Surrogate Gradient Learning in Spiking Neural Networks. http://arxiv.org/abs/1901.09948 arXiv:1901.09948 [cs, q-bio].

[23] Rui Nian, Jinfeng Liu, and Biao Huang. 2020. A review On reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering* 139 (2020), 106886. https://doi.org/10.1016/j.compchemeng.2020.106886

[24] Joo Park, Sung-Cherl Jung, and Su-Yong Eun. 2014. Long-term Synaptic Plasticity: Circuit Perturbation and Stabilization. *The Korean Journal of Physiology & Pharmacology* 18 (12 2014), 457–60. https://doi.org/10.4196/kjpp.2014.18.6.457

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[26] Fernando M. Quintana, Fernando Perez-Peña, and Pedro L. Galindo. 2022. Bioplausible digital implementation of a reward modulated STDP synapse. *Neural Comput & Applic* 34, 18 (Sept. 2022), 15649–15660. https://doi.org/10.1007/s00521-022-07220-6

[27] F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* 65, 6 (1958), 386–408. https://doi.org/10.1037/h0042519 Place: US Publisher: American Psychological Association.

[28] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (Oct. 1986), 533–536. https://doi.org/10.1038/323533a0 Number: 6088 Publisher: Nature Publishing Group.

[29] Tim Schiesser. 2018. *Intel Core i7-8750H Review: Hexa-core Processor for Laptops Performance on the Go: i7-8750H vs. i7-7700HQ.* https://www.techspot.com/review/1604-intel-core-i7-8750h/

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. http://arxiv.org/abs/1707.06347 arXiv:1707.06347 [cs].

[31] Myung Seok Shim and Peng Li. 2017. Biologically inspired reinforcement learning for mobile robot collision avoidance. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Anchorage, AK, USA, 3098–3105. https://doi.org/10.1109/IJCNN.2017.7966242

[32] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA.

[33] Aboozar Taherkhani, Ammar Belatreche, Yuhua Li, Georgina Cosma, Liam P. Maguire, and T.M. McGinnity. 2020. A review of learning in biologically plausible spiking neural networks. *Neural Networks* 122 (2020), 253–272. https://doi.org/10.1016/j.neunet.2019.09.036

[34] Vasileios Tsoukas, Anargyros Gkogkidis, Aikaterini Kampa, Georgios Spathoulas, and Athanasios Kakarountas. 2022. Enhancing Food Supply Chain Security through the Use of Blockchain and TinyML. *Information* 13, 5 (2022). https://doi.org/10.3390/info13050213

[35] Meagan Wiederman. 2019. Biological Faithfulness is Unnecessary for Machine Learning. *University of Western Ontario Medical Journal* 87 (03 2019), 27–29. https://doi.org/10.5206/uwomj.v87i2.1134