

A Novel Neuromorphic Processors Realization of Spiking Deep Reinforcement Learning for Portfolio Management

Seyyed Amirhossein Saeidi¹, Forouzan Fallah², Soroush Barmaki³, Hamed Farbeh⁴

Department of Computer Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran

¹sahsaeedi@aut.ac.ir, ²forouzan.fallah@aut.ac.ir, ³sr.barmaki@aut.ac.ir, ⁴farbeh@aut.ac.ir

Abstract— The process of constantly reallocating budgets into financial assets, aiming to increase the anticipated return of assets and minimizing the risk, is known as *portfolio management*. Processing speed and energy consumption of portfolio management have become crucial as the complexity of their real-world applications increasingly involves high-dimensional observation and action spaces and environment uncertainty, which their limited onboard resources cannot offset. Emerging neuromorphic chips inspired by the human brain increase processing speed by up to 500 times and reduce power consumption by several orders of magnitude. This paper proposes a spiking deep reinforcement learning (SDRL) algorithm that can predict financial markets based on unpredictable environments and achieve the defined portfolio management goal of profitability and risk reduction. This algorithm is optimized for Intel's Loihi neuromorphic processor and provides 186x and 516x energy consumption reduction compared to a high-end processor and GPU, respectively. In addition, a 1.3x and 2.0x speed-up is observed over the high-end processors and GPUs, respectively. The evaluations are performed on cryptocurrency market benchmark between 2016 and 2021.

Keywords— *neuromorphic computing, deep reinforcement learning, portfolio management, Loihi*

I. INTRODUCTION

The continuous decision-making process of allocating a certain amount of capital to various financial asset products to maximize the risk managing return is called *portfolio management* (PM). Traditional PM methods are divided into 4 sections: *Follow-Winner*, *Follow-the-Loser*, *Pattern matching* and *Meta-Learning*. The performance of these methods was not sufficient and their results depended on the validity of the models in different markets. Because of the complexity of PM process, it causes four major problems: a) Repeat cycles, b) Ignoring real-world factors in the first stages, c) Simplified trading rules by human transactions, and d) Optimizing the inefficient trading system [1].

Deep machine learning methods were developed for financial market transactions in recent years. Although these methods were performing better than traditional methods, their accuracy in predicting the future was not enough due to their dependence on the financial market history. Recently, several methods have been presented based on deep reinforcement learning with relatively acceptable results [2]. One of the best methods is offered by [3]. Using a deep learning network called *Policy*, this method predicts an estimate of the capital distribution between different stocks instead of predicting the future of the financial market. This method has two main drawbacks.

a) Due to the depth of the Policy network, the reinforcement learning process (RL) is limited and has no improvement with increasing learning time and b) Due to the use of deep channels in the Policy network, feature extraction is not done well. As a result, it limits the learning process [2].

With the advancement of technology and the increasing number of IoT devices [4], the interest in solving real-world problems on devices such as smartphones and embedded systems has increased [5]. High processing power consumption of deep reinforcement learning (DRL) methods has made it impossible to use these methods on devices with limited power budget, e.g., smartphones. Another problem with the PM is the high inference time due to the complex models in powerful processors such as the GPU [1]. Reducing the inference time requires shrinking the policy grid, which may be due to the ample space of the issue not being processed and the algorithm not reaching the optimal policy [6]. Energy efficiency is one of the main benefits of Spiking Neural Networks (SNNs). SNNs are an emerging architecture motivated by the human brain. In this architecture, neurons are computed asynchronously and communicate via discrete occasions called spikes [7].

We show how SNNs create a low-power solution on neuromorphic processors such as Intel's Loihi. Most DRL methods perform the learning process based on maximizing reward. Despite the biologically sufficient nature of this learning rule, SNNs suffer from fateful forgetfulness and the absence of policy evaluation, which limits their ability to learn policies under challenging environments [8]. The DRL method uses reply memory to evaluate policies to overcome forgetfulness. This led to the question of the extent to which we can use the possibilities of DRL methods in SNN-based methods. Recent efforts have made it possible to use reply memory alongside the SNN-based method [9].

Recently, a policy-gradient algorithm has been proposed to train an SNN for policy learning. However, this algorithm is for a discrete action space with constraints on the dimensions of the ongoing problem. Another factor that limits the learning of policy by SNNs is high dimensionality.

Interestingly, a key element in the brain associated with efficient calculation is the use of neurons to display information from sensory stimulants to output signals. Early studies on population coding have shown its ability to communicate better incentives, which has led to recent success in training SNNs to reduce high-dimensional complexity [10]. The effectiveness confirmed in population coding opens up prospects for evolving population-based

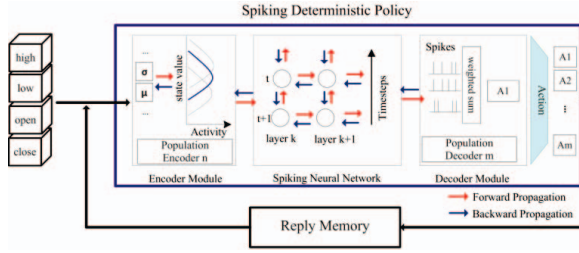


Fig. 1. training of multilayered SDP

SNNs that can lead to optimal policies on ongoing high-dimensional problems [6].

Some SNN-based methods require much time for inference [11], which is particularly challenging in PM for real-time decision-making. One possible way to solve this limitation is direct SNN training using gradient descent techniques such as spatiotemporal backpropagation (STBP). This method provides faster inference while displaying better performance for a broad range of issues [12].

In this paper, we propose a deterministic spiking policy (SDP), a neuromorphic energy-efficient method based on population coding for optimal policy learning. We use the cryptocurrency market to evaluate the proposed method on 11 currencies with maximum volume. The unique feature of the proposed method is the ability to encode ample state space by population neurons with the ability to learn different parts, which effectively increases the network capacity. In this architecture, an SNN network is trained using gradient descent. For SDP training, we introduce a particular mode of STBP that allows SDP training on *Intel's Loihi*.

We evaluate the proposed method by comparing inference time and energy efficiency with the state-of-the-art methods and architectures. The proposed SDP on Loihi shows at least 186 times less energy consumption per inference than running other algorithms on CPU and at least 516x times less energy per inference than running other algorithms on GPU. while learning the optimal policy and performing better in terms of financial base parameters. In addition, the proposed algorithm provides 1.3x speed-up over the GPU and 2.0x speed-up over the CPU.

The remainder of this paper is organized as follows. In Section II, the PM problem is explained, and the proposed solution is described. Section III describes the evaluation methodology and the results. The last Section concludes and continues the future work.

II. PROPOSED METHOD

A. Problem Definition

In portfolio management, our goal is to maximize the profit by keeping trading risk low. We are presenting a deterministic spiking policy (SDP) learning algorithm (Fig. 1) that can learn an optimal policy by taking stock attributes that include $state = \{w_{t-1}, close\ price, high\ price, low\ price, open\ price\}$ and a probability function that represents generates the amount of capital distribution among the assets at moment t . The mode includes the entry price, the lowest price, the highest price, the close price, and the weight of the capital division. On the other hand, the action consists of a weighting to place the capital in different stocks. This algorithm includes an SNN model that

generates an action at time $t + 1$ during a training process by receiving state at moment t .

In this case, the agent's goal is to maximize the final portfolio value (ρ_f) at the end of time $t_f + 1$. The agent has no control over the exact choice of the value ρ_o , and t_f is the length of the portfolio management process interval. Hence, the goal is to maximize the average reward logarithm obtained according to (1) [3].

$$R = \frac{1}{t_f} \sum_{t=1}^{t_f+1} r_t \quad (1)$$

$$= \frac{1}{t_f} \ln \frac{\rho_f}{\rho_o} = \frac{1}{t_f} \sum_{t=1}^{t_f+1} \ln(\mu_t y_t \cdot w_{t-1})$$

Where μ_t is transaction residual factor for real world market simulation, y_t is price relative vector for t^{th} and w_{t-1} trading range are obtained by performing action on $t-1$. The reward function converges the SDP algorithm to achieve the goal during exploration, which facilitates training. For inference, we place the trained SDP algorithm on Loihi to obtain an action for the best portfolio value in the capital market. The training and inference process is described in the following subsections.

B. Spiking deterministic policy (SDP)

The SDP network has an encoder module that uses it to spike real-world data. In this encoder module, we encode the dimensions of the action mode space with the input and output of spiking population neurons. For i^{th} dimension of M -dimensional state S_i $i \in \{1, \dots, M\}$, we create a population of neurons, E_i , to encode the information. E is a Gaussian function with μ and deviation from the ε criterion. The value of μ equals the equal distribution of state space, and ε is a considerable predetermined value of non-zero population activity in all state spaces. The encoder module calculates the activity of the population E in two steps. In the first step, the state values are converted to stimulation strength for each population neuron, A_E , according to (2).

$$A_E = \text{Exp} \left(-\frac{1}{2} \cdot \left(\frac{s - \mu}{\sigma} \right)^2 \right) \quad (2)$$

In the second step, the calculated A_E is used to generate spike E neurons. There are two ways to do this: a) Probability encoding in which the spikes produced by all neurons at each timestep are probabilistically defined by A_E and b) Deterministic encoding where the neurons inside E are simulated as one-step soft-reset Leaky-Integrate-and-Fire (LIF) neurons in which A_E plays the role of presynaptic inputs of neurons. Eq. (3) and (4) govern the dynamic of neurons.

$$V(t) = V(t-1) + A_E \quad (3)$$

$$o_k(t) = 1 \ \& \ V_k(t) = V_k(t) - (1 - \varepsilon) \quad (4)$$

$$\text{if } V_k(t) > 1 - \varepsilon$$

Where k represents the index of each neuron in E and ε is a small constant value. The output of the encoder module is input to the LIF neurons. LIF neurons have been used to build the SDP network (Fig. 1). These neurons update the i^{th} state in time t in two steps. In the first step, spike inputs are converted to synaptic current according to (5).

$$c_i(t) = d_c \cdot c_i(t-1) + \sum_j w_{ij} o_j(t) \quad (5)$$

Where c is the synaptic current, d_c is decay factor for the current, w_{ij} is the binding weight of the j^{th} of the presynaptic neuron, and o_j is a value between 0 and 1, indicating the spike event j of the presynaptic neuron. According to the membrane voltage in (6) and (7), neurons are obtained through synaptic current in the second step. In this way, the neuron only produces a spike if the membrane voltage exceeds the threshold.

$$V_i(t) = d_v \cdot V_i(t-1) + c_i(t) \text{ if } V_i(t-1) < V_{th} \quad (6)$$

$$o_i(t) = 1 \& V_i(t) = 0 \text{ otherwise} \quad (7)$$

Where V is the membrane voltage, d_v is, decay factor for voltage, and V_{th} is the spike fire threshold.

The decoder module converts the generated spike into three possible values for action. In the first step, after the T timestep, the spikes produced in the last layer are added together to obtain the firing rate after T . In the second step, $tempAction$ is obtained by weighting the sum $firingRate$ according to (8) and (9).

$$firingRate^{(i)} = \frac{\text{sum spikes}^{(i)}}{T} \quad (8)$$

$$tempAction^{(i)} = w_d^{(i)} \cdot firingRate^{(i)} + b_d^{(i)} \quad (9)$$

In the third step, action is brought using (10) (Algorithm 1).

$$a^{(i)} = \frac{tempAction^{(i)}}{\sum_{j=1}^N tempAction^{(j)}} \quad (10)$$

Where N is the number of actions.

C. training of SDP with Back Propagation

We train the SDP network using the STBP algorithm to achieve optimal policy. The main STBP is limited to training networks with simple LIF neurons that have only one mode (voltage), but we use presented the STBP to train networks with LIF neurons in both current and voltage methods defined by (5) and (6) [9].

In this way, model training on Loihi with dual-state neurons is possible, because the threshold function that defines spike is immutable. The STBP algorithm demands a pseudo-gradient function to estimate the gradient of a spike. We use the rectangular function which gives us the best performance experimentally [13].

At the end, SDP network action is obtained with forwarding propagation [9]. The parameters for the output i , $i \in \{1, \dots, M\}$ is updated independently according to (11).

$$\begin{aligned} \nabla_{w_d^{(i)}} &= \nabla_{a^{(i)}} L \cdot W_d^{(i)} \cdot firingRate^{(i)} \\ \nabla_{b_d^{(i)}} L &= \nabla_{a^{(i)}} L \cdot W_d^{(i)} \end{aligned} \quad (11)$$

The loss gradient concerning the SDP parameters for each k layer is obtained by adding the Backpropagation gradients from all timesteps according to (12).

$$\nabla_{w^{(k)}} L = \sum_{t=1}^T o^{(t)(k-1)} \cdot \nabla_{c^{(t)(k)}} L, \nabla_{b^{(k)}} L = \sum_{t=1}^T \nabla_{c^{(t)(k)}} L \quad (12)$$

Algorithm 1: Forward propagation through SDP

Output: action $a^{(i)} \in [0, 1] \ i \in \{1, \dots, N\}$, $N = M + 1$, $M =$ number of assets

Require: Maximum timestep T ; Network depth L ;

Require: $w^{(i)}$, $i \in \{1, \dots, L\}$ the weight matrices

Require: $w_d^{(i)}$, $i \in \{1, \dots, N\}$ the weight action matrices

Require: $b^{(i)}$, $i \in \{1, \dots, L\}$ the bias parameters

Require: $w_d^{(i)}$, $i \in \{1, \dots, N\}$ the action bias parameters

Require: $X^{(i)}$, $i \in \{1, \dots, T\}$ the Input spike trains

Require: states s

Spike from Encoder Module = $X = Encoder(s, \mu, \sigma)$;

for $t = 1, \dots, T$ **do**

$o^{(t)(0)} = X^{(t)}$;

for $k = 1, \dots, L$ **do**

$c^{(t)(k)} = d_c \cdot c^{(t-1)(k)} + w^{(k)} o^{(t)(k-1)} + b^{(k)}$;

$v^{(t)(k)} = d_v \cdot v^{(t-1)(k)} \cdot (1 - o^{(t-1)(k)}) + c^{(t)(k)}$;

$o^{(t)(k)} = Threshold(v^{(t)(k)})$;

end

end

$SumSpike = \sum_{t=1}^T o^{(t)(L)}$;

for $t = 1, \dots, N$ **do**

$firingRate^{(i)} = \frac{\text{sum spikes}^{(i)}}{T}$;

$tempAction^{(i)} = e^{w_d^{(i)} \cdot firingRate^{(i)} + b_d^{(i)}}$;

end

for $t = 1, \dots, N$ **do**

$a^{(i)} = \frac{tempAction^{(i)}}{\sum_{j=1}^N tempAction^{(j)}}$;

end

III. EXPERIMENTS

We evaluate the proposed algorithm on cryptocurrency market data and design three experiments for evaluation algorithm. Each test consists of a portfolio of 11 cryptocurrencies with the highest trading volume in the last 30 days before the test data. This data is collected from polonix.com. At the same time, we compare the performance, power consumption, and inference time of the investment strategies presented in this paper with popular investment strategies that have recently been compared. Since cryptocurrencies are traded 24 hours a day, all training and testing sets from 0:00 A-M Coordinated Universal Time (CUT). The duration of each experiment is three years. In each experiment, 80% of the collected data is considered for the training set and 20% for the algorithm test. To investigate the effect of forwarding propagation, we trained the SDP model for value $T=5$. The model hyperparameters used for training can be seen in Table 1.

Table 1 HYPERPARAMETERS FOR TRAINING SDP

Parameter	values
Neuron parameters (Vth, dc, dv)	0.5, 0.5, 0.80
Pseudo-gradient function parameters (a1, a2)	9.0, 0.4
Neurons per hidden layer for SDP	128, 128
Batchsize	128
Learning rate for training SDP	10e-5

A. Discussion

To compare and analyze the proposed solution with other recently proposed algorithms (Table 2), we consider the two parameters, average power consumption and

inference speed, during the execution of the test data. The competitor algorithms run on Corei7-7500 CPU and Tesla K80 GPU processors, and the proposed algorithm runs on the Loihi chip. We use the tools that monitor onboard sensors to measure the power criterion of each device: powerstat for CPU, Nvidia-smi for GPU, and energy probe for Loihi.

Table 2 POWER PERFORMANCE ACROCC HARDWARE

Algorithm	Device	Idle(w)	Dyn(w)	Inf/s	nJ/Inf
DRL-Exp1	CPU	7.98	24.02	2.09	3835.85
	GPU	100.80	29.15	1.23	9165.32
SDP-Exp1	Loihi (T=5)	1.01	0.012	1.04	15.81
DRL-Exp2	CPU	9.09	22.91	1.60	2935.62
	GPU	100.25	29.66	1.09	8119.44
SDP-Exp2	Loihi (T=5)	1.01	0.011	0.82	15.72
DRL-Exp3	CPU	8.69	23.31	2.02	3706.38
	GPU	106.03	24.33	1.07	7998.76
SDP-Exp3	Loihi (T=5)	1.01	0.012	1.01	15.43

Energy cost per inference is acquired by dividing the energy consumed per second by the number of inferences achieved per second. Compared to our proposed algorithm, the algorithms running on CPU and GPU chips have more than 186 times more power consumption during high-speed inference. There is a trade-off for performance cost between SNN's with different timesteps, indicating that the larger the T, the better the performance cost, but the higher the energy cost.

Further reduction of energy costs and allowing portfolio management to be solved on devices such as mobile devices will reduce overall costs such as costs for powerful processing devices. While SNNs are more indicative of reduced energy consumption and sometimes cause a decline in performance [12], [14], [15] this study may be the first to show that an energy-efficient method is more accurate than other portfolio management algorithms during testing. In addition, the SNN's innately noisy representation may have helped to flee the 'bad' local minimum [9].

In sum, all efforts have been made to implement a real-time and energy-efficient solution to the portfolio management problem. In future work, we can research the combination of DNN models with SNN in order to achieve high accuracy in solving the problem by reducing energy consumption.

IV. CONCLUSION AND FUTURE WORK

This paper presents a neuromorphic framework combining lower power consumption and high robustness capacities of SNN networks with DNN learning capacities and tests them on the cryptocurrency market. While recent efforts have focused on architecture and restructuring that architecture we have introduced a new method using SNNs and changed models and their training. By changing the data structure, the training of our proposed model facilitated the training and transfer of important information during learning, which indicates a better feature extraction than other architectures in the past. As a result, an optimal and energy-efficient solution to the portfolio management problem was presented by implementing it on a neuromorphic Loihi processor.

Our method is by 186x more energy-efficient than previous algorithms on DNN compatible low-power devices. This performance advantage includes the asynchronous and event-based calculations provided by SNNs and the ability of our method to train SNN at low T values with negligible performance loss. This energy efficiency improvement enables us to solve the portfolio management problem on mobile and embedded devices with limited onboard resources.

REFERENCES

- [1] L. Conegundes and A. C. M. H. Pereira, "Beating the Stock Market with a Deep Reinforcement Learning Day Trading System," *Proceedings of the International Joint Conference on Neural Networks*, 2020, doi: 10.1109/IJCNN48605.2020.9206938.
- [2] R. Sun, Z. Jiang, and J. Su, "A Deep Residual Shrinkage Neural Network-based Deep Reinforcement Learning Strategy in Financial Portfolio Management," *2021 IEEE 6th International Conference on Big Data Analytics, ICBDA 2021*, vol. 5, no. 1, pp. 76–86, 2021, doi: 10.1109/ICBDA51983.2021.9403210.
- [3] Z. Jiang, D. Xu, and J. Liang, "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem," pp. 1–31, 2017.
- [4] S. Salehi and H. Farbeh, "Energy Optimization in SDN-based Internet of Things," *Journal of Soft Computing and Information Technology*, vol. 10, no. 2, pp. 27–38, 2021.
- [5] F. Sanei and H. Farbeh, "A link adaptation scheme for reliable downlink communications in narrowband IoT," *Microelectronics Journal*, vol. 114, p. 105154, 2021, doi: https://doi.org/10.1016/j.mejo.2021.105154.
- [6] G. Tang, N. Kumar, R. Yoo, and K. P. Michmizos, "Deep Reinforcement Learning with Population-Coded Spiking Neural Network for Continuous Control," no. CoRL, pp. 1–14, 2020.
- [7] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997, doi: 10.1016/S0893-6080(97)00011-7.
- [8] Z. Bing *et al.*, "End to End Learning of Spiking Neural Network Based on R-STDP for a Lane Keeping Vehicle," *Proceedings - IEEE International Conference on Robotics and Automation*, no. June, pp. 4725–4732, 2018, doi: 10.1109/ICRA.2018.8460482.
- [9] G. Tang, N. Kumar, and K. P. Michmizos, "Reinforcement co-learning of deep and spiking neural networks for energy-efficient mapless navigation with neuromorphic hardware," *IEEE International Conference on Intelligent Robots and Systems*, pp. 6090–6097, 2020, doi: 10.1109/IROS45743.2020.9340948.
- [10] Z. Pan, J. Wu, M. Zhang, H. Li, and Y. Chua, "Neural Population Coding for Effective Temporal Classification," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2019-July, 2019, doi: 10.1109/IJCNN.2019.8851858.
- [11] D. Patel, H. Hazan, D. J. Saunders, H. T. Siegelmann, and R. Kozma, "Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game," *Neural Networks*, vol. 120, no. August, pp. 108–115, 2019, doi: 10.1016/j.neunet.2019.08.009.
- [12] G. Tang, A. Shah, and K. P. Michmizos, "Spiking Neural Network on Neuromorphic Hardware for Energy-Efficient Unidimensional SLAM," *IEEE International Conference on Intelligent Robots and Systems*, pp. 4176–4181, 2019, doi: 10.1109/IROS40897.2019.8967864.
- [13] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in Neuroscience*, vol. 12, no. MAY, pp. 1–10, 2018, doi: 10.3389/fnins.2018.00331.
- [14] A. Dorostkar, A. Asad, M. Fathy, M. R. Jahed-Motlagh, and F. Mohammadi, "Low-power heterogeneous uncure architecture for future 3D chip-multiprocessors," *ETRI J.*, vol. 40, no. 6, pp. 759–773, Dec. 2018, doi: 10.4218/etrij.2017-0095.
- [15] A. Dorostkar, A. Asad, M. Fathy, and F. Mohammadi, "Optimization-based reconfigurable approach for low-power 3D chip-multiprocessors," in *2018 IEEE 9th Latin American Symposium on Circuits & Systems (LASCAS)*, Puerto Vallarta, Feb. 2018, pp. 1–4. doi: 10.1109/LASCAS.2018.8399968.