Handling Delay in Reinforcement Learning Caused by Parallel Computations of Neurons

Ivan Anokin
ivan.anokhin@mila.quebec
Mila, Université de Montréal

Rishav
rishav.rishav@mila.quebec
Mila, Ecolé Technologié Superieuré

Stephen Chung
mhc48@cam.ac.uk
University of Cambridge

Irina Rish
irina.rish@mila.quebec
Mila, Université de Montréal, CIFAR AI Chair

Samira Ebrahimi Kahou
samira.ebrahimi.kahou@gmail.com
Mila, University of Calgary, CIFAR AI Chair

Abstract

Biological neural networks operate in parallel, a feature that sets them apart from artificial neural networks and can significantly enhance inference speed. However, this parallelism introduces challenges: when each neuron operates asynchronously with a fixed execution time, an N-layer feed-forward neural network without skip connections experiences a delay of N time-steps. While reducing the number of layers can decrease this delay, it also diminishes the network's expressivity. In this work, we investigate the balance between delay and expressivity in neural networks. In particular, we study different types of skip connections, such as residual connections, projections from every hidden representation to the action space, and projections from the observation to every hidden representation. We evaluate different architectures and show that those with skip connections exhibit strong performance across different neuron execution times, common reinforcement learning algorithms, and various environments, including four Mujoco environments and all MinAtar games. Additionally, we demonstrate that parallel execution of neurons can accelerate inference on standard modern hardware by 6-350%.

1 Introduction

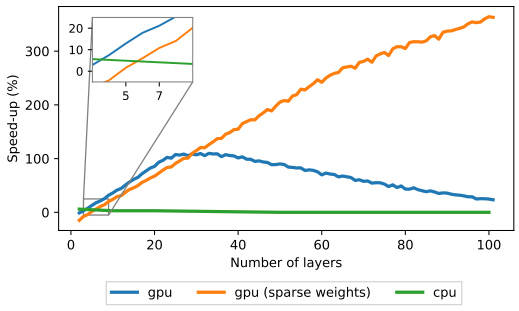

Figure 1: Parallel computations speed-up inference time. Speed-up on GPU is achieved using default Pytorch software and widely accessible Nvidia A100SXM4 GPU with 40 GB memory.

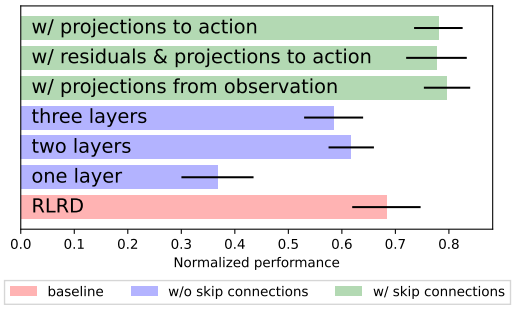

Figure 2: Normalized averaged performance and standard error of agents across Mujoco environments and neuron execution times. Agents w/ skip connections exhibit strong performance.

Inference delay, or the time required to compute the output of a model, can significantly impact performance in many real-world applications. Faster inference times for a given model can express

better policies, as the model processes more observations per second. This is particularly crucial in domains such as robotics, algorithmic trading, real-time gaming and other fields where quick reaction time is essential.

A straightforward approach to speeding up inference is to compute layers in parallel (Carreira et al., 2018; Iuzzolino et al., 2021). Instead of waiting for the entire neural network to complete its forward pass, each layer can begin processing the next input as soon as it produces its output. This method mirrors the asynchronous operation of neurons in the brain (Zeki, 2015). Moreover, neuromorphic computing inherently utilizes such a speed-up.

However, even with parallel execution, a traditional N-layer feed-forward neural network without skip connections still experiences a delay equivalent to the combined execution times of N neurons, as each layer processes the input from the previous time-step. To mitigate this delay, we may reduce the number of layers, but this also limits the network's expressivity. Similarly, adding skip connections can reduce the delay (see Fig. 3) as they do not only shortcut between layers along depth, but also along time, by sending activations forward in time. However, the computational path through these temporal skip connections is shorter and thus offers limited expressivity compared to longer paths through more neurons.

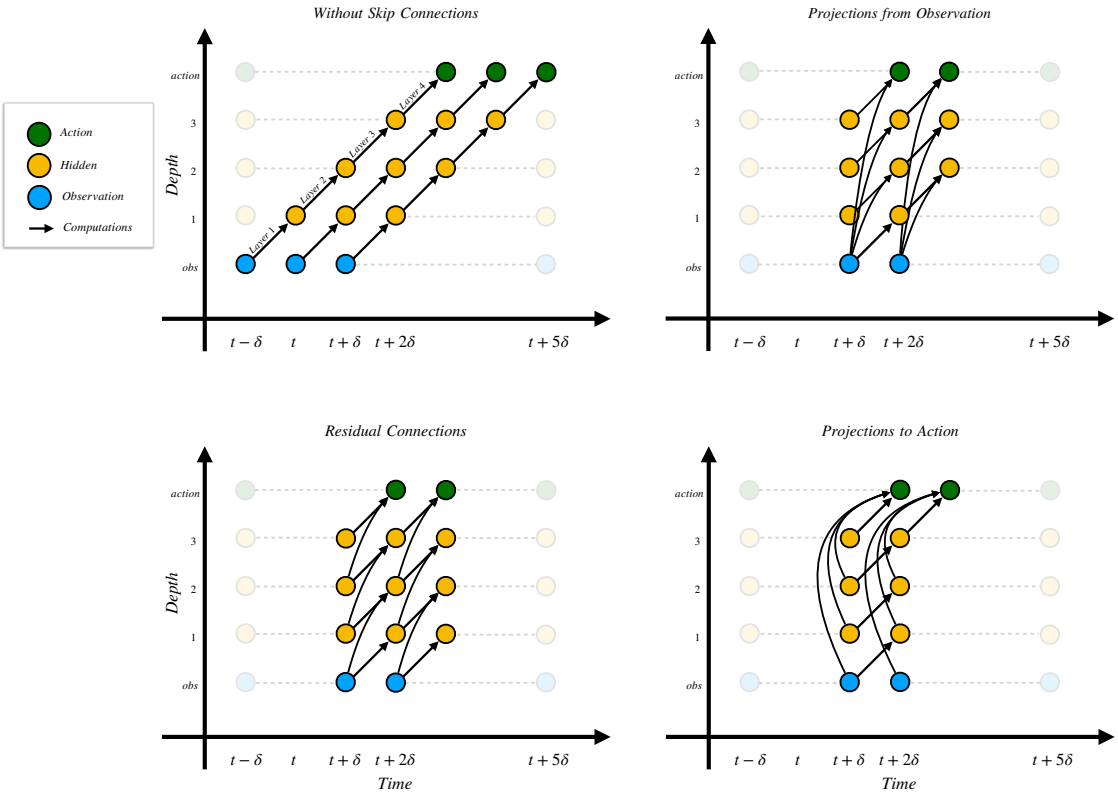

Figure 3: Computation flow of agents with different architectures in a parallel neuron computation framework. δ represents execution time of each neuron. Architectures with skip connections exhibit less delay as they perform shortcuts along time-steps.

We explore the trade-off between delay and network expressivity and investigate various types of skip connections to find an optimal balance. Specifically, we study residual connections, projections from every hidden representation to the action space, and projections from the observation to every hidden representation to handle delay more effectively in a setup with parallel computations.

To summarize, we introduce a new problem in reinforcement learning (RL): training an RL agent in the presence of delay caused by the parallel execution of neurons. We provide a naive solution to train the agent with skip connections using regular backpropagation. Although the parallel layer execution approach and temporal skip connections were proposed before to accelerate predictions on image (Iuzzolino et al., 2021; Fischer et al., 2018) and video (Carreira et al., 2018; Kugele et al., 2020) domains, we are not aware of prior works that explore such ideas in RL. We argue that this problem is more challenging in RL. Unlike image or video prediction, where incorrect predictions can be corrected in subsequent frames, one bad action in RL can lead to the failure of the entire policy as the agent's actions influence the dynamics of the environment.

Our contributions are as follows:

- **Introducing a new problem in RL.** We present the challenge of training an RL agent in the presence of delay caused by parallel execution of neurons (or layers), a problem previously explored in image and video domains but now examined in the context of RL.
- **Providing solution and analysis.** We evaluate different agents and show that in a setup with parallel computation of neurons, agents with skip connections trained with regular backpropagation exhibit strong performance. The agents with skip connections always perform as well as or better than those without skip connections, often significantly outperforming them across different environments, neuron execution times, and RL algorithms. Furthermore, while allowing fast parallel computations, we show that in many cases, the drop in performance is either absent or not significant when compared to the original RL algorithm that assumes an instant forward pass. Our analysis supports the intuition that skip connections enhance policy performance by facilitating faster actions.
- **Estimating speed-up on modern hardware.** While the ideal speed-up is equal to the number of layers with asynchronous neuron execution on a neuromorphic computers, we demonstrate that significant speed-up can also be achieved on common modern hardware, such as GPU, by executing each layer in parallel.

2 Related Work

Parallel execution of neurons (or layers). Parallel processing of information is consistent with popular mathematical models of the human cortex (Tomita et al., 1999; Betti & Gori, 2019; Kubilius et al., 2018; Larkum, 2013), where neurons operate asynchronously. Inspired by this, several attempts have been made to parallelize neural networks, aiming to maximize processing resource utilization and reduce latency. Carreira et al. (2018) introduced parallel video networks that employ parallel layer execution and temporal skip connections, significantly boosting throughput (or frame rate) during inference. Similarly, Iuzzolino et al. (2021) explored this approach for still images, enabling fast "anytime predictions" that improve over time. Additionally, Fischer et al. (2018) provided a theoretical framework for these ideas, and Kugele et al. (2020) applied them for Spiking Neural Networks on image and video domains. Unlike these approaches, we apply these ideas in RL.

Several studies have proposed techniques to handle parallel execution of layers not only during the forward pass but also during the backward pass (by modifying or replacing backpropagation) in both training and inference. Sideways (Malinowski et al., 2020; 2021) achieved this with approximate backpropagation in the video domain. Asynchronous Coagent Networks (Kostas et al., 2020) and Chung (2022) introduced methods where each neural network unit operates independently to maximize its own reward, enabling asynchronous inference and training of neurons. However, Sideways focuses on video data, and both Coagent Networks and Chung (2022) are limited to a small number of neurons, making scalability to larger networks challenging compared to our approach.

Delay in RL. Early works on handling delays in traditional RL settings include (Walsh et al., 2007; Bander & White, 1999; Katsikopoulos & Engelbrecht, 2003; Altman & Nain, 1992). Notably,

(Katsikopoulos & Engelbrecht, 2003) was the first to introduce the notion of a Delayed Markov Decision Process (DMDP). However, their results have not been fully translated into Deep RL.

Recent efforts have addressed delays in Deep RL. Firoiu et al. (2018) tackled delay by predicting future observations, while Wang et al. (2023) trained the critic without delay, augmented state information with historical data, and used self-supervised losses to improve performance on DMDPs. The RLRD method (Bouteiller et al., 2021) further enhanced the critic by augmenting its input with future on-policy actions available due to delay, resulting in more accurate value estimations.

These approaches consider delay as an external factor to the agent. However, our agent inherently introduces delays due to parallel computations, resulting in additional interplay between the agent's architecture and these inherent delays. This allows us to introduce more inductive biases, such as temporal skip connections, into the neural network architecture to effectively mitigate such delays.

3 Background: Delayed Observation Markov Decision Processes

A Markov Decision Process (MDP) is a mathematical framework used to model decision-making in environments where outcomes are partly random and partly under the control of a decision-maker. An MDP is defined by a set of states S, a set of actions $\mathcal{A}$, transition probabilities $\mathcal{P}(s' \mid s, a)$ that describe the probability of moving from state s to state s' given action a, and a reward function $\mathcal{R}(s, a, s')$ that specifies the immediate reward received after transitioning from state s to state s' due to action a.

However, stated in the previous section, in many real-world scenarios, an agent's observations are not instantaneous and there is a delay between taking an action and observing its effect. Thus, to handle such situations, we need to extend the MDP framework to consider delayed observations.

A delayed-observation Markov Decision Process (DOMDP) can be converted to an MDP by estimating the true state of the environment based on the history (delayed observation Markov decision processes (DOMDP) is the special case of partially observable Markov decision process (POMDP)) (Wang et al., 2023). We present the mathematical formulation of DOMDP below.

Consider an Markov decision process (MDP) where there is a delay of d steps in observation. The key components are modified as follows: The extended state space $\mathcal{S}$ is represented as: $\mathcal{S} = S \times \mathcal{A}^d \times S^d$, where S is the original set of states, $\mathcal{A}^d$ is the set of action sequences of length d, and $\mathcal{S}^d$ is the set of state sequences of length d. The action space $\mathcal{A}$ remains unchanged. The transition probability $\mathcal{P}(\tilde{s}' \mid \tilde{s}, a)$ that accounts for the observation delay is given by:

$$\mathcal{P}(\tilde{s}' \mid \tilde{s}, a) = \mathcal{P}(s_{t-d} \mid s_{t-d-1}, a_{t-d-1}) \cdot \prod_{i=1}^{d-1} \mathbb{1}(s'_{t-i} = s_{t-i-1}, a'_{t-i} = a_{t-i-1}),$$

where $\mathbb{1}$ is the indicator function, ensuring historical consistency in states and actions. The reward function $\mathcal{R}(\tilde{s}, a, \tilde{s}')$ is defined as:

$$\mathcal{R}(\tilde{s}, a, \tilde{s}') = \mathcal{R}(s_{t-d}, a_{t-d}, s_{t-d+1}).$$

This function captures the reward received after a delay, based on actions taken d steps prior.

4 Method

In this work, we explore a new problem formulation in RL settings where each neuron's execution takes one time-step, introducing an additional layer of complexity due to inherent delays. This formulation is motivated by the asynchronous firing of neurons in the brain.

To address this new problem, we apply Soft Actor Critic (SAC) (Haarnoja et al., 2018), training a critic without delay and an actor with appropriate delay following suggestions from (Wang et al.,

2023). While doing so we appropriately account for the order of computations inside the actor, as illustrated in Fig. 3, during forward and backward pass while training with vanilla backpropagation.

We also employ skip connections to handle delay more effectively. Skip connections do not only shortcut between layers along depth in parallel neuron execution framework, but also along time, transforming the state-based model into a spatio-temporal one.

The basic structure of our algorithm is presented in Algorithm 1. To begin collecting experience, we initialize the first observation from the environment and set initial hidden activations, depicted in Fig. 3, to zero. While the critic is trained online without delay, our actor is trained within the in-parallel execution framework by unrolling on sub-trajectories sampled from the buffer (with hidden activation set to zero at the first state of a sub-trajectory), allowing all weights to be available for backpropagation.

We model a neuron execution time (δ) relative to the interval between environment observations. For instance, if δ is 2, the environment transitions from o_t to o_{t+2} during the neuron's execution. If the δ is 0.5, two subsequent neurons execute between transitions from o_t to o_{t+1}. It means that with the neuron execution time $\delta > 1$, the actor bases its actions on $o_{t-\delta}$, and the next observation it sees is o_t. To produce some action in between we use a sticky-action mechanism that repeats action $a_{t-\delta}$ for δ steps. On the other hand if δ is less than one, we repeat observations $\lceil 1/\delta \rceil$ times.

Algorithm 1 General Actor-Critic Algorithm with parallel neuron execution.

1: Init an actor and a critic with random parameters.
2: Set initial state to be $s_0, h_0^0, ..., h_0^N$, where h_0^j is activations for layer j at a time step 0.
3: Wrap the environment with sticky actions or repeating observations wrapper if needed based on neural execution time.
4: **for** $t \in 0, \ldots, L$ **do**
5: $a_t, h_{t+1}^0, ..., h_{t+1}^N = Actor(s_t, h_t^0, ..., h_t^N)$ (Query current policy for the next action and next Actor's hidden activations given current observation and hidden activations)
6: Take the action a_t and receive $\{r_t, s_{t+1}\}$ from the environment.
7: Put $\{s_t, a_t, r_t, s_{t+1}\}$ to the buffer.
8: Sample transition $\{s_i, a_i, r_i, s_{i+1}\}$ from the buffer and update the critic on it.
9: Sample sub-trajectory from the buffer $\{s_i, a_i, r_i, s_{i+1}, ..., r_{i+k}, s_{i+k}\}$
10: Init $h_0^0, ..., h_0^N$ and simulate the actor dynamic forward on given sub-trajectory.
11: Update the actor on the last transition of the sub-trajectory (via back-propagation through time if needed)
12: **end for**

We need to highlight that methods that experience delays due to neuron parallelization are not directly comparable with approaches that deal with delay but treat the neural network as a "black box" without considering the processing time of individual neurons. For example, in RLRD (Bouteiller et al., 2021) and other "black-box" approaches, actors always base their actions on observation o_{t-n} with a delay of n, and the next observation an actor receives is o_{t-n+1} which is in contrast with parallel neuron computation framework. Another key difference is that given a neuron execution time of 1, in a parallel computation framework, agents experience a delay of one only if there is a skip-connection from the observation to the action space. For all other layers, the delay will be greater. On one hand, RLRD is advantageous to our framework as it provides the NN with unlimited expressivity to react to delayed observations. On the other hand, our framework employs sticky actions that may enhance performance.

5 Experiments

We perform our main experiments on Mujoco (Todorov et al., 2012) and MiniAtar (Young & Tian, 2019) environments with continuous and discrete action space respectively. We train our agents using SAC for Mujoco and discrete version of SAC for MinAtar. We report mean and one standard

error (SE) in all our plots and experiments if not stated otherwise. Results with PPO (Schulman et al., 2017) can be found in Appendix G.

5.1 Mujoco Results

We present results on four Mujoco environments, HalfCheetah-v4, Walker2d-v4, Ant-v4, and Hopper-v4, varying neuron execution time from one to four for each of them. In all our experiments, unless otherwise specified, we use a three-layer neural network with ReLU activation. We normalize return for every environment and neuron execution time with respect to vanilla SAC performance without delay. Additionally, we report results for the RLRD method, which is available online. We ran RLRD on these environments to obtain statistics. Though it should be noted that RLRD is not directly comparable with this framework (please see Section 4 for the discussion).

Fig. 2 shows that normalized return averaged across four Mujoco environments and neuron execution times is high for the three-layer architectures with skip connections, specifically those with projections to action, with residuals and projections to action and with projections from observation described in Fig. 3. These architectures outperform three baseline agents without skip-connections with different depth, and RLRD method.

Fig. 4 further illustrates this, showing that the actor with projections from observation performs stronger or the same against agents without any skip connections and RLRD across all tested environments and neuron execution times. The plots also indicate that, in many cases, there is no drop in performance when compared to the vanilla SAC without any delay for our with projections from observation agent. For instance, this holds true for Hopper across all neuron execution time, as well as for Walker and Ant with neuron execution times of one and two.

HalfCheetah is the only environment with a significant performance drop compared to the instantaneous actor. We accelerated neuron execution time by a factor of two, adjusting the environment to generate twice as many states per second. This resulted in a normalized performance of 0.87 ± 0.06 for the agent w/ projections from observations, closer matching vanilla SAC performance.

These results suggest the effectiveness of temporal skip connections in handling delay. Additionally, they show that with skip connections and a suitable neuron execution time, an agent in the parallel regime can achieve similar performance to an agent in the instantaneous regime across all Mujoco environments while significantly boosting inference time. Full Mujoco results can be found in Appendix B.

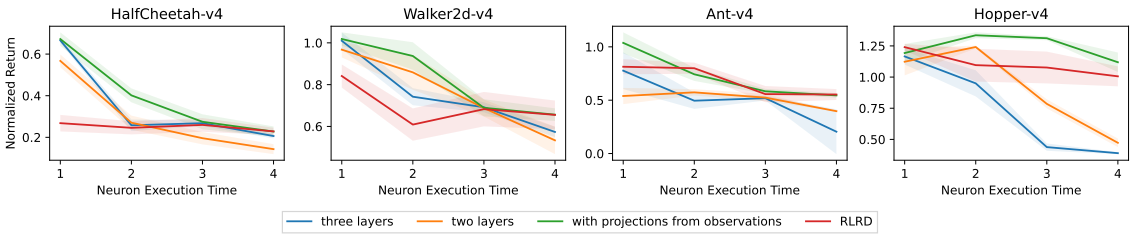

Figure 4: Comparison between w/ projections from observations agent, three and two-layer agents without skip connections and RLRD baseline. The w/ projections from observations agent performs as well or better than other agents. The vanilla SAC w/o delay, which has a normalized performance of one, is omitted from the plots. The shaded area indicates the standard error across 10 seeds.

5.2 MinAtar Results

The results for all MinAtar games are presented in Fig. 5. Consistent with our findings in continuous action spaces, the method with skip connections significantly outperforms those without. Full results can be found in Table 4 in Appendix.

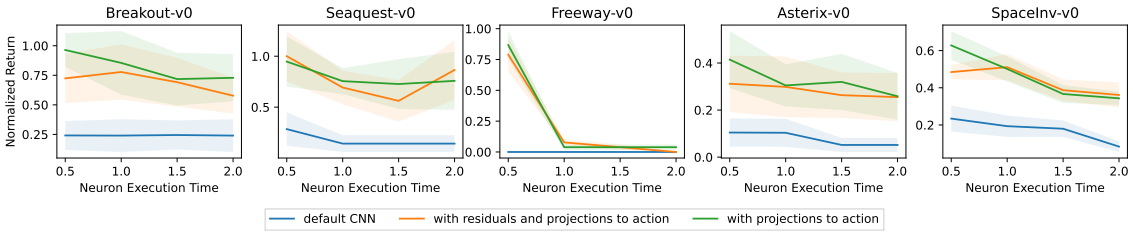

Figure 5: Comparison between different skip connections and default CNN without skip connections. The shaded area indicates the standard error across 3 seeds. The vanilla SAC w/o delay, which has a normalized performance of one, is omitted from the plots.

Our actor employs a 3-layer Convolutional Neural Network (CNN) followed by one fully connected layer. All CNN layers have a kernel size of 3 and $C = \{32, 64, 64\}$ channels, maintaining the same resolution throughout the CNN. The feature volume is then flattened and fed into the fully connected layer for action prediction. For architectures with skip connections, the feature volumes from previous layers are maxpooled, then concatenated and then flattened and subsequently fed to the fully connected layers.

When working with networks incorporating skip connections, we observed that performance deteriorated when directly combining all convolutional features from skip connections by flattening and concatenating them into a single feature volume. We experimented with various methods to combine the features and found that max-pooling all features to a fixed size, followed by flattening and concatenating, yielded the best results. Our Q-network shares the same architecture as the actor.

Qualitative analysis. Qualitative analysis of the agent with projections to action and the default CNN agent (without skip connections) is presented in Fig. 6 for the SpaceInvaders and Breakout games. The analysis shows the agent with skip connections demonstrates better policy compared with the one without, showing quicker and more decisive responses in both Breakout-v0 and SpaceInvaders-v0. In Breakout-v0, it accurately predicts the ball's movement even when distant, while in SpaceInvaders-v0, it effectively dodges alien fire much quicker and effectively, highlighting its superior performance.

5.3 Inference Time Speed-Up

We evaluated the speed-up caused by parallel computations of neurons on various hardware platforms, observing significant improvements in inference time when utilizing a GPU. Fig. 1 illustrates the percentage improvement in inference speed as the number of layers increases across different hardware configurations.

GPU. For GPU setting we measured performance speed-up on a single A100SXM4 GPU with 40 GB memory. The tests were conducted on a deep Multilayer Perceptron (MLP) with a batch size of one and a hidden layer size of 256 for all layers. For parallel computation on the GPU, we naively concatenated all inputs to the layers and combined all layer weights into one large sparse matrix. For agents without skip connections, this matrix has a block-diagonal form. We then used either regular or sparse matrix multiplication to compute the output for each layer. In Fig. 1, these approaches are labeled as GPU and GPU (sparse weights), respectively. The MLP was implemented in PyTorch, utilizing PyTorch's sparse tensor representations and sparse matrix multiplication for the GPU (sparse weights) approach.

Fig. 1 shows that the parallel computations on the GPU accelerate inference time considerably for deep neural networks. Regular matrix multiplication reached its peak performance speed-up around 30 layers, after which the speed-up started to decline; sparse matrix multiplication surpassed regular

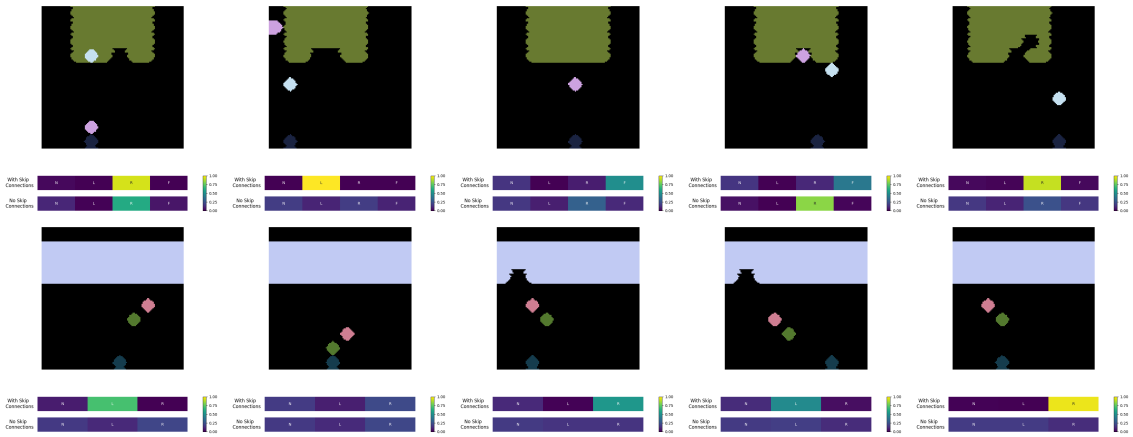

Figure 6: Qualitative analysis for SpaceInvaders-v0 (1st row) and Breakout-v0 (2nd row) (best viewed at 400% zoom). In SpaceInvaders-v0, the pink ball represents the agent's fire, while the white ball represents the alien's fire. In Breakout-v0, the green ball indicates the direction of movement: if the ball is below the paddle, it is moving downward; if it is above, it is moving upward. The action codes are as follows: `N: No move, L: Left, R: Right, F: Fire.` The agent with skip connections (w/ projections to action) performs significantly better in both games, perceiving incoming objects and acting more quickly and decisively. In Breakout-v0, as seen in the 2nd and 5th visuals in the 2nd row, the agent can determine the ball's movement direction with high confidence even when the ball is far away. In contrast, the agent without skip connections struggles due to delayed observations. Similarly, in SpaceInvaders-v0, as seen in the 1st and 5th visuals in the 1st row, the agent with skip connections successfully and confidently dodges the alien's fire. This trend is consistent across all figures.

matrix multiplication at around 30 layers and continued to increase almost linearly with the number of layers achieving 350% speed-up for 100-layers MLP in our test setting.

CPU. We examined the benefits of parallelization on a CPU by running each layer in a separate thread utilizing C++ multi-threading. Our results showed a modest improvement of approximately 6% for a network with 10 layers. However, for networks with more than 20 layers, the gains were marginal (around 0.1-1%). This reduction in speed-up is likely due to the overhead associated with thread synchronization. The CPU used in our tests had 32 cores and 32 GB of RAM. For the CPU tests, we assumed a batch size of 10,000 and a hidden layer dimension of 10,000 in MLP.

We also hypothesize that the limited speed-up on the CPU is due to the use of the Eigen C++ package for matrix multiplication. Eigen already employs multi-threading extensively to optimize matrix operations, ensuring efficient CPU utilization even for sequential matrix multiplication. As a result, additional multi-threading provides only minor speed-up.

6 Conclusion

In conclusion, our work addresses the challenge of delays in reinforcement learning caused by parallel computations of neurons. We found that architectures with temporal skip connections significantly outperform traditional feed-forward networks without skip connections. These architectures demonstrate robust performance across various environments and neuron execution times. Furthermore, we show that with skip connections and a suitable neuron execution time, an agent in the parallel regime can achieve similar performance to an agent in the instantaneous regime while significantly accelerating inference time on GPU, proving particularly beneficial in dynamic settings requiring rapid decision-making.

References

Eitan Altman and Philippe Nain. Closed-loop control with delayed information. *ACM sigmetrics performance evaluation review*, 20(1):193–204, 1992.

James L Bander and CC White. Markov decision processes with noise-corrupted and delayed state observations. *Journal of the Operational Research Society*, 50:660–668, 1999.

Alessandro Betti and Marco Gori. Backprop diffusion is biologically plausible. *arXiv preprint arXiv:1912.04635*, 2019.

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=QFYnKlBJYR.

João Carreira, Viorica Pătrăucean, Laurent Mazare, Andrew Zisserman, and Simon Osindero. Massively parallel video networks. In *European Conference on Computer Vision (ECCV)*. DeepMind and Department of Engineering Science, University of Oxford, 2018.

Stephen Chung. Learning by competition of self-interested reinforcement learning agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 6384–6393, 2022.

Vlad Firoiu, Tina Ju, and Josh Tenenbaum. At human speed: Deep reinforcement learning with action delay, 2018.

Volker Fischer, Jan Köhler, and Thomas Pfeil. The streaming rollout of deep networks-towards fully model-parallel execution. *Advances in Neural Information Processing Systems*, 31, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Michael L. Iuzzolino, Michael C. Mozer, and Samy Bengio. Improving anytime prediction with parallel cascaded networks and a temporal-difference loss. In *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2021.

K.V. Katsikopoulos and S.E. Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE Transactions on Automatic Control*, 48(4):568–574, 2003. doi: 10.1109/TAC.2003.809799.

James Kostas, Chris Nota, and Philip Thomas. Asynchronous coagent networks. In *International Conference on Machine Learning*, pp. 5426–5435. PMLR, 2020.

Jonas Kubilius, Martin Schrimpf, Aran Nayebi, Daniel Bear, Daniel LK Yamins, and James J DiCarlo. Cornet: Modeling the neural mechanisms of core object recognition. *BioRxiv*, pp. 408385, 2018.

A Kugele, T Pfeil, M Pfeiffer, and E Chicca. Efficient processing of spatio-temporal data streams with spiking neural networks front, 2020.

Matthew Larkum. A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex. *Trends in neurosciences*, 36(3):141–151, 2013.

Mateusz Malinowski, Grzegorz Swirszcz, Joao Carreira, and Viorica Patraucean. Sideways: Depth-parallel training of video models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Mateusz Malinowski, Dimitrios Vytiniotis, Grzegorz Swirszcz, Viorica Patraucean, and Joao Carreira. Gradient forward-propagation for large-scale temporal video modelling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9249–9259, 2021.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pp. 5026–5033. IEEE, 2012. ISBN 978-1-4673-1737-5. URL http://dblp.uni-trier.de/db/conf/iros/iros2012.html#TodorovET12.

Hyoe Tomita, Machiko Ohbayashi, Kiyoshi Nakahara, Isao Hasegawa, and Yasushi Miyashita. Top-down signal from prefrontal cortex in executive control of memory retrieval. *Nature*, 401(6754): 699–703, 1999.

Thomas J. Walsh, Ali Nouri, Lihong Li, and Michael L. Littman. Planning and learning in environments with delayed feedback. In Joost N. Kok, Jacek Koronacki, Raomon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron (eds.), *Machine Learning: ECML 2007*, pp. 442–453, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-74958-5.

W. Wang, D. Han, X. Luo, and D. Li. Addressing signal delay in deep reinforcement learning. In *The Twelfth International Conference on Learning Representations*, Virtual Event, October 2023. ICLR.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for more efficient reinforcement learning experiments. *CoRR*, abs/1903.03176, 2019. URL http://arxiv.org/abs/1903.03176.

Semir Zeki. A massively asynchronous, parallel brain. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 370(1668):20140174, 2015.

A Ablation Study and Analysis.

To understand the impact of different architectural choices on the agent's performance and identify potential improvements, we conducted an ablation study. Fig. 7 compares agents with different skip connections. Although no single variant outperforms the others in all cases, the agent with projections from observation demonstrates the most robust performance across Mujoco environments.

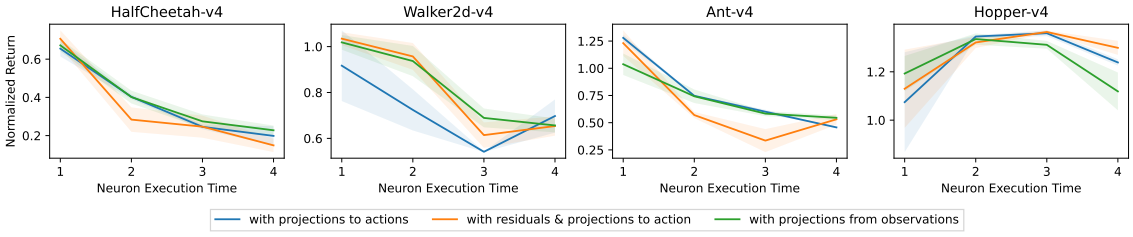

Figure 7: Comparison between different skip connections. The shaded area indicates the standard error across 10 seeds for /w prjections to observation agent and 3 seeds for other agents.

To account for delayed observations in Markov Decision Processes (DOMDP), we augmented observations with the two previous actions or use an LSTM. Augmenting the state with previous actions resulted in better performance for the w/ projections from observation agent, producing an average normalized return of 0.83 ± 0.04 compared to 0.79 ± 0.04 for the non-augmented state space. This suggests potential improvements in our pipeline. However the LSTM did not yield good results in Mujoco environments with delay. Both results are consistent with the findings of Wang et al. (2023).

Additionally, while maintaining the augmented state, we tested all possible forward skip connections between layers. This variant achieved an average normalized return of 0.82 ± 0.04, which is slightly lower but comparable to the best-performing augmentated agent with projections from observation, which had a return of 0.83 ± 0.04.

Furthermore, we varied the number of layers in the augmented agent with projections from observation having a neuron execution time of four across three Mujoco environments, Fig. 8. We observed that all environments benefited from increasing the number of layers from two to three. However, the trends diverged beyond this point. For Walker2d, performance continued to improve with an increasing number of layers. In contrast, for HalfCheetah and Hopper, performance either deteriorated or remained the same.

Distillation. We attempted to distill (employing Dagger (Ross et al., 2011)) a vanilla SAC HalfCheetah policy, which achieved a return of $11,000$, into our agent with projections from observation and a neuron execution time of one. However, we did not observe any performance improvement. The distilled agent's performance was 7590 ± 93, compared to the delayed agent trained with SAC, which achieved 7892 ± 378. This indicates that the performance bottleneck is due to the lack of the agent's expressivity to recover the true state rather than the specific RL algorithm, such as SAC.

Analyzing skip connections. We hypothesize that skip connections can produce good fast actions, and more layers of computations help to refine these fast actions. To confirm it we delete skip connections from observations, skip connections from the first layer representations, and connections from the last second layer representations to action space in the agent with projections to action (Fig. 9). The agent performs very poorly without the first two skip-connections, but it can produce some non-zero return if we delete connections from the last layer, which supports our hypothesis.

Asynchronous neuron computation. Though we simulate parallel executions of neurons during inference and training time, the executions were globally synchronized, in as sense that all neurons finish and start new executions at the same time. To test whether it would be possible to train capable agents without the global synchronization we apply dropout as a proxy to asynchronous execution in every hidden layer during the training and inference stage to our with projection to the action agent, Fig. 10. One can see that the agent is quite robust to a large amount of dropout, and the performance starts to deteriorate if the probability of "not updating a neuron" becomes more than 40%.

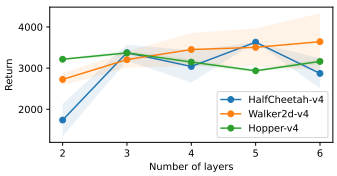

Figure 8: Varying number of layers in w/ projections from observation agent with neuron execution time of four. The shaded area indicates the standard error.

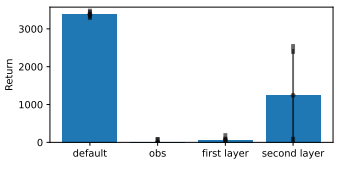

Figure 9: Removing different connections in the w/ projections to action actor in Ant-v4 with neuron execution time of four. Mean and one SD across 100 episodes are reported.

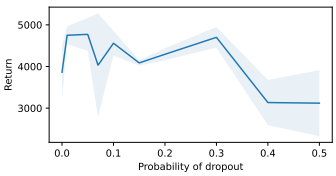

Figure 10: Average return with one SE across 3 seeds vs different amount of dropout for w/ projections from observation agent on HalfCheetah with neuron execution time of two.

B Full Mujoco Results

Full Mujoco results for all considered environments and neuron execution times can be found in Tables 1 and 2. SAC-sticky-2 and SAC-sticky-3 refer to the vanilla version of SAC with sticky

actions, SAC-sticky-2 and SAC-sticky-3 always repeat an action in the environment two or three times respectively.

Table 1: Mujoco average returns after 1mln states of training for the four selected environments. The results are averaged across ten seeds for two layers, three layers and w/ projections from observation agents, across 5 seeds for RLRD and the rest of results use 3 seeds. Mean and one standard deviation are reported. We take mean action as a policy during evaluation stage as we notice it may significantly boost the performance of the delayed actor. The same results but with the policy that samples actions from the normal distribution instead of taking the mean can be found in the Table 2. RLRD (Bouteiller et al., 2021) is a baseline.

	Halfcheetah-v4	**Walker2d-v4**	**Ant-v4**	**Hopper-v4**
SAC	**11739 ± 490**	4415 ± 320	3595 ± 1779	2672 ± 801
SAC-sticky-2	8626 ± 523	**4670 ± 221**	**4102 ± 1228**	**3520 ± 140**
SAC-sticky-3	8168 ± 618	3763 ± 582	2625 ± 796	3517 ± 94
	neuron execution time of 1			
RLRD (delay of 1)	3147 ± 1044	3714 ± 547	2924 ± 568	**3314 ± 157**
one layer	5086 ± 1147	1209 ± 1304	1043 ± 744	759 ± 173
two layers	6660 ± 1081	4271 ± 232	1938 ± 865	3001 ± 971
three layers	7814 ± 412	4459 ± 528	2792 ± 1753	**3115 ± 548**
w/ projections to action	7690 ± 831	4048 ± 1179	**4599 ± 157**	2871 ± 956
w/ residuals & projections to action	**8295 ± 904**	**4567 ± 213**	4425 ± 734	3019 ± 749
w/ projections from observations	7892 ± 1198	**4496 ± 445**	3728 ± 1123	**3187 ± 615**
	neuron execution time of 2			
RLRD (delay of 2)	2884 ± 831	2688 ± 764	**2874 ± 422**	2928 ± 787
one layer	1846 ± 880	1535 ± 466	1783 ± 323	1980 ± 1161
two layers	3173 ± 1263	3791 ± 900	2061 ± 264	3317 ± 99
three layers	3027 ± 1041	3277 ± 571	1780 ± 827	2538 ± 888
w/ projections to action	**4729 ± 252**	3197 ± 686	**2685 ± 33**	**3597 ± 25**
w/ residuals & projections to action	3330 ± 1301	**4226 ± 447**	2049 ± 176	**3531 ± 106**
w/ projections from observations	**4715 ± 1241**	4137 ± 920	**2669 ± 592**	**3569 ± 199**
	neuron execution time of 3			
RLRD (delay of 3)	2150 ± 547	**3014 ± 814**	**2002 ± 611**	2877 ± 763
one layer	1293 ± 1272	2157 ± 268	1597 ± 191	1150 ± 919
two layers	2299 ± 1160	**3037 ± 441**	1884 ± 279	2098 ± 337
three layers	**3145 ± 524**	**3054 ± 352**	1865 ± 137	1171 ± 261
w/ projections to action	2886 ± 254	2391 ± 10	**2160 ± 94**	**3633 ± 56**
w/ residuals & projections to action	2897 ± 1176	2711 ± 425	1205 ± 658	**3647 ± 26**
w/ projections from observations	**3224 ± 1339**	**3043 ± 574**	**2097 ± 188**	3505 ± 137
	neuron execution time of 4			
RLRD (delay of 4)	**2682 ± 466**	2893 ± 678	**1988 ± 412**	2688 ± 479
one layer	1284 ± 795	1975 ± 318	1132 ± 971	1372 ± 476
two layers	1681 ± 933	2355 ± 582	1427 ± 809	1263 ± 302
three layers	2421 ± 442	2532 ± 362	735 ± 1512	1041 ± 68
w/ projections to action	2330 ± 1029	**3079 ± 452**	1642 ± 51	**3310 ± 61**
w/ residuals & projections to action	1748 ± 717	2886 ± 322	1909 ± 251	**3472 ± 135**
w/ projections from observations	**2674 ± 911**	2898 ± 422	**1959 ± 157**	2990 ± 661

Table 2: The same as Table. 1, but a policy samples actions from normal distribution during evaluation. The relative order of agent performance stay approximately the same.

	Halfcheetah-v4	**Walker2d-v4**	**Ant-v4**	**Hopper-v4**
SAC	11108 ± 432	4665 ± 452	4284 ± 871	2737 ± 813
	neuron execution time of 1			
one layer	3675 ± 458	1084 ± 993	1518 ± 919	832 ± 320
two layers	6137 ± 926	4010 ± 575	1989 ± 1061	2969 ± 784
three layers	7120 ± 388	4339 ± 510	2453 ± 1527	$\mathbf{3012 \pm 393}$
w/ projections to action	7051 ± 947	4089 ± 1036	$\mathbf{4339 \pm 263}$	2687 ± 771
w/ residuals & projections to action	7785 ± 762	$\mathbf{4299 \pm 252}$	4001 ± 733	2888 ± 682
w/ projections from observation	$\mathbf{7305 \pm 1123}$	$\mathbf{4496 \pm 376}$	3414 ± 1001	$\mathbf{3012 \pm 524}$
	neuron execution time of 2			
one layer	1635 ± 711	1859 ± 896	1487 ± 181	1754 ± 1015
two layers	2787 ± 1073	3652 ± 806	1994 ± 241	3156 ± 242
three layers	2728 ± 896	3234 ± 550	1684 ± 762	2242 ± 827
w/ projections to action	$\mathbf{4279 \pm 261}$	3117 ± 584	$\mathbf{2444 \pm 98}$	$\mathbf{3540 \pm 93}$
w/ residuals & projections to action	3059 ± 1188	$\mathbf{4182 \pm 426}$	1862 ± 84	$\mathbf{3490 \pm 154}$
w/ projections from observation	$\mathbf{4241 \pm 1095}$	$\mathbf{4043 \pm 858}$	$\mathbf{2426 \pm 538}$	$\mathbf{3522 \pm 187}$
	neuron execution time of 3			
one layer	1306 ± 912	2159 ± 284	1501 ± 948	1024 ± 875
two layers	1986 ± 1062	2888 ± 412	1538 ± 667	2033 ± 343
three layers	$\mathbf{2779 \pm 417}$	$\mathbf{2998 \pm 351}$	1640 ± 543	1059 ± 270
w/ projections to action	2699 ± 170	2377 ± 18	$\mathbf{2066 \pm 88}$	$\mathbf{3591 \pm 58}$
w/ residuals & projections to action	2609 ± 1013	2702 ± 422	911 ± 606	$\mathbf{3523 \pm 5}$
w/ projections from observation	$\mathbf{2926 \pm 1219}$	$\mathbf{3005 \pm 549}$	1896 ± 334	3430 ± 129
	neuron execution time of 4			
one layer	1168 ± 726	1876 ± 401	1074 ± 879	1327 ± 460
two layers	1501 ± 828	2393 ± 381	1371 ± 871	1197 ± 297
three layers	2138 ± 385	2493 ± 351	988 ± 992	1026 ± 39
w/ projections to action	2084 ± 961	$\mathbf{3041 \pm 435}$	1559 ± 8	$\mathbf{3234 \pm 68}$
w/ residuals & projections to action	1548 ± 625	2821 ± 322	1771 ± 175	$\mathbf{3250 \pm 54}$
w/ projections from observation	$\mathbf{2385 \pm 795}$	2845 ± 381	$\mathbf{1854 \pm 149}$	2910 ± 640

Table 3

Table 4: Best average returns after 5 million training steps on MinAtar games. Results are averaged across three seeds and standard deviation is reported.

	Breakout-v0	**Seaquest-v0**	**Freeway-v0**	**Asterix-v0**	**SpaceInv-v0**
SAC	8.33 ± 2.6	7.03 ± 2.06	25.76 ± 5.06	19.33 ± 5.0	60.0 ± 20.0
neuron execution time of 0.5					
default CNN	2.01 ± 1.76	2.0 ± 2.0	0 ± 0	2.02 ± 2.0	14.06 ± 7.33
w/ projections to action	8.03 ± 2.06	6.66 ± 3.0	22.33 ± 5.06	8.01 ± 4.06	37.66 ± 8.0
w/ res & proj to action	6.03 ± 3.0	7.03 ± 3.00	20.33 ± 6.06	6.03 ± 4.0	29.06 ± 4.0
neuron execution time of 1					
default CNN	2.0 ± 2.0	1.0 ± 1.0	0.0 ± 0.0	2.0 ± 2.0	11.61 ± 5.98
w/ projections to action	7.12 ± 3.88	5.31 ± 1.55	1.0 ± 1.0	5.89 ± 3.0	30.07 ± 7.0
w/ res & proj to action	6.48 ± 3.38	4.87 ± 2.0	2.0 ± 1.0	5.77 ± 4.32	30.65 ± 7.34
neuron execution time of 1.5					
default CNN	2.04 ± 1.78	1.0 ± 1.0	0.0 ± 0.0	1.0 ± 1.0	10.79 ± 4.58
w/ projections to action	5.98 ± 3.22	5.11 ± 2.98	1.0 ± 1.0	6.18 ± 4.0	22 ± 5
w/ res & proj to action	5.76 ± 2.97	3.95 ± 2.48	1.0 ± 1.0	5.09 ± 3.24	23.22 ± 6.0
neuron execution time of 2					
default CNN	2.0 ± 2.0	1.0 ± 1.0	0.0 ± 0.0	1.0 ± 1.0	5.02 ± 3.0
w/ projections to action	6.07 ± 2.88	5.33 ± 3.47	1.0 ± 1.0	5.0 ± 3.32	20.59 ± 3.71
w/ res & proj to action	4.81 ± 2.2	6.08 ± 3.57	0.0 ± 0.0	4.93 ± 3.44	21.68 ± 6.95

C Full MinAtar Results

Full Mujoco results for all considered environments and neuron execution times can be found in Tables 4. Skip connections consistently improve performance across all cases for both architectures that utilize skip connections, with projections to action, and with residuals and projections to action, over the default CNN architecture.

D Limitations

We execute each neuron in parallel, though not asynchronously. While we apply dropout (see Fig. 10) as a proxy for asynchronous neuron updates, true asynchronous updates are more challenging to model. Additionally, since neuromorphic chips are not widely available, our immediate impact on the field may be limited.

E Training & Hardware

We used A100SXM4 GPU for training all our methods. We use the same GPU for testing our methods as well. It took us approximately 6 hours per seed to train a MinAtar experiment for 5 million steps while it took 7 hours per seed to train one MuJoCo experiment on the same GPU for 1 million steps.

F Hyperparameters Used in Experiments

The hyperparameters used in the main experiments on SAC Mujoco and MinAtar can be found in Table 5.

Table 5: Hyperparameters used in experiments.

Parameter	Value
SAC Mujoco	
Discount rate γ	0.99
Policy frequency	2
Target network frequency	1
Target smoothing coefficient	0.005
Policy learning rate	3e-4
Q-function learning rate	1e-3
Optimizer	Adam
Adam beta	(0.9, 0.999)
Adam epsilon	1e-8
Replay buffer size	1,000,000
Batch size	256
Learning starts	10,000
Entropy regularization	Auto-tuned
Target entropy scale	1
SAC MinAtar	
Discount rate γ	0.99
Policy frequency	32
Target network frequency	8000
Target smoothing coefficient	1
Policy learning rate	3e-4
Q-function learning rate	3e-4
Optimizer	Adam
Adam beta	(0.9, 0.999)
Adam epsilon	1e-4
Replay buffer size	1,000,000
Batch size	64
Learning starts	20,000
Entropy regularization	Auto-tuned
Target entropy scale	0.445

G PPO Learning Curves

PPO learning curves are present in Fig. 11. Agent with projections to action reaches almost the same level of performance in all four Mujoco environments as vanilla PPO agent without delay except Ant-v4 where it completely fails. The agent without skip connections performs much worse. The neural execution time is one. The shaded area indicates the standard error across 3 seeds.

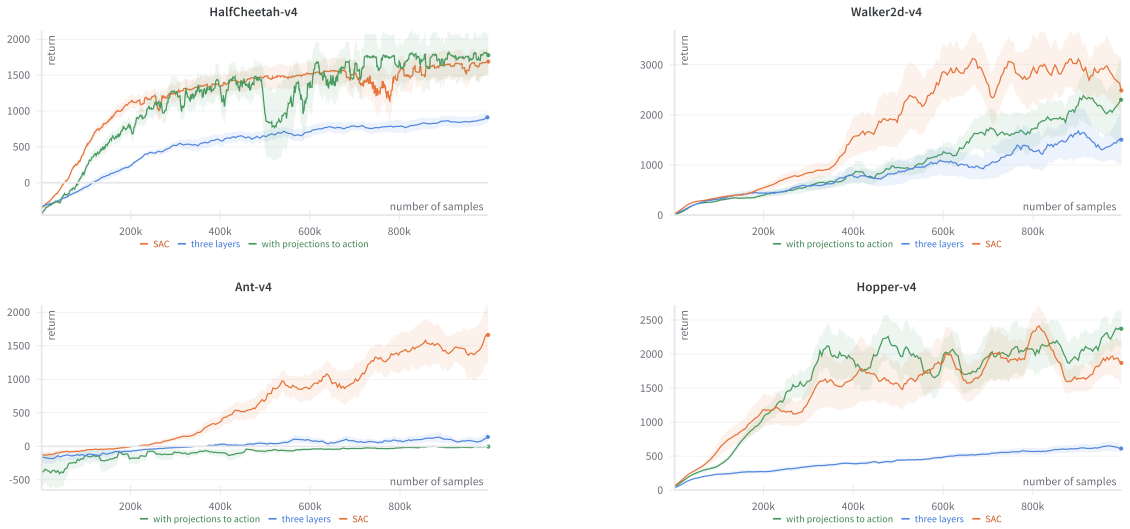

Figure 11: Results of PPO on various Mujoco environments. Agent with projections to action agent reaches almost the same level of performance in all Mujoco environments except Ant-v4 where it completely fails.