The Cart-Pole Application as a Benchmark for Neuromorphic Computing

James S. Plank *, Charles P. Rizzo, Christopher A. White, Catherine D. Schuman

Posted Date: 6 December 2024

doi: 10.20944/preprints202412.0532.v1

Keywords: neuromorphic computing; cart-pole; benchmark; spiking neural networks; genetic algorithms

 doi:10.20944/preprints202412.0532.v1

Article

The Cart-Pole Application as a Benchmark for Neuromorphic Computing

James S. Plank *,†,‡, Charles P. Rizzo †,‡, Christopher A. White †,‡ and Catherine D. Schuman †,‡

Department of Electrical Engineering and Computer Science, University of Tennessee
* Correspondence: jplank@utk.edu
† Current address: 401 Min Kao Building, University of Tennessee, Knoxville, TN 37996.
‡ These authors contributed equally to this work.

Abstract: The cart-pole application is a well-known control application that is often used to illustrate reinforcement learning algorithms with conventional neural networks. An implementation of the application from OpenAI Gym is ubiquitous and popular. In this paper, we explore using this application as a benchmark for spiking neural networks. We propose four parameter settings that scale the application in difficulty, in particular beyond the default parameter settings which do not pose a difficult test for AI agents. We propose achievement levels for AI agents that are trained on these settings. Next, we perform an experiment that employs the benchmark and its difficulty levels to evaluate the effectiveness of eight neuroprocessor settings on success with the application. Finally, we perform a detailed examination of eight example networks from this experiment, that achieve our goals on the difficulty levels, and comment on features that enable them to be successful. Our goal is to help researchers in neuromorphic computing to utilize the cart-pole application as an effective benchmark.

Keywords: neuromorphic computing; cart-pole; benchmark; spiking neural networks; genetic algorithms

1. Introduction

Control applications compose a significant application domain in the realm of machine learning and neural networks. With classic neural networks, they present a challenge because of the time component, which is not an elemental feature of a classic neural network. They present a moving target — unlike image classification, where the features and the classifications are unchanging, and a simple loss function can be leveraged for training, in a control application the agent's actions affect subsequent observations. Simple backpropagation is insufficient as a training technique.

Control applications are a natural fit for neuromorphic processors [1]. These employ spiking neural networks (SNN's), in which time is a fundamental element, and memory is co-located with processing. As with most applications for neuromorphic computing, the main challenge is in designing or training the SNN. Nonetheless, SNN's have been applied successfully to a variety of control problems [2–4]. Additionally, solving control problems in a real-world setting with hardware often requires low size, weight, and/or power solutions, as well as certain latency requirements. As such, control applications are an interesting application from a hardware perspective as well.

Benchmarks for control applications are inherently more challenging than for classification applications, because the neuromorphic agent must interact with a dynamic system, rather than assign classification labels to a data set. Even a precise mathematical definition of the system may be insufficient to describe a control application, because the definition must be rendered to a computational dynamic system. If the system is real (e.g., a real robotic system), then there are myriad system parameters that are not repeatable from run to run. If the system is simulated, then differing processor architectures and computational environments can also hamper repeatability.

Regardless, benchmarking is essential for a field of study, for without benchmarking, comparison of neuromorphic processors, systems and training methodologies is impossible.

The "cart-pole" problem is a classic control problem [5,6]. A wheeled cart is placed on a one-dimensional track. A pole that may "tip" to the left or the right is balanced on the cart. At periodic

 doi:10.20944/preprints202412.0532.v1

intervals (typically 1/50s), information about the cart and the pole are communicated to an agent that may respond with one of two actions: push the cart left or push the cart right. The agent's objective is to keep the cart within the bounds of the track, and to keep the pole balanced so that its angle to vertical is below a given threshold.

The cart-pole problem is attractive as a benchmark for many reasons. Chief among them are simplicity and availability. Agents receive just four observations: cart position (x), cart velocity (dx), pole angle (θ), pole angle velocity $(d\theta)$. At each decision point, they only have two actions: push left and push right. The equations that govern this system are rudimentary and well-documented [6,7]. Moreover, a widely available implementation exists as part of the OpenAI Gym project [8]. It is one of the first applications employed by mutiple tutorials on reinforcement learning for classic neural networks (e.g., [9,10]), and it is employed as an *ad hoc* benchmark in hundreds of papers on reinforcement learning (see Section 10 below for a sampling).

The benchmark is *ad hoc*, because the metrics of success differ from project to project. For example, in one of the earliest applications of neural networks to the problem, Anderson's goal is 500,000 timesteps without failure [6]. In Kumar's tutorial, the goal is to balance the cart for an average of at least 195 timesteps for 100 consecutive episodes [9]. Ding *et al* report that their spiking neural network achieves up to 450 timesteps [11].

In this work, we present four benchmarks employing the cart-pole application. We present them in detail below, but we summarize them here:

- *Easy*: The standard cart-pole problem with a mission time of up to 15,000 timesteps, averaged over 1,000 random starting positions (i.e., 1,000 different episodes).
- *Medium*: The *Easy* problem with an additional action: "do-nothing". The agent must employ "do-nothing" for at least 75 percent of its actions. This is more challenging and also more attractive, as agents focus more on putting the pole into a naturally balanced state, rather than keeping it constantly perturbed.
- *Hard*: The *Medium* problem with only two observations instead of the standard four – just the position of the cart and the angle of the pole. There are no velocities. Unlike the *Medium* benchmark, there are no restrictions on the actions chosen. This is a very challenging benchmark, because it encourages the agent to maintain some system state in its "memory."
- *Hardest*: This is the same as the *Hard* problem, but the only actions are to push the cart left or right.

For each benchmark, we derive goals for the performance of AI agents. We then perform an experiment to determine the most effective techniques for encoding the cart-pole observations into spikes for SNN agents. The first three benchmarks leverage a "Flip-flop" encoder, where positive and negative values each have their own neurons, and the number of spikes is proportional to the absolute value of the observation. The *Hardest* benchmark is most successfully solved by an "Argyle" encoder, where each observation results in exactly nine spikes, distributed to two of four input neurons.

We next employ the benchmark to evaluate the effectiveness of eight parameter settings of the RISP neuroprocessor. For the *Easy* benchmark, the simplest parameter setting of RISP trains effective SNN's, but as the benchmarks progress in difficulty, richer neuroprocessor parameters are required. The experiment also highlights how one must pay attention to the interplay between the neuroprocessor parameters (in this case, synaptic delay) and the application in order to achieve good training.

Finally, for each benchmark, we present and discuss two example SNN's, trained in the previous experiment, that achieve the benchmark's goal. For each, we highlight properties and features of the networks that lead to their success.

2. Contributions of This Paper

In this paper, we make the following contributions:

1. We codify four ways in which to employ the cart-pole problem as a benchmark for AI agents, along with target goals for agents to achieve on each benchmark.

 doi:10.20944/preprints202412.0532.v1

2. We perform an experiment to determine the best way to encode observations into spikes when the AI agent is a neuroprocessor running a spiking neural network.
3. We perform an experiment to demonstrate how various neuroprocessor parameter settings effect the performance of the neuroprocessor on each of the benchmarks. Along with this, we highlight how to improve the performance by paying attention to how parameter settings interact with the way the observations are encoded.
4. We highlight features of SNN's that are useful in neural network design. For example, it is sometimes advantageous to arrange for multiple spikes to arrive at a neuron at different times, so that their effect is additive. Other times, it is advantageous to arrange for the spikes to arrive at the same time, so that their effect is not additive. In such as way, simple SNN constructs may implement, for example, a *max()* function that helps solve the benchmark.

3. The Cart-Pole Problem

The cart-pole problem is described in multiple places: [5–8]. Accordingly, we do not redescribe it here. The following parameter settings are standard with respect to OpenAI gym:

- Observations: Cart position (x), cart velocity (dx), pole angle (θ), pole velocity $(d\theta)$.
- Track extent: -2.4 to 2.4 meters. If the cart moves beyond this, it is a failure.
- Pole angle: -0.2095 to 0.2095 radians (12 degrees). If the pole falls beyond these angles, it is a failure.
- Timestep duration: $1/50$ second.
- Reward: +1 for each non-failed timestep.
- Actions: push left and push right.

The following parameters are non-standard, and are used to make the problem more challenging:

- Mission length: 15,000 timesteps, or five simulated minutes. This is much longer than OpenAI's length of 200 timesteps (4 seconds) in v0, and 500 timesteps (10 seconds) in v1.
- Starting cart position: between -1.2 and 1.2 on the track. OpenAI's state is between -0.05 and 0.05.
- Starting pole angle: between -0.10475 and 0.10475 radians.
- Optional action: do-nothing.
- Optional restriction: require 0.75 of actions to be "do-nothing."
- Optional restriction: remove cart velocity and pole velocity observations.

In this paper, we employ the implementation of the cart-pole problem in the TENNLab Exploratory Neuromorphic Computing Framework [12,13]. This implementation has been used in multiple evaluations of neuromorphic processors and training algorithms [14–17]. We use this as an alternative to the more standard OpenAI implementation because its C++ implementation is much faster than the implementation in OpenAI. Functionally, however, the two implementations are equivalent, as the SNN's trained using the TENNLab implementation work identically when applied to the OpenAI implementation. The *Easy* and *Hardest* benchmarks work on OpenAI's implementation with no modification. The additional parameters for *Medium* and *Hard* are simple modifications to the OpenAI python environment. We have verified that the TENNLab-trained networks work identically on OpenAI Gym's implementation.

4. The Benchmarks and Performance Goals

Below we give some more description of the benchmarks, plus comments on performance goals for the benchmarks. For these goals, we apply a testing methodology to agents where they must perform the application on 1000 separate instances using 1000 different starting positions/velocities for the cart and pole. The average mission time over a possible time of 15,000 timesteps is reported. The starting positions and velocities used for testing must be different from those that are used for training.

For each benchmark, we give a target fitness, which is a fitness that we have verified to be obtainable, but challenging for spiking neural networks. Selecting a target is a bit of a black art,

 doi:10.20944/preprints202412.0532.v1

because training algorithms for regular and spiking neural networks are randomized, and success can be the result of luck. To wit, some of the best neural networks that we have trained have used hyperparameters that typically train poorly, but just happened to "strike gold" in one instance. Thus, our target values are those that should be achieved by a significant fraction of training runs for a set of hyperparameters.

4.1. The Easy *Benchmark – Too Easy*

With the *Easy* benchmark, the agent is given four observations: cart position, cart velocity, pole angle and angle velocity. The only actions are to push the cart left or right. These are the defaults for both OpenAI Gym and the TENNLab application. After performing many experiments on this benchmark, we agree with Anderson that this version of the problem is too easy [6]. In particular, Anderson solves the problem mathematically with a four-parameter controller, and concludes that testing random values of controller parameters is as effective as any learning strategy to solve this problem.

However, we believe that it is a good first-step problem for any machine-learning project that involves control applications, and is a necessary stepping stone to more challenging problems. Hence, we set a target fitness for this benchmark of 14,250 timesteps, which is 4 minutes, 45 seconds of mission time. Below we show that very simple spiking neural networks exceed this target.

4.2. The Medium *Benchmark – More Natural*

The *Medium* benchmark differs from the *Easy* benchmark in two ways. First, there is a third action in addition to push-left and push-right: do-nothing. Second, the fitness function is adjusted according to how often the do-nothing action is the selected action. The adjustment is defined as follows:

- Let the number of timesteps that the agent survives be t.
- Let the number of do-nothing actions be d.
- Define an "activity threshold", a, to be a number between 0 and 1.
- If $d/t > a$, then the fitness is equal to t.
- If $d/t \leq a$, then the fitness is equal to d/a.

For example, suppose the agent survives for 500 timesteps, 250 of which are do-nothing actions, and the activity threshold is 0.75. Then the fitness is equal to $250/0.75 = 333$. Were the activity threshold 0.25, then the fitness would be 500. As such, a higher activity threshold provides a reward for the do-nothing action.

In the *Medium* benchmark, we set the activity threshold to 0.75, which promotes more natural and less frantic agents than the *Easy* benchmark. With the *Medium* benchmark, the agents only get busy with the cart when the pole is out of balance. Otherwise, they simply let the system be.

Our target fitness for the *Medium* application is 12,000 timesteps, or 4 minutes, scaled by the activity factor if necessary.

4.3. The Hard *Benchmark – Removing Velocities, but Retaining "Do-Nothing"*

With the *Hard* benchmark, we remove the two observations that report velocities, and instead simply report the position of the cart and the angle of the pole. Without velocities, the agent is encouraged to use its memory of previous observations to determine how to handle the system. We retain the do-nothing action, but remove the activity threshold. The reason is that we have found that agents train more easily with do-nothing as an option.

Our target fitness for this application is 9,000 timesteps, or three minutes.

4.4. The Hardest *Benchmark – Removing Velocities, and Only Two Actions*

With the *Hardest* benchmark, we remove velocities and restrict the agent's actions to just push-left and push-right. This benchmark may be achieved using OpenAI gym and simply disallowing the agent

 doi:10.20944/preprints202412.0532.v1

to use two of the four observations. In our explorations, this has been the most difficult benchmark for which to train spiking neural networks.

We define a target fitness for this application as 6,000 timesteps, or two minutes. We also note that although we have trained spiking neural networks that well exceed this target, the training process is more difficult than with the other benchmarks.

5. Spiking Neural Networks

According to Roy, spiking neural networks compose the "third generation" of neural networks [1]. They are the focus of our research. Drawing inspiration from the brain, there has been a large amount of research dedicated to designing *neuroprocessors* that operate on spiking neural networks using every kind of hardware imaginable [18].

At the core of every neuroprocessor is a collection of integrate-and-fire neurons. Each neuron maintains an action potential whose value may increase or decrease as a result of spikes that arrive from the external environment or from incoming synapses. When the potential meets or exceeds a programmable threshold, the neuron fires, sending spikes along its outgoing synapses.

Each synapse has a programmable delay and weight. When a spike is fired from a synapse's pre-neuron, then it travels along the synapse to its post-neuron. The duration of this travel is the synapse's delay. When the spike arrives at the post-neuron, then the synapse's weight is added to (or subtracted from) the post-neuron's action potential.

The main feature of spiking neural networks that makes them attractive as computing platforms is their combination of computational expressivity [19] and low power. Spiking neuroprocessors are fundamentally different from von Neumann computers as there is no central memory or central processing unit that communicates with this memory. Since storage and processing is performed locally, the power demands on the neuroprocessor are reduced.

Various neuroprocessors provide constraints or add features to the description above [18]. For example, action potentials may have discrete values, upper and lower limits, and may leak their values over time [20]. They may also have refractory periods where they do not accept spikes for a period of time after they fire [21]. Synapse delays may be limited to discrete values [22]. Neuroprocessors may also feature learning rules, where the neuron and synapse properties become modified as a result of how and when they fire[23]. One popular learning rule is spike time-dependent plasticity (STDP), where a synapse's weight grows when it causes its post-neuron to fire, and shrinks when it doesn't [24].

One of the important usages of benchmarks is to help evaluate the various constraints and features of spiking neural networks. It is currently an open area of research to determine whether the various constraints and features of SNN's are too limiting, essential or unnecessary. A primary goal of this paper is to aid researchers in using the cart-pole application as one of these benchmarks.

5.1. Spike Encoding

When an application, such as the cart-pole application, presents its observations to a neuroprocessor, it must encode the observations into spikes. Common techniques are rate coding, spike coding, population coding (binning) and temporal coding. Schuman *et al* have evaluated multiple encoding techniques on multiple neuromophic applications and multiple neuroprocessors [16,25], and demonstrated conclusively that the encoding algorithm can have a significant impact on the success of a given neuroprocessor on a given application. One of the most affected applications was the cart-pole application [16]. As such, in this work, we study various spike encoding techniques and include spike encoding recommendations for each of the four benchmarks.

We refer the reader to [16,25] for more complete definitions of spike encoding, but we summarize the encoding algorithms that we explore below:

5.1.1. Values versus Spikes versus Time

When applying input to a single neuron, we explore three techniques for conveying information:

 doi:10.20944/preprints202412.0532.v1

- *Values*: The magnitude or weight of the value added to the neuron's potential is scaled by the value being encoded.
- *Spikes*: Values are converted into a number of spikes applied to the input neuron periodically. Higher values mean more spikes. The inter-spike duration is the same, regardless of the number of spikes, and each spike has the same weight (typically the maximum weight).
- *Time*: A single spike is applied to the neuron, with maximum weight. The timing of the spike is determined by the value being encoded. One can either have higher values correspond to earlier spikes or to later spikes.

5.1.2. Binning

Binning is a simple form of population coding, where an observation's value may be converted into spikes that are applied to one of multiple neurons [26,27]. These neurons are called "bins." The domain of the observations' values is partitioned equally among the bins. For example, the cart's position, which is a number in the range [-2.4,2.4], may be converted into spikes that go to two bins — negative values are converted into spikes in the first bin and positive values are converted into spikes in the second bin. Alternatively, if three bins are used, then values in [-2.4,-0.8] become spikes in the first bin, values in [-0.8,0.8] become spikes in the second bin, and values in [0.8,2.4] become spikes in the third bin.

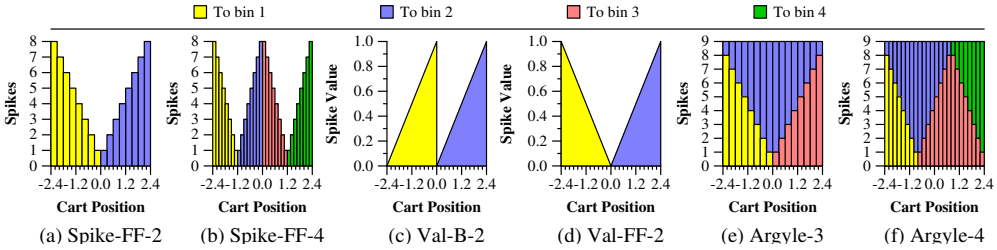
Once a bin has been selected, then the conversion of value to spikes is performed as described above in Section 5.1.1. As above, we refer the reader to [16,25] for definitions.

5.1.3. Six Highlighted Spike Encoders

In this paper, we make specific reference to six spike encoders:

1. **Spike-FF-2**: There are two bins, which, for all cart-pole observations, partitions the observations into negative values for the first bin and positive values for the second bin. The number of spikes applied to each bin is porportional to the absolute value of the observation, up to a maximum of eight spikes. The "FF" stands for "Flip Flop" [16]. The spikes all have a maximum charge value, and therefore cause their corresponding neurons to fire. Figure 1(a) illustrates how the cart positions are mapped to spikes with Spike-FF-2.
2. **Spike-FF-4**: Instead of two bins, there are four bins. Figure 1(b) illustrates how the cart positions are mapped to spikes with Spike-FF-4.
3. **Val-B-2**: There are two bins, and each value corresponds to a single spike into one of the bins. The spike's value is proportional to the cart's position within the interval for the bin, with a maximum value of 1.0. Figure 1(c) illustrates how the cart positions are mapped to bins and values with Val-B-2.
4. **Val-FF-2**: This is the same as Val-B-2, except the value of the spike to the first bin ranges from 1 to 0 rather than from 0 to 1. Figure 1(d) illustrates how the cart positions are mapped to bins and values with Val-FF-2.
5. **Argyle-3**: With the Argyle spike encoders, each value puts a total of nine spikes into two bins. With Argyle-3, the first and third bins are identical to the first and second bins of Spike-FF-2. The second bin receives spikes such that the total count of spikes for each value is nine. The name "argyle" comes from the spike encoder documentation within the TENNLab software [12,13,28]. Figure 1(e) illustrates how the cart positions are mapped to bins and spikes with Argyle-3.
6. **Argyle-4**: This continues the "argyle" pattern, but with four spikes instead of three. Figure 1(f) illustrates how the cart positions are mapped to bins and spikes with Argyle-4.

 doi:10.20944/preprints202412.0532.v1



(a) Spike-FF-2 (b) Spike-FF-4 (c) Val-B-2 (d) Val-FF-2 (e) Argyle-3 (f) Argyle-4

Figure 1. Six spike encoders that are highlighted in this paper, and how they map the cart position into spikes, bins, and in the case of the "Val" encoders, values.

5.2. Spike Decoding

As observations must be encoded into spikes that are inputs to the SNN, output spikes from the SNN must be decoded into actions. For this application, we use one output neuron for each potential action, and then use one of two techniques to decide how to turn spikes on these neurons into actions:

1. **Voting**: Here the number of spikes in each neuron is counted, and the one with the most spikes determines the actions. Ties go to the "lower" neuron.
2. **Temporal**: The first (or last) neuron to spike is the winner.

Although we test both decoding techniques in our experiments below, the voting decoder emerged as superior in all tests, so we employ that decoder in the remainder of the paper.

5.3. Neuroprocessor

In this work, we use the RISP neuroprocessor [17,29]. RISP is a bare-bones neuroprocessor, where neurons simply have programmable thresholds, and synapses have programmable delays and weights. There are some additional features with RISP, such as limiting thresholds and weights to discrete values, and allowing neurons to be configured with all-or-nothing leak. A neuron so configured leaks all of its activation potential to zero at the end of each integration cycle. One attractive feature of RISP is that because of its limited nature, SNN's developed for RISP are straightforward to port to other more complicated neuroprocessors. RISP has open-source support for both CPU simulation and FPGA implementation [28,30].

6. Training

For the training in this experiment, we use the EONS genetic algorithm [31] implemented in the TENNLab exploratory neuromorphic computing framework [12,13]. Since independent EONS runs are massively parallel, we employ a large variety of processors for training, including Linux, Macintosh and Raspberry Pi computers of all vintages. Unless otherwise specified, for each test, we run 100 independent optimizations.

7. Spike Encoding and Decoding Experiment

As an initial experiment, for each benchmark, we performed a grid search, similar to the grid search by Schuman *et al* [16], on a variety of encoding and decoding techniques. The number of combinations of techniques was 96. We used the default parameters of RISP (RISP-F, defined below in Section 8). The goal of this experiment is to determine candidate encoding and decoding techniques for each of the benchmarks, in order to do further explorations. As such, we opted for relatively short optimization runs: In EONS, we used populations of 25 SNN's, and ran the optimization for 100 epochs. For each test, we performed 75 independent optimizations.

In Figure 2, we display the testing fitnesses of the best ten encoding techniques for each benchmark. The displays are Tukey plots of the 75 independent runs. On the x-axis, we show the encoder if it is one of the six described in Section 5.1 above. Otherwise, we omit the label of the encoder.

 doi:10.20944/preprints202412.0532.v1

We have ordered the encoders by their third quartile fitness. The reason is that it is indicative of a "good" optimization for that encoder, that one can expect to achieve or exceed with 10-20 independent optimization runs. We don't order by the maximum fitness values, because sometimes a run is particularly lucky, and one cannot expect to achieve that fitness except in rare circumstances. Also, note that the *Hard* and *Hardest* fitnesses are displayed on different y-axes than *Easy* and *Medium*.

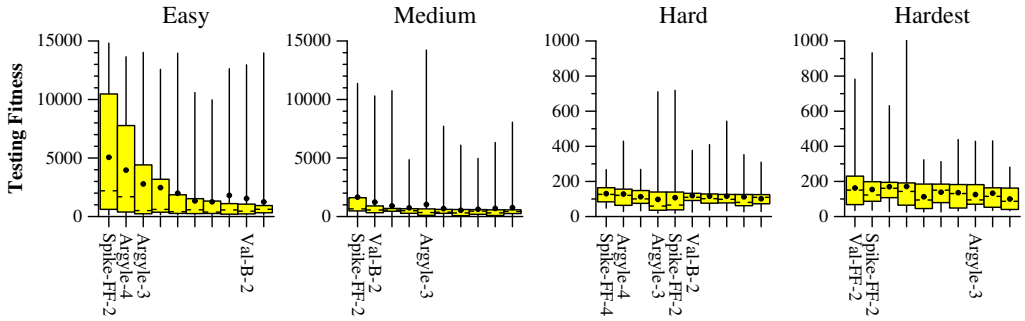


Figure 2. Results of testing 96 combinations of spike encoders and decoders for the four benchmarks. Each Tukey plot shows the testing fitnesses of 100 independent runs, ordered by the third quartile fitness.

For each benchmark, the best two encoders are from the six that we describe in Section 5.1.3. The *Easy* benchmark shows results that are reminiscent of the polebalancing tests in [16], with a similar encoder (Spike-FF-2) significantly outperforming the others. That encoder also significantly outperforms the others in the *Medium* benchmark, which appears to be a much harder problem indeed than the *Easy* benchmark.

With the *Hard* and *Hardest* benchmarks, the various encoders do not differentiate to a great degree in performance. It is interesting that Spike-FF-2 and Argyle-3, whose input spikes are superset of Spike-FF-2 (see Figure 1(a) and 1(e)), appear in the top ten networks for each of the four benchmarks. Val-B-2 appears in three of the four benchmarks.

From Figure 2, we conclude that we should focus on Spike-FF-2 as the encoder for the *Easy* and *Medium* benchmarks. For the others, we must explore further. Accordingly, we performed a second experiment where we only tested the six encoders listed in Section 5.1, but we extended the optimizations longer. Instead of using populations of 25 SNN's for 100 epochs, we used populations of 100 networks for 200 epochs. For *Easy* and *Medium*, we ran 100 independent jobs, and for *Hard* and *Hardest*, we ran 1000 independent jobs.

We show the results in Figure 3. A few conclusions stand out from this figure. First, we confirm that Spike-FF-2 is indeed the best encoder of the six on *Easy* and *Medium*, although in both, Argyle-4 is not far behind. In *Hard*, Spike-2-FF and Argyle-3 emerge as the most effective encoders, while in the *Hardest* benchmark, there is little discernible difference between Spike-FF-2, Argyle-3, Argyle-4 and Val-B-2.

To explore further, in Figure 4, we show the fitnesses and encoders for the 25 best networks produced in this experiment. This graph demonstrates that Argyle-4 clearly produces the best networks in *Hardest*. In the *Medium* test, even though Spike-FF-2 demonstrates the best overall performance, it only produced the 5th best network in the *Medium* benchmark. The difference, however, is not quite as pronounced as in the *Hardest* benchmark.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

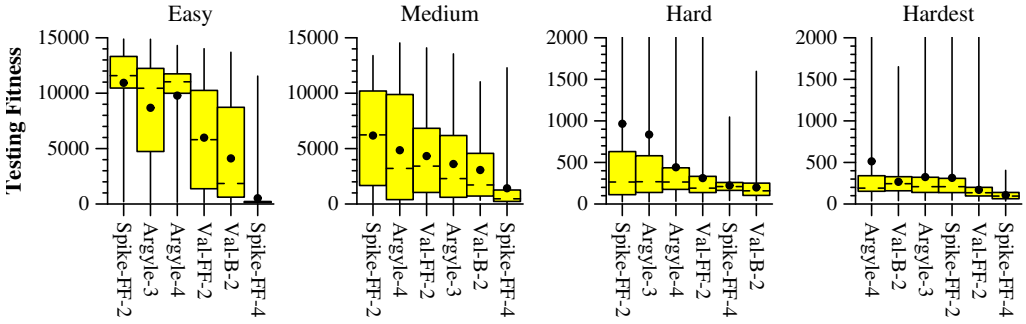


Figure 3. A second encoding experiment that tests the six encoders listed in Section 5.1 in longer optimization runs.

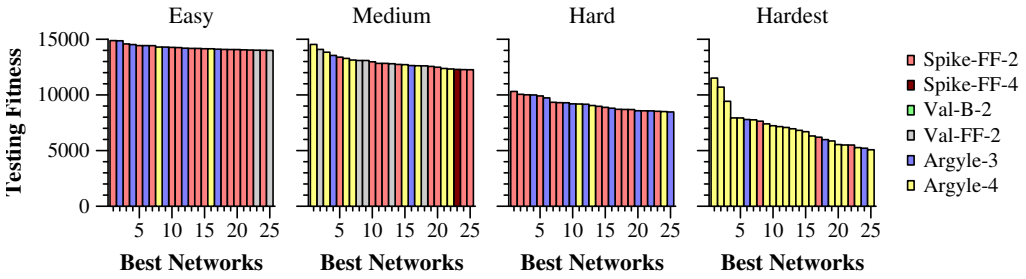


Figure 4. The 25 best networks, colored by their spike encoders, for each of the four benchmarks.

As a result of these tests, for the remainder of this work, we use Spike-FF-2 as the spike encoder for *Easy*, *Medium* and *Hard*. For *Hardest*, we use Argyle-4.

8. Neuroprocessor Experiment - RISP

In this section we use the four benchmarks to evaluate eight parameter settings of the RISP neuroprocessor. These are the eight "recommended" parameter settings from the open-source RISP simulator [17,28]. The settings are detailed in Table 1. RISP-F+ is the default parameter setting for RISP; however, because it uses floating point for thresholds and weights, it is only supported in CPU simulation. The last six parameter settings are supported by the FPGA version of RISP [30].

Table 1. Eight recommended parameters settings for the RISP neuroprocessor.

Label	Discrete Values	Threshold Range	Potential Range	Weight Range	Delay Range
RISP-F	No	$[0,1]$	$[-1,1]$	$[-1,1]$	$[1,15]$
RISP-F+	No	$[0,1]$	$[0,1]$	$(0,1]$	$[1,15]$
RISP-1	Yes	$[0,1]$	$[-1,1]$	$\{-1,1\}$	$[1,15]$
RISP-1+	Yes	$\{1\}$	$\{0,1\}$	$\{1\}$	$[1,15]$
RISP-7	Yes	$[0,7]$	$[-7,7]$	$[-7,7]$	$[1,15]$
RISP-15+	Yes	$[1,15]$	$[0,15]$	$[1,15]$	$[1,15]$
RISP-127	Yes	$[0,127]$	$[-127,127]$	$[-127,127]$	$[1,127]$
RISP-255+	Yes	$[1,255]$	$[0,255]$	$[1,255]$	$[1,255]$

As n gets larger, RISP-n requires more FPGA resources for implementing the neurons and synapses, with RISP-7/RISP-15+ requiring four bits to store weights, thresholds and potentials, and RISP-127/RISP-255+ requiring eight bits. The "plus" variants do not have inhibitory weights, and therefore may be more limited in what functionalities they can perform. Finally, RISP-1 and RISP-1+ are so limited that their neurons and synapses do not requiring arithemtic for implementation. As such, one may put larger RISP-1 and RISP-1+ networks onto an FPGA.

In Figure 5, we show the results of the eight processor settings with the four benchmarks, using the training parameters shown in Table 2. The Tukey plots are from 300 independent training runs per

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

parameter setting. The performance goals from Section 4 are shown on each graph in dark red, and at the top of the graph is the percentage of trained networks that meet or exceed the performance goals.

Table 2. Training hyperparameters for the four cart-pole benchmarks. These parameters are determined in the experiment detailed in the Appendix of this paper.

Benchmark	*Easy*	*Medium*	*Hard*	*Hardest*
Target Fitness	14,250	12,000	9,000	6,000
Encoder	Spike-FF-2	Spike-FF-2	Spike-FF-2	Argyle-4
Population Size	500	600	1000	600
Epochs	40	38	30	100
Episodes	10	10	10	10
Initial Passes	10	100	100	100
% Exceeding Target	19.4	23.6	9.2	8.9
3rd-Quartile Pi5 Timing (m)	4.9	11.6	8.5	7.1

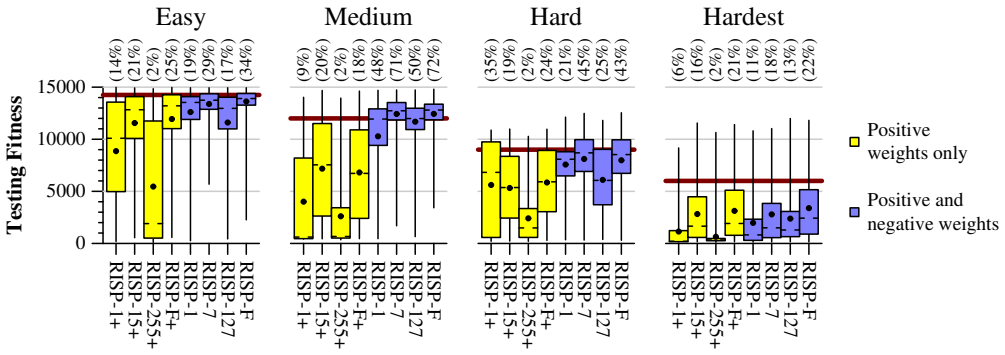


Figure 5. Testing fitness results for the four benchmarks and eight recommended perparameter settings of the RISP neuroprocessor. The goals for each benchmark are drawn as lines in dark red.

The benchmarks allow us to draw the following conclusions about the RISP parameter settings. First, the settings that allow negative weights train much better on all four benchmarks. In particular, with the *Hard* benchmark, the best 3% of the networks *all* have negative weights. Thus, when selecting a neuroprocessor for these applications, it is more advantageous to use one that employs negative weights.

Second, of the processors with negative weights, the performance, from best to worst is RISP-F, RISP-7, RISP-127 and RISP-1, with the only non-negligible difference between RISP-7 and RISP-F coming in the *Hardest* benchmark. This is significant, because RISP-7, unlike RISP-F, has just 14 synapse weights and neuron potentials, and just 7 neuron thresholds. It has an FPGA implementation [17] whose small values lead to efficient use of FPGA resources. This is an encouraging result for the effectiveness of RISP-7.

It is also encouraging that the very limited neuroprocessors, RISP-1+ and RISP-1, produce networks that exceed our goals. We present examples in the next section.

The performance of RISP-127 and RISP-255+ is a curiosity relative to the others. The reason is that RISP-127 is a superset of RISP-1 and RISP-7, and RISP-255+ is a superset of RISP-1+ and RISP-15+; yet their performance is worse than their counterparts. We hypothesize that the long delays available in RISP-127 and RISP-255+ penalize their training. Specifically, for each timestep of the cart-pole application, we run the neuroprocessor for 24 timesteps. Therefore, the long delays of RISP-127 (up to 127 neuroprocessor timesteps) and RISP-255+ (up to 255 neuroprocessor timesteps) will cause input spikes during one application timestep to produce output spikes in up to 10 future application timesteps. That is unlikely to be beneficial, or at the very least, makes training difficult.

To test this hypothesis, we reduced the maximum synapse delay in RISP-127 and RISP-255+ to 15, as with the other neuroprocessor settings. The results are in Figure 6, with the two new data points

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

labeled with "D15". In these graphs, RISP-127 fits naturally between RISP-7 and RISP-F, and RISP-255+ fits naturally between RISP-15+ and RISP-F+. This underscores the importance of considering the implications of certain parameter settings with respect to specific applications.

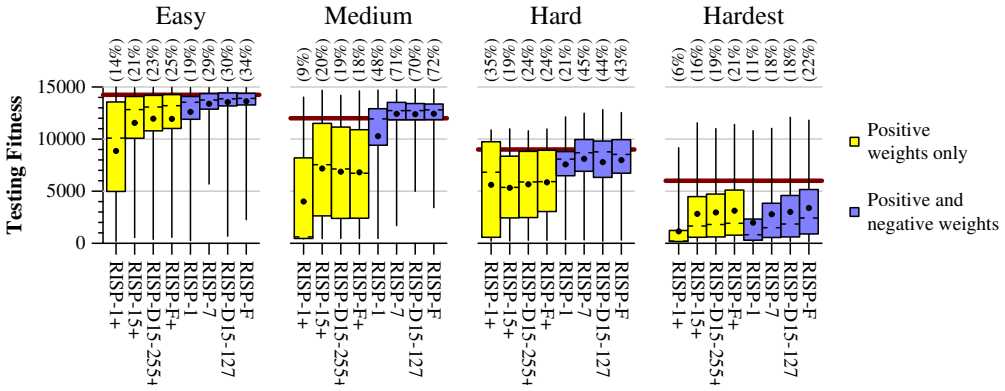


Figure 6. The same graph as Figure 5, but with the maximum delays in RISP-127 and RISP-255+ reduced to from 127 and 255 to 15.

9. Example Networks

In this section, for each benchmark, we present two of the best networks from the previous exploration. For each of these networks, we try to glean some information about which network properties aid in solving the problem, with the hope of figuring out some infomation about "good" network properties, that may be advantageous in designing further good networks.

Each of these networks has been included as an example network in the TENNLab Open Source Neuromorphic Computing Framework, along with instructions on how to employ it in the Gymnasium python environment [28].

9.1. Two Example Easy *Networks*

In Figure 7, we show two RISP-1+ networks that were generated in the previous experiments. The left network achieves a testing fitness of 14,970.2 timesteps (4 minutes, 59.4 seconds), and the right network achieves a testing fitness of 14,985.0 timesteps (4 minutes, 59.7 seconds). Both networks are stunning in their simplicity: Between 7 and 9 synapses each, and no cycles. The $x-$ and $dx+$ inputs are ignored, which is not uncommon for EONS-trained networks. Functionally, when a network ignores one of the two neurons that encode a value, such as the $x-$ neuron, it simply treats the lack of spikes on the other neuron (here $x+$) as significant. In the case of the network in Figure 7(a), over the 1000 testing runs, the cart stays in the center 1/8th of the track 94.2% of the time, so even though the $x-$ input is ignored, the network does in general center the cart on the track.

The networks are simple enough that each is runnable on nearly all integrate-and-fire neuroprocessors, regardless of whether they are leaky. As such, we verified that the network performs identically on all eight of the RISP settings listing in Table 1. Moreover, since synapse fires always result in neuron fires, this network is very tolerant of noisy synapses [32–34] — the neuron thresholds may be set as low as possible, and the synapse weights as high as possible, to assure that neurons always fire when synapses fire.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

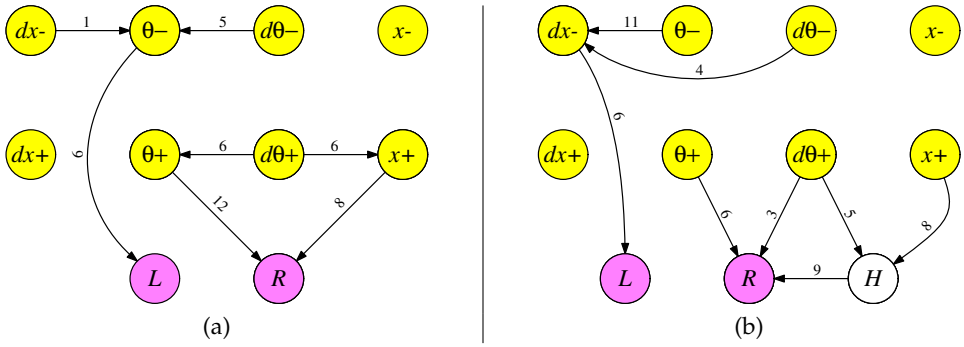


Figure 7. Two RISP-1+ networks that solve the *Easy* benchmark of the cart-pole problem. The network in (a) averages of 14,970.2 timesteps (4 minutes, 59.4 seconds), and the network in (b) averages 14,985.0 timesteps (4 minutes, 59.7 seconds). All neuron thresholds and synapse weights are one. The synapse delays are specified above each synapse.

The similarity of the two networks is also striking and worth further examination. In all of these experiments, there are 24 neuroprocessor timesteps per application timestep, and each input value is converted into a number of spikes from zero to eight. Therefore, input spikes potentially arrive every three timesteps. In both networks, the delays on the synapes from $dx-$, $\theta-$ and $d\theta-$ all have different values when taken modulo 3. Therefore, their spikes always arrive at the L neuron at different timesteps, and to a first degree of approximation, in both networks, one can calculate the number of spikes arriving at L as follows:

$$L \approx \left\lceil \frac{8(dx-)}{2.0} \right\rceil + \left\lceil \frac{8(\theta-)}{0.209} \right\rceil + \left\lceil \frac{8(d\theta-)}{2.0} \right\rceil. \quad (1)$$

This calculation is approximate, because there are times when the delays in the synapses mean that the spikes arrive at L on the next application timestep rather than the current one. In network 7(b) for example, if there are more than six spikes from $dx-$, three spikes from $\theta-$, or five spikes from $d\theta-$, then L receives those spikes in the next application timestep.

This is a very subtle effect, because it happens rarely. Over the course of the 1000 tests, there are spikes that go to the next application timestep in just 0.17% of the timesteps. If the synapse delay from $\theta-$ to L is changed from 6 to 9, then the fitness is reduced by 30 timesteps – over half a second.

In both networks, the combination of $x+$, $\theta+$ and $d\theta+$ cause the R neuron to spike, and as with the L neuron, there are subtleties. In both networks, $\theta+$ and $d\theta+$ have synapses whose delays are multiples of three, meaning their spikes align, and therefore are not always summed. For example, suppose that in network 7(b), $\theta+$ spikes once and $d\theta+$ spikes twice. The R neuron receives one spike at time 3 and two at time 6. Because the two arrive simultaneously at time 6, R only spikes twice instead of three times. This has the effect of implementing a conditional *max* operation. Suppose we define `cm(i,j)` with the following C code:

```
int cm(int i, int j)
{
  if (i > j) return i;
  if (i == 0) return j;
  return j+1;
}
```

Then, we may approximate the number of spikes on R as:

$$R \approx cm\left(\left\lceil \frac{8(d\theta+)}{2.0} \right\rceil, \left\lceil \frac{8(\theta+)}{0.209} \right\rceil\right) + cm\left(\left\lceil \frac{8(d\theta+)}{2.0} \right\rceil, \left\lceil \frac{8(x+)}{2.4} \right\rceil\right). \quad (2)$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

As with L, there are times when spikes will arrive at R on the next application timestep rather than the current one. Therefore, even though both networks may be reduced to rather simple equations at a high level their actual operation is more subtle.

In Figure 8, we show the x observations for a single run of the application on each network. In these graphs, and in subsequent timeline graphs, the starting observations of the system are the same, and all networks run the application successfully for the entire mission time of five minutes. In comparison to subsequent networks, these networks do an excellent job of keeping the cart centered and relatively still. At times, the cart drifts to the right, and then gets corrected (sometimes over-corrected), and goes back to its stable state. It is interesting to note that although the two networks are extremely similar, and although the anomalies in the timeline graphs have very similar shapes, the two networks run quite differently. For example, in the five-minute span, network (a) has three overcorrections, while network (b) has five.

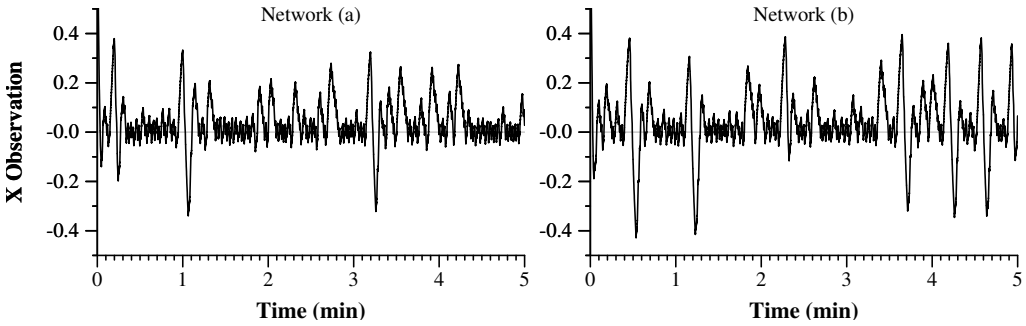


Figure 8. Timeline of the x observations of an example run of the two *Easy* networks.

9.2. *Two Example* Medium *Networks*

In Figure 9 we show two networks for the *Medium* benchmark. Both well exceed the desired fitness of 12,000 timesteps — the RISP-1 network in Figure 9(a) has a testing fitness of 14,698.7 timesteps (4 minutes, 53.9 seconds), and the RISP-127 network in Figure 9(a) has a testing fitness of 14,820.5 timesteps (4 minutes, 56.4 seconds). Both networks have at least 75% of their actions be "do-nothing," which means that they show considerably less movement than the *Easy* benchmark networks. Both networks feature inhibitory as well as excitatory synapses, plus recurrent connections.

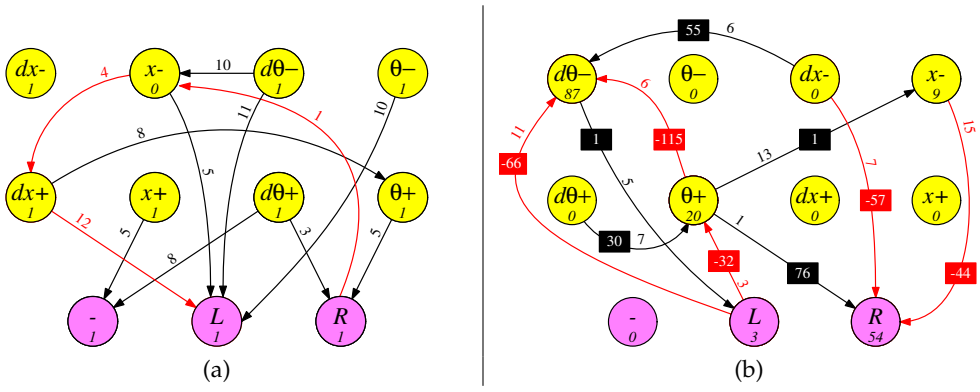


Figure 9. Two networks for the *Medium* benchmark. (a) A RISP-1 network which achieves a fitness of 14,698.7 timesteps, and (b) A RISP-127 network which achieves a fitness of 14,820.5 timesteps. In the RISP-1 network, all synapses weights are either -1 (red) or 1 (black). In RISP-127 networks, the weights are shown in the boxes. In both networks, the neuron thresholds are shown in the bottom of the neurons, and synapse delays are noted on the synapses.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

Although the networks still feature a small number of synapses (12 and 10), the networks are complex enough that one cannot merely "eyeball" them to draw conclusions about how they work. Instead, we must probe a bit.

Focusing on the network in Figure 9(a), We see that the only times that the "-" neuron spikes is when $x > 0$ or $d\theta > 0$. Moreover, if $x < 0$ or $d\theta < 0$, then L will definitely spike. This raises a curiosity. At a first glance, we would expect for each variable to be positive half of the time and negative half of the time, and for them to be roughly independent. However, were that true, then $x-$ and $d\theta-$ would both be negative 25% of the time, and in these cases, the "-" neuron does not fire. It is easy to concoct scenarios where $x+$ spikes once, $d\theta < 0$ and $theta+$ spikes more than once. These cannot be "do-nothing" scenarios, so it seems that we cannot have "do-nothing" compose at least 75% of the actions.

The answer must be that the network ensures that the variables are not equally positive and negative, and they are not independent. To confirm, we ran on the testing scenario of 1000 episodes, and see that "do-nothing" is indeed the selected action 82% of the time. This is achieved partially by keeping the cart in the right half of the track — x is positive 97.0% of the time. Similarly, $d\theta$ is positive 55.0% of the time, and the combination of either x or $d\theta$ being positive occurs 98.4% of the time. Therefore, the network ensures that the "-" neuron is spiking during nearly all application timesteps, skewing the system toward "do-nothing."

The network in Figure 9(b), on the other hand, never spikes the "-" neuron; yet, "no-action" is the chosen action 81.0% of the time. Therefore, during 81.0% of the timesteps, there are no output neurons that spike. An examination of the network shows that of the 10 synapses, five are positive, with a total weight of 163, while five are negative, with a total weight of -314 – clearly the overall nature of this network is to inhibit spiking.

Put another way, the only time that L spikes is when $d\theta-$ spikes at least three times (9.5% of the time), and the only time that R spikes is when $theta+$ spikes more than $x-$ and $dx-$ (10.1% of the time). Clearly the network is skewed toward not spiking.

Figure 10 shows the x observations over time on the same starting parameters as Figure 8. Since the two lines do not collide, we draw them on the same graph. The RISP-1 network keeps the cart on the right half of the track and the RISP-127 network keeps it on the left half. It is clear from the picture that the RISP-127 network is both less active and less reactive than the RISP-1 network, as the cart travels a lesser distance, and exhibits much less back-and-forth.

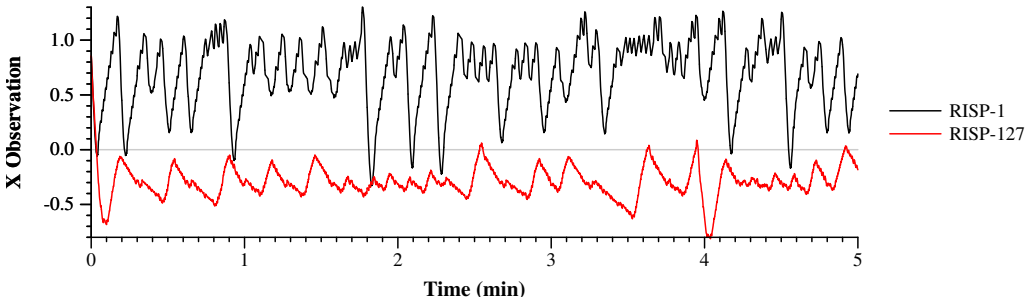


Figure 10. Timeline of the x observations of an example run of the two *Medium* networks.

9.3. Two Example Hard *Networks*

In Figure 11 we show two networks for the *Hard* benchmark. Both exceed the desired fitness of 9,000 timesteps — the RISP-1 network in Figure 11(a) has a testing fitness of 12,123.8 timesteps (4 minutes, 2.5 seconds) and the RISP-7 network in Figure 11(b) has a testing fitness of 12,479.3 timesteps (4 minutes, 9.6 seconds). Although each network contains a "no-action" action, neither is required to use it.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

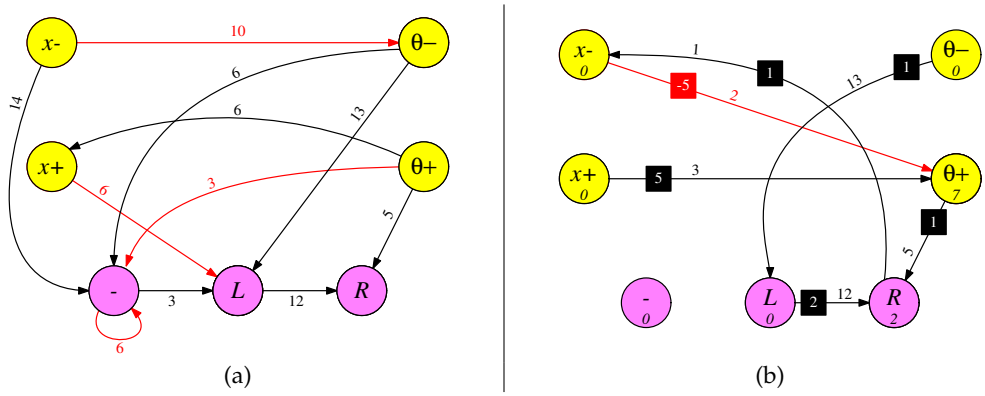


Figure 11. Two networks for the *Hard* benchmark. (a) A RISP-1 network which achieves a fitness of 12,123.8 timesteps. All neuron thresholds in this network are zero. (b) A RISP-7 network which achieves a fitness of 12,479.3 timesteps. Weights, thresholds and delays are shown as in Figure 9.

Both networks are simpler than their *Medium* counterparts, featuring fewer neurons and synapses. Focusing on the network in Figure 9(a), 7 of the 10 synapses have delays that are multiples of three, which is reminiscent of the *Easy* RISP-1+ networks above, and ensures that some spikes will arrive at neurons simultaneously. For example, neurons $x+$ and $\theta+$ have a similar relationship in Figure 9(a) as neurons $\theta+$ and $d\theta+$ in Figure 7(a). Since all of the neurons have thresholds of zero, when two positively weighted spikes arrive at a neuron simultaneously, the neuron only one spikes once, implementing a functionality like `cm(i,j)` in Section 9.1 above.

The network in Figure 9(b) contains only six synapses, and only one of these is inhibitory. One interesting feature of both networks is the path from $\theta-$ to R, through neuron L. Consider what happens when $\theta-$ fires. Thirteen timesteps later (in each network), it causes L to fire, which, 12 timesteps later, causes R to fire. Thus, whenever neuron $\theta-$ fires during one application timestep, then L fires either in the same application timestep or the next, and R always fires in the next application timestep. The effect is one of balance – whenever L fires, R fires soon thereafter.

It is interesting that although both networks have a "do-nothing" neuron, neither is required to produce "do-nothing" actions. Regardless, roughly a fifth of each network's actions are "do-nothing." We surmise that one reason that the *Hard* benchmark is easier than the *Hardest* one is the availability of the action that does nothing.

We show timelines for the two networks in Figure 12. Both networks oscillate the cart periodically from left to right. The RISP-1 network oscillates roughly every 8 seconds, keeps the cart on the left side of the track and uses less of the track overall. The RISP-7 network oscillates roughly every 10 seconds, keeps the cart centered on the track, and travels past $x = -1$ and $x = 1$ on nearly every oscillation.

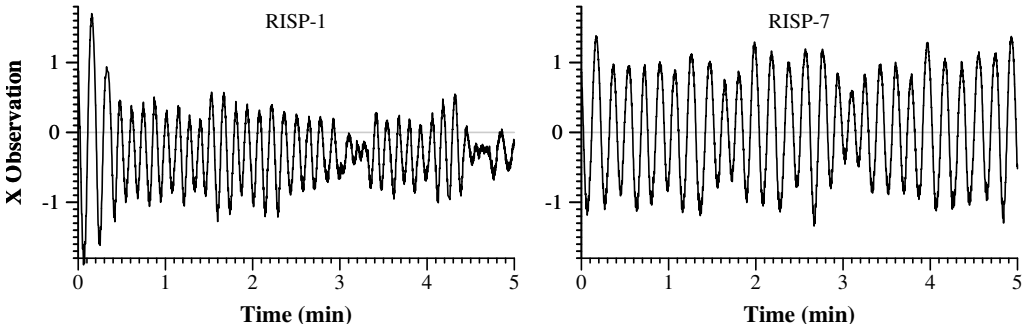


Figure 12. Timeline of the x observations of an example run of the two *Hard* networks.

One feature of the *Hard* benchmark is that the lack of velocity information encourages the networks to have some notion of memory, from timestep to timestep. The paths from $\theta-$ through L to R in

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

each network are a simple examples of memories, where actions from one application timestep are "remembered" and reflected in the next application timestep.

9.4. *Two Example* Hardest *Networks*

In Figure 13 we show two networks for the *Hardest* benchmark. As with the other networks, both exceed the desired fitness of 6,000 timesteps — the RISP-15+ network in Figure 13(a) has a testing fitness of 11,342.9 timesteps (3 minutes, 46.9 seconds), and the RISP-127 network in Figure 13(b) has a testing fitness of 11,991.6 timesteps (3 minutes, 59.8 seconds). Both networks use the Argyle-4 encoder (please see Figure 1(f)), where each input causes exactly nine spikes into two of four neurons. As such, the neurons are labeled, for example, from smallest to biggest, $x--$, $x-+$, $x+-$ and $x++$. Each network has a hidden neuron labeled H.

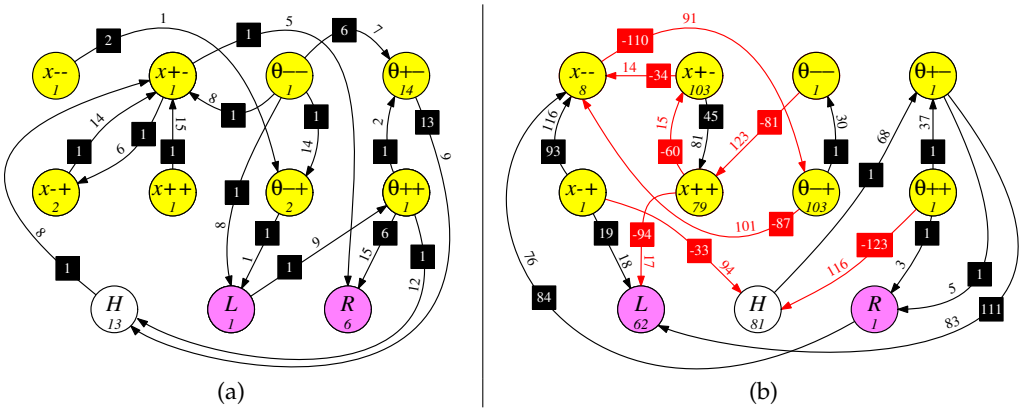


Figure 13. Two networks for the *Hardest* benchmark. (a) A RISP-15+ network which achieves a fitness of 11,342.9 timesteps. All synapses are drawn in black, as RISP-15+ does not have inhibitory synapses. (b) A RISP-127 network which achieves a fitness of 11,991.6 timesteps. Weights, delays and thresholds are shown as in the previous Figures.

The RISP-15+ network has 16 synapses, all of which are positive. As with the networks in the previous section, there is a path from L to R (in this case, through $\theta++$) with a delay of 24, meaning that whenever L fires in one application timestep, then R fires in the next timestep. As will be seen in the timeline graph below, this network has the cart spend the majority of its time (63.1%) on the left side of the track. It should be noted that the longer path lengths in this network provide a memory between application timesteps. For example, the path from $\theta--$ to $\theta+-$ to H to $x+-$ to R has a total delay of 29, which spans application timesteps.

Table 3. Number of RISP-127 networks with long delays in the top 100 networks for each benchmark, generated by the experiment in Section 8.

Easy	*Medium*	*Hard*	*Hardest*
3	7	4	11

The RISP-127 network was generated during the first set of tests in Section 8, before we lowered the maximum delays for RISP-127 and RISP-255+ to 15. As such, this network features delays that span many application timesteps. For example, the synapse from $\theta--$ to $x++$ has a delay of 123, meaning it spans five application timesteps. This is a longer-term memory than the previous examples, which only span one application timestep. We hypothesize that this may help with the hardest version of the benchmark. To support this hypothesis, we examined the 100 best networks for each benchmark, generated in Section 8, and counted the number that were RISP-127 with the long delays. The results are in Table 3, and they do support the hypothesis, although of course they are far from conclusive.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

In Figure 14, we show the x observations for an example run of each network. As mentioned above, the RISP-15+ network keeps the cart in the left half of the track. Both networks oscillate the cart as in *Hard* networks, however in each, there are times when the oscillation is broken, presumably to reach a better steady-state. The RISP-127 network centers the cart and exhibits a much tighter oscillation around $x = 0$.

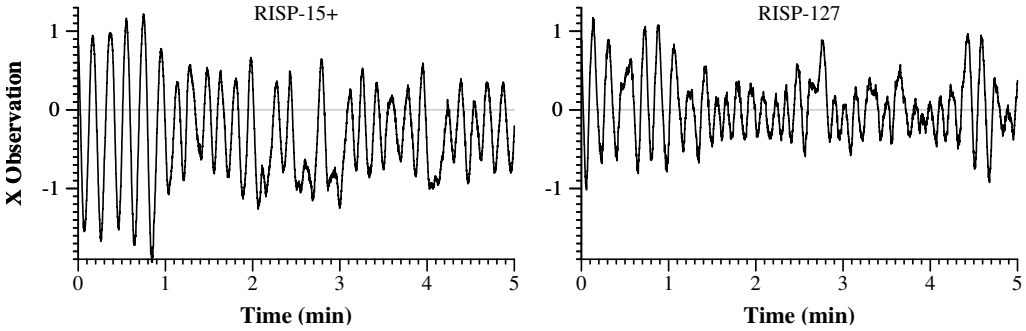


Figure 14. Timeline of the x observations of an example run of the two *Hardest* networks.

In summary, the networks displayed in this section do not exhibit regular structure, which makes it difficult to draw conclusions about them. We have attempted to do so with close examination of the networks combined with measurements of observations, actions and spike counts while they run.

10. Related Work

We have placed related work at the end of the paper, because we feel that it's best to have context from the results of the paper before explaining the related work. There are too many papers employing the cart-pole problem for machine learning to list, so we focus on a sampling.

As mentioned above, early presentations of thie problem by Barto *et al* [5] and Anderson [6] surmise that the problem is too easy to be an effective machine-learning problem, and that random search for control parameters is often as good of a solution as more complex ones. As mentioned above, we agree that the *Easy* benchmark, which is the one on which they focus, is too easy, which is why we focus additionally on harder versions of the problem. Their approach to input encoding is to partition the input space into 162 bins for each of the four parameters, and at each time step, set one of those bins to one, and the other 161 to zero. Their metric is time steps until failure, for which they achieve values exceeding 200,000 on certain trials, while other trials fail very quickly.

In Table 4, we list 12 sample papers (including this one) and compare them very loosely. The reason for the looseness is that each paper has a different focus, often with a different formulation of execution, success and failure, and with different extensions to the problem. All results pertain to the *Easy* benchmark.

Table 4. Sampling of papers on the cart-pole problem over the last 17 years.

Year	Citation	Data Structure	ML Technique	Estimated Parameters	Measure of Success (sec)
2007	Schaefer *et al* [35]	RNN/RCNN	BPTT	240	2,000
2017	Nagendra *et al* [36]	Various	Various	44	2,000
2017	Chang *et al* [37]	ANN	Generational	212	4
2020	Kumar [9]	DNN	Q-learning	~800	3.9
2021	Buvanesh [38]	DNN	Actor-critic/DeepQ	130,000	4,000
2021	Variengien *et al* [39]	Neural Cellular Automata	DeepQ	1,854	200
2021	Rafe *et al* [40]	Izhikevich Neuron	Evolutionary	8-256	3.9
2022	Shin & Kim [41]	DNN	Genetic	67	10
2022	Ding *et al* [11]	Leaky Integrate & Fire SNN	Gradient Descent	4096	9
2022	Tran *et al* [42]	ANN/DNN on FPGA	DeepQ	64	4
2023	Han *et al* [43]	FNN/CNN	Q-Learning, Sarsa	?	4
2024	This paper	Integrate & Fire SNN	Genetic	25	300

 doi:10.20944/preprints202412.0532.v1

We make a final mention to a flamboyant web posting entitled "How to Beat the CartPole Game in 5 Lines – A Simple Solution without Artificial Intelligence," by Xu [44]. In this posting, the author proposes the following simple solution to the cart-pole problem that only uses the θ and $d\theta$ observations (transposed to C/C++):

```
char cart_pole(double theta, double dtheta) {
{
  if (fabs(theta) < 0.03) return (dtheta < 0) ? 'L' : 'R')
  return (theta < 0) ? 'L' : 'R';
}
```

When we test this solution on 1,000 episodes, it achieves an average fitness of 682.7 timesteps (13.7 seconds), which succeeds given the objectives of OpenAI Gym, but falls short of the goals of this paper. In contrast, we can build a similar simple agent that sets L and R using Equations 1 and 2 from Section 9.1 (including the `cm()` function defined there) and compares them:

```
char cart_pole(double x, double dx, double theta, double dtheta)
{
  double x_p, dx_m, theta_p, theta_m, dtheta_p, dtheta_m, l, r;

  dx_m = (dx < 0) ? -dx : 0;
  theta_m = (theta < 0) ? -theta : 0;
  dtheta_m = (dtheta < 0) ? -dtheta : 0;
  x_p = (x > 0) ? x : 0;
  theta_p = (theta > 0) ? theta : 0;
  dtheta_p = (dtheta > 0) ? dtheta : 0;

  l = ceil(8.0*dx_m/2.0) + ceil(8.0*theta_m/0.209) + ceil(8.0*dtheta_m/2.0);
  r = cm((int) ceil(8.0*dtheta_p/2.0), (int) ceil(8.0*theta_p/0.209))  +
      cm((int) ceil(8.0*dtheta_p/2.0), (int) ceil(8.0*x_p/2.4));

  return (l >= r) ? 'L' : 'R';
}
```

This agent achieves an average fitness of 14,970.1 on the 1,000 testing episodes (0.1 lower than the RISP-1+ network from which it was derived).

11. Conclusions

In this paper, we have examined the well-known Cart-Pole application in detail as it pertains to neuromorphic computing. We have agreed with others that the standard formulation of the problem is not challenging to be employed as a benchmark. Instead, it provides a good starting point for other more challenging benchmarks.

Accordingly, we have defined four parameter settings of the problem, named *Easy*, *Medium*, *Hard* and *Hardest*. Two of these settings (*Easy* and *Hardest*) map directly to the OpenAI Gym environment, and the other two may be employed with some simple modifications. We have also defined target achievements for all four benchmarks.

We have performed extensive experimentation with the Cart-Pole problem on the RISP neuroprocessor. The first experiment determined the best encoding and decoding techniques when rendering the problem on a spike-based neuroprocessor. The second experiment (in the Appendix) determined recommended hyperparameter settings for using the EONS genetic algorithm [31] to train spiking neural networks for the four benchmarks. These settings consider both success in fitness and compute time. The third experiment uses results from the first two experiments to determine the effectiveness of eight recommended parameter settings for the RISP neuroprocessor. As anticipated, more complex

 doi:10.20944/preprints202412.0532.v1

neuroprocessor features generated more effective networks; however, some features, such as very long maximum synapse delays, needed to be restricted to gain better traction in optimization.

Finally, from the last experiment, we selected eight spiking neural networks that achieve the desired benchmarking goals on the four benchmarks. We explore these networks both quantitatively and qualitatively to reflect on what features they have that enable them to solve the application effectively. The networks for the *Easy* benchmark are so simple that they do not require integration. Accordingly, they may be applied to any spiking neuroprocessor, even those that feature stochasticity and noise. The RISP-1 network for the *Medium* benchmark is also exceptionally simple, featuring only excitatory and inhibatory synapses with unit weights. It too may be applied to a variety of neuroprocessors.

The eight networks are included in the TENNLab Open Source Neuromorphic Computing Framework, along with instructions on how to apply them to OpenAI gym [28]. They may also be executed on the open source RISP FPGA [30].

With these networks, we have demonstrated that the Cart-Pole problem, in all of its difficulty levels, may be implemented effectively with spiking neural network agents, where the neuroprocessor is quite simple (no leak, refractory periods or learning rules), and the networks are quite small (under 12 neurons and under 20 synapses). The size of these networks, as well as their simplicity, lends them to very efficient neuromorphic hardware implementations.

In demonstrating these small, simple spiking neural networks' ability to solve various configurations of the Cart-Pole problem, we highlight their ability to operate on applications featuring "state homeostasis." These types of problems are ubiquitous and are found across domains such as health, aerospace, and many others where low-power solutions are necessary for edge or embedded computing. The networks presented in this work exemplify solutions that are low-SWaP and whose simplicity aids in their explainability.

In future work, we plan to address more control applications, such as those in OpenAI Gym, and those involving UAV control. It is our hope that we may demonstrate, as in this paper, that simple neuroprocessors with relatively small neural networks may be effective agents for these control problems.

Author Contributions: Conceptualization, J.P., C.R., and C.S.; methodology, J.P.; software, J.P., C.R., and C.S.; validation, J.P., and C.W.; formal analysis, J.P.; investigation, J.P., C.R., and C.W.; resources, J.P.; data curation, J.P., and C.R.; writing—original draft preparation, J.P.; writing—review and editing, J.P., C.R., C.W., and C.S.; visualization, J.P.; supervision, J.P., and C.S.; project administration, J.P.; funding acquisition, J.P., and C.S.

Funding: This research was funded by AFRL grant number FA8750-21-1-1018, ARL, and a gift from Accenture.

Acknowledgments: The authors thank Jeff Young, Nick Skuda, Larry Barnett, Sylvia Rilee, Rachel Knotts and Daniel Mallett for contributing computing resourses for this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Experiment to Determine Training Hyperparameters

In this section, we detail experiments to derive training hyperparameters for the four benchmarks. The material in this section is specific to the TENNLab software framework, but may have usefulness in other training environments, especially those that rely on genetic algorithms for training.

Appendix A.1. Parameters for Easy

For the *Easy* benchmark, we performed two sets of experiments. The first increased the population size from 50 to 2000, while keeping the product of population size and epochs constant at 20,000. For each population size, we performed 500 independent optimization runs, and show the testing fitnesses in Figure A1(a).

 doi:10.20944/preprints202412.0532.v1

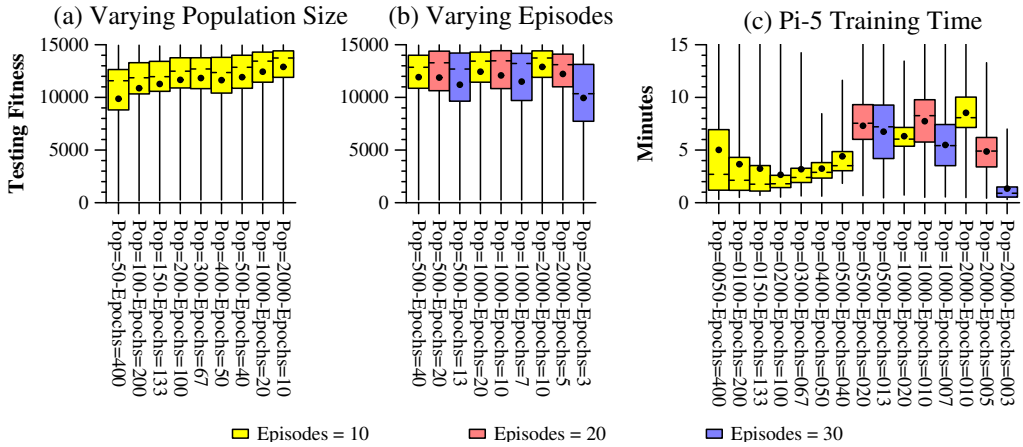


Figure A1. Experiment to determine hyperparameter settings for the *Easy* benchmark.

As is clearly demonstrated in the figure, fitnesses improve with population size, with the best fitnesses obtained using a population of 2,000 networks. With this population size, and just 10 epochs of training, the median testing fitness is 13,740 timesteps (4 minutes, 35 seconds), and the third quartile testing fitness is 14,419 timesteps (4 minutes, 49 seconds).

A second hyperparameter to test is the number of application episodes per training run. Increasing this parameter increases the running time of training, but can be effective in reducing overfitting. As such, we test settings of this parameter at 10, 20 and 30 for population sizes of 500, 1000 and 2000 in Figure A1(b). For these tests we maintain a fixed product of population size, epochs and episodes. We affix this value at 200,000 (same as in Figure A1(a).

In this figure, increasing the episodes does not improve the testing fitnesses. This effect is most pronounced with the large population size of 2000. We presume that the reason is that it cuts down on the number of epochs too severely.

In Figure A1(c), we show timing information. Because these optimizations are CPU-intensive, we performed them on a cobbled collection of machines, ranging from a Raspberry Pi-3 at the slowest, to a MacBook Pro with the Apple M3 Pro chip. This collection includes five Raspberry Pi5's, which are among the faster processors on these optimizations (we note that the optimizations do not feature operations that work naturally on GPU's or map easily to vector instructions, and run with surprising speed on the Pi5's). Figure A1(c) shows the timings from the Pi5's, dedicating all four of the Pi's CPUs to each run.

While the product of population size, epochs and episodes is constant in all of these runs, their running times differ due to other factors:

- Runs with better fitnesses take longer, because the application must simulate more timesteps.
- We stop execution when the training fitness stalls at a maximum value for eight successive epochs. Therefore, runs with higher populations and more epochs are likely to truncate earlier. Runs with more episodes are likely to truncate later.
- Some SNN's have more activity than others and take longer to simulate.

Given the results of these graphs, we advocate for using a population of 500 and with 10 episodes for 40 epochs. In our tests, nearly a fifth of the networks (19.4%) exceed the target testing fitness of 14,250 timesteps, and 78.6% of the runs complete in under five minutes on the Pi5. If one wants to devote more CPU time to the problem, then increase the population size to 2000 and decrease the epochs to 10. Nearly a third of the networks (32.8%) exceed the target testing fitness, and 73.6% of the runs complete in under 10 minutes on the Pi5.

 doi:10.20944/preprints202412.0532.v1

Appendix B. Parameters for *Medium*

As in the previous section, we began our search for hyperparameters by expanding the population size and epochs. After an initial exploration, we settled on the following grid search:

- Vary the population size in $\{600, 800, 1000\}$.
- Vary the epochs so that the product of epochs and population size is a constant $c \in \{60000, 30000, 22500\}$.
- Perform i initial passes to create the first population. At each pass, insert the best $\frac{p}{i}$ networks into the first population. Vary $i \in \{10, 100\}$.

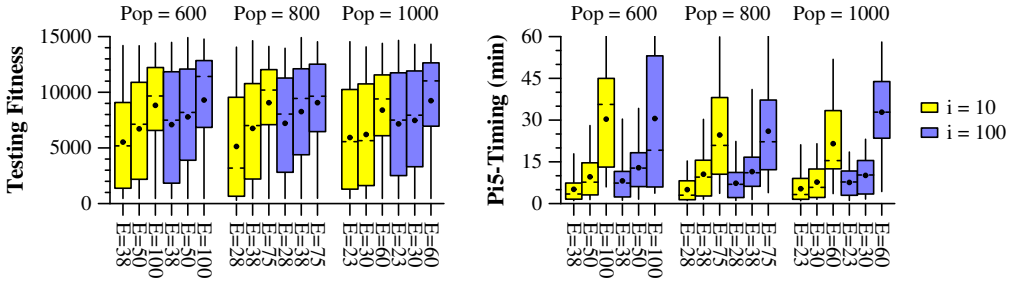


Figure A2. Experiment to determine hyperparameter settings for the *Medium* benchmark.

We display the testing fitnesses and Pi5 timings of the 18 parameter settings in Figure A2. For the six settings where the median timing was over 15 minutes, we truncated the tests at 100 runs, as we determined that these took too long. These are the tests where the product of epochs and population size is 60,000.

For the remaining parameters, we ran 300 independent runs for each parameter setting. It is clear from Figure A2 that more initial passes ($i = 100$) leads to more efficient optimizations. However, that improvement comes with a time penalty that is roughly proportional to the fitness improvement. There is no combination of population size, epochs and initial passes that stands out from the data in Figure A2. In the work that follows, we use a population size of 600 for 38 epochs, using 100 initial passes. In our tests, 23.6% of the networks with these hyperparameters exceeded our target fitness of 12,000, albeit with a high third quartile training time of 11.6 minutes on the Pi5.

This is clearly a more challenging problem than *Easy*, but with a reasonable amount of training time, we are able to train effective SNN's. The best network trained during this experiment achieved a testing fitness of 4 minutes, 58 seconds, and the training run took 27.45 minutes on a 2019 vintage Dell workstation.

In terms of expectations, of the roughly 5800 independent runs in this experiment, nearly a fifth (20.1%) exceeded the testing goal of 12,000 timesteps.

Appendix C. Parameters for *Hard*

Once again, we performed a parameter exploration, and after an initial evaluation, we focused on the same parameter settings as in Section B. For the tests where the product of population size and epochs was 60,000, we performed 100 independent runs. For the remaining tests, we performed 250 independent runs. We display the results in Figure A3.

In terms of fitness, the tests where the product of population size and epochs = 60,000 demonstrated the best performance. Within those tests, the tests where population size equals 1000 produced the best networks as a collection, with a median fitness of 5,798 timesteps, and the third quartile fitness of 8,147 timesteps. These runs took significantly longer than the runs with fewer epochs, with the median time being 12.3 minutes on the Pi5, and the third quartile time being 21.2 minutes.

When scaling the number of epochs back to smaller values, the tests with a population size of 1000 have poor median networks (1,785 timesteps for 23 epochs), but their third quartile networks

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 December 2024 doi:10.20944/preprints202412.0532.v1

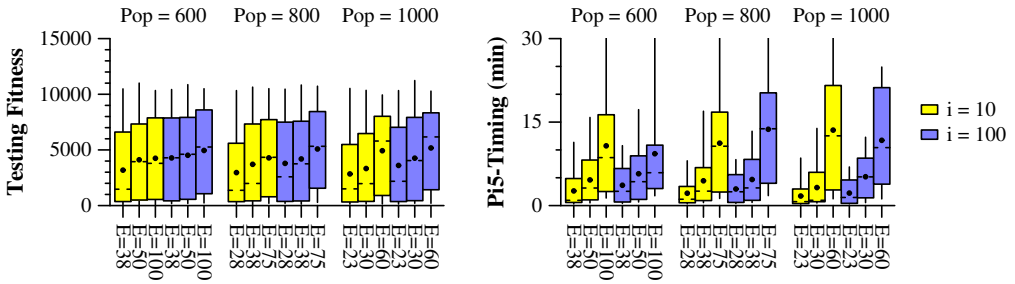


Figure A3. Experiment to determine hyperparameter settings for the *Hard* benchmark.

show good testing fitness (6,242 timesteps) with a reasonable third quartile training time (3.1 minutes on the Pi5). That compares favorably to the tests with a population size of 600 and 38 epochs, whose third quartile testing fitness is 17% higher (7,324 timesteps), but whose third quartile training time is 85% higher (5.8 minutes on the Pi5).

Therefore, a reasonable optimization strategy is to use the higher population size of 1000, perform 100 passes to create the initial population ($i = 100$), and train as long as it is convenient. In terms of expectations, of the roughly 3600 runs, 7.6% achieved the target testing performance of at least 9,000 timesteps.

Appendix D. Parameters for *Hardest*

We performed a similar exploration for the *Hardest* benchmark, broadening the search to include populations of 400 and 500 networks, but using the same products of population size and epochs. Because all five timing graphs were very similar, we only show the timing graph for a population size of 600.

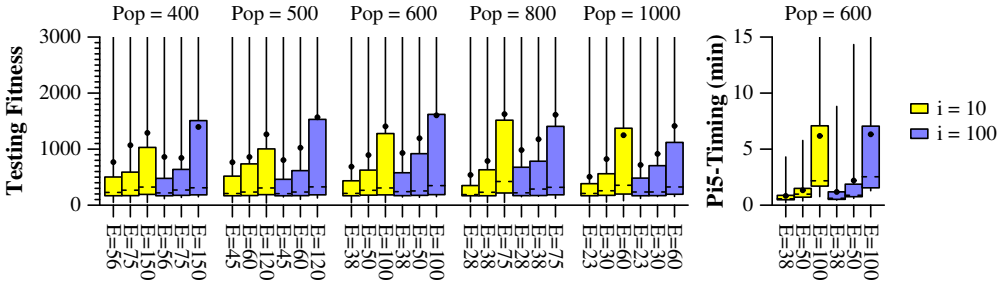


Figure A4. Experiment to determine hyperparameter settings for the *Hardest* benchmark.

A strong conclusion from Figure A4, in comparison with Figure A3, is that the *Hardest* benchmark is indeed harder than the *Hard* benchmark, at least when using EONS for training. For example, the average third quartile testing fitness of all of the runs in Figure A3 is 7,336 timesteps. For the runs in Figure A4, the average testing fitness is 777 timesteps, a reduction of 89.5%. The training times are much faster, because training time is affected by the fitness.

In terms of training parameters, a population size of 600 appears best, and as in Section C above, we advocate to train for as long as is convenient. Performing 100 initial passes to create the initial population ($i = 100$) is advantageous to generating better networks when the population size is 600 or less. When the popuation size is 800 and 1,000, using just 10 passes generates networks with better third quartile fitness. We suspect that with these large population sizes and the difficulty of the problem, using fewer passes increases the diversity of the initial population, which allows for more effective training.

In terms of expectations, of roughly 7900 optimizations, 4.7% of them result in networks that exceed our target of 6,000 timesteps.

 doi:10.20944/preprints202412.0532.v1

Appendix E. Summary

We summarize the results of these tests in Table 2 in the body of the paper, which gives recommended training parameters for each benchmark.

References

1. Roy, K.; Jaiswal, A.; Panda, P. Towards spike-based machine intelligence with neuromorphic computing. *Nature* **2019**, *575*, 607 – 617. doi:10.1038/s41586-019-1677-2.
2. Stroobants, S.; Depeyrous, J.; De Croon, G. Design and implementation of a parsimonious neuromorphic PID for onboard altitude control for MAVs using neuromorphic processors. International Conference on Neuromorphic Computing Systems (ICONS). ACM, 2022.
3. Plank, J.S.; Rizzo, C.; Shahat, K.; Bruer, G.; Dixon, T.; Goin, M.; Zhao, G.; Anantharaj, J.; Schuman, C.D.; Dean, M.E.; Rose, G.S.; Cady, N.C.; Van Nostrand, J. The TENNLab Suite of LIDAR-Based Control Applications for Recurrent, Spiking, Neuromorphic Systems. 44th Annual GOMACTech Conference; , 2019.
4. Rizzo, C.P.; Schuman, C.D.; Plank, J.S. Neuromorphic Downsampling of Event-Based Camera Output. NICE: Neuro-Inspired Computational Elements Workshop. ACM, 2023. doi:10.1145/3584954.3584962.
5. Barto, A.G.; Sutton, R.S.; Anderson, C.W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* **1983**, *SMC-13*, 834–846. https://doi.org/10.1109/TSMC.1983.6313077.
6. Anderson, C.W. Learning to control an inverted pendulum using neural networks. *Control Systems Magazine* **1989**, *9*, 31–37.
7. Florian, R.V. Correct equations for the dynamics of the cart-pole system. Center for Cognitive and Neural Studies (Coneural), Romania, 2007.
8. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym. arXiv:1606.01540, 2016.
9. Kumar, S. Balancing a CartPole System with Reinforcement Learning - A Tutorial. *CoRR* **2020**, *abs/2006.04938*.
10. Galarnyk, M.; Mika, S. An Introduction to Reinforcement Learning with OpenAI Gym, RLlib, and Google Colab. https://www.anyscale.com/blog/an-introduction-to-reinforcement-learning-with-openai-gym-rllib-and-google, 2021.
11. Ding, Y.; Zhang, Y.; Zhang, X.; Chen, P.; Zhang, Z.; Yang, Y.; Cheng, L.; Mu, C.; Wang, M.; Xiang, D.; Wu, G.; Zhou, K.; Yuan, Z.; Liu, Q. Engineering Spiking Neurons Using Threshold Switching Devices for High-Efficient Neuromorphic Computing. *Frontiers in Neuroscience* **2022**, *15*. doi:10.3389/fnins.2021.786694.
12. Plank, J.S.; Schuman, C.D.; Bruer, G.; Dean, M.E.; Rose, G.S. The TENNLab Exploratory Neuromorphic Computing Framework. *IEEE Letters of the Computer Society* **2018**, *1*, 17–20. doi:10.1109/LOCS.2018.2885976.
13. Schuman, C.D.; Plank, J.S.; Parsa, M.; Kulkarni, S.R.; Skuda, N.; Mitchell, J.P. A Software Framework for Comparing Training Approaches for Spiking Neuromorphic Systems. IJCNN: The International Joint Conference on Neural Networks, 2021, pp. 1–10.
14. Parsa, M.; Mitchell, J.P.; Schuman, C.D.; Patton, R.M.; Potok, T.E.; Roy, K. Bayesian-Based Hyperparameter Optimization for Spiking Neuromorphic Systems. IEEE International Conference on Big Data, 2019, pp. 4472–4478.
15. Schuman, C.D.; Disney, A.; Singh, S.P.; Bruer, G.; Mitchell, J.P.; Klibisz, A.; Plank, J.S. Parallel Evolutionary Optimization for Neuromorphic Network Training. Proceedings of the Workshop on Machine Learning in High Performance Computing Environments. IEEE Press, 2016, pp. 36–46.
16. Schuman, C.D.; Plank, J.S.; Bruer, G.; Anantharaj, J. Non-Traditional Input Encoding Schemes for Spiking Neuromorphic Systems. IJCNN: The International Joint Conference on Neural Networks; , 2019; pp. 1–10. doi:10.1109/IJCNN.2019.8852139.
17. Plank, J.S.; Dent, K.E.M.; Gullett, B.; Rizzo, C.P.; Schuman, C.D. The RISP Neuroprocessor - Open Source Support for Embedded Neuromorphic Computing. IEEE International Conference on Rebooting Computing (ICRC); , 2024.
18. Schuman, C.D.; Potok, T.E.; Patton, R.M.; Birdwell, J.D.; Dean, M.E.; Rose, G.S.; Plank, J.S. A Survey of Neuromorphic Computing and Neural Networks in Hardware. arXiv:1705.06963, 2017.

 doi:10.20944/preprints202412.0532.v1

19. Cabessa, J.; Siegelmann, H.T. The Super-Turing Computational Power of Plastic Recurrent Neural Networks. *International Journal of Neural Systems* **2014**, *24*.
20. Akopyan, F.; Sawada, J.; Cassidy, A.; Alvarez-Icaza, R.; Arthur, J.; Merolla, P.; Imam, N.; Nakamura, Y.; Datta, P.; Nam, G.J.; Taba, B.; Beakes, M.; Brezzo, B.; Kuang, J.B.; Manohar, R.; Risk, W.P.; Jackson, B.; Modha, D.S. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2015**, *34*, 1537–1557.
21. Lehmann, H.M.; Julian, J.H.; Grassmann, C.; Issakov, V. Leaky integrate-and-fire neuron with a refractory period mechanism for invariant spikes. 17th conference on Ph.D research in microelectronics and electronics (PRIME), 2022, pp. 365–368. doi:10.1109/PRIME55000.2022.9816777.
22. Davies, M.; Srinivasa, N.; Lin, T.H.; Chinya, G.; Cao, Y.; Choday, S.H.; Dimou, G.; Joshi, P.; Imam, N.; Jain, S.; Liao, Y.; Lin, C.K.; Lines, A.; Liu, R.; Mathaikutty, D.; McCoy, S.; Paul, A.; Tse, J.; Venkataramanan, G.; Weng, Y.H.; Wild, A.; Yang, Y.; Wang, H. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* **2018**, *38*, 82–99. doi:10.1109/MM.2018.112130359.
23. Azghadi, M.R.; Moradi, S.; Fasnacht, D.B.; Ozdas, M.S.; Indiveri, G. Programmable spike-timing dependent plasticity learning circuits in neuromorphic VLSI architectures. *ACM Journal on Emerging Technologies in Computing Systems* **2015**, *12*.
24. Masquelier, T.; Guyonneau, R.; Thorpe, S.J. Spike Timing Dependent Plasticity Finds the Start of Repeating Patterns in Continuous Spike Trains. *PLoS ONE* **2008**, *3*. doi:10.1371/journal.pone.0001377.
25. Schuman, C.D.; Rizzo, C.; McDonald-Carmack, J.; Skuda, N.; Plank, J.S. Evaluating Encoding and Decoding Approaches for Spiking Neuromorphic Systems. International Conference on Neuromorphic Computing Systems (ICONS). ACM, 2022, pp. 1–10.
26. Dupeyroux, J. A Toolbox for Neuromorphic Sensing in Robotics. arXiv:2103.02751v1, 2021.
27. Tang, G.; Kumar, N.; Yoo, R.; Michmizos, K. Deep Reinforcement Learning with Population-Coded Spiking Neural Network for Continuous Control. Conference on Robot Learning (CoRL); , 2020.
28. Plank, J.S.; Schuman, C.D.; Rizzo, C.P. Framework-Open: Open-source part of the TENNLab Exploratory Neuromorphic Framework. https://github.com/TENNLab-UTK/framework-open, 2024.
29. Plank, J.S.; Zheng, C.; Gullett, B.; Skuda, N.; Rizzo, C.; Schuman, C.D.; Rose, G.S. The Case for RISP: A Reduced Instruction Spiking Processor. arXiv:2206.14016, 2022, [2206.14016].
30. Dent, K.; Gullett, B. Neuromorphic FPGA. https://github.com/TENNLab-UTK/fpga, 2024.
31. Schuman, C.D.; Mitchell, J.P.; Patton, R.M.; Potok, T.E.; Plank, J.S. Evolutionary Optimization for Neuromorphic Systems. NICE: Neuro-Inspired Computational Elements Workshop, 2020.
32. Li, C.; Chen, R.; Moutafis, C.; Furber, S. Robustness to Noisy Synaptic Weights in Spiking Neural Networks. IJCNN: The International Joint Conference on Neural Networks, 2020.
33. Fonseca Guerra, G.A.; Furber, S.B. Using stochastic spiking neural networks on SpiNNaker to solve constraint satisfaction problems. *Frontiers in Neuroscience* **2017**, *11*, 714.
34. Dampfhoffer, M.; Lopez, J.M.; Mesquida, T.; Valentian, A.; Anghel, L. Improving the Robustness of Neural Networks to Noisy Multi-Level Non-Volatile Memory-based Synapses. IJCNN: The International Joint Conference on Neural Networks, 2023. doi:10.1109/IJCNN54540.2023.10191804.
35. Schaefer, A.M.; Udluft, S.; Zimmermann, H.G. A Recurrent Control Neural Network for Data Efficient Reinforcement Learning. IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007. doi:10.1109/ADPRL.2007.368182.
36. Nagendra, S.; Podila, N.; Ugarakhod, R.; George, K. Comparison of Reinforcement Learning algorithms applied to the Cart Pole problem. International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017, pp. 26–32. doi:10.1109/ICACCI.2017.8125811.
37. Chang, O.; Kwiatkowski, R.; Chen, S.; Lipson, H. Agent Embeddings: A Latent Representation for Pole-Balancing Networks. Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. ACM, 2019, pp. 656–664. doi:10.5555/3306127.3331753.
38. Buvanesh, P.V. DDPG Agent to Swing Up and Balance Cart-Pole System. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)* **2021**, *4*.
39. Variengien, A.; Pontes-Filho, S.; Glover, T.E.; Nichele, S. Towards Self-organized Control: Using Neural Cellular Automata to Robustly Control a Cart-pole Agent. *Innovations in Machine Intelligence (IMI)* **2021**, *1*, 1–14. doi:10.54854/imi2021.01.

 doi:10.20944/preprints202412.0532.v1

40. Rafe, A.W.; Garcia, J.A.; Raffe, W.L. Exploration Of Encoding And Decoding Methods For Spiking Neural Networks On The Cart Pole And Lunar Lander Problems Using Evolutionary Training. IEEE Congress on Evolutionary Computation (CEC), 2021. doi:10.1109/CEC45853.2021.9504921.
41. Shin, S.S.; Kim, Y.H. A surrogate model-based genetic algorithm for the optimal policy in cart-pole balancing environments. Genetic and Evolutionary Computation Conference, 2022, pp. 503–505.
42. Tran, D.D.; Le, T.T.; Duong, M.T.; Phan, M.Q.; Nguyen, M.S. FPGA Design for Deep Q-Network: a case study in cartpole environment. International Conference on Multimedia Analysis and Pattern Recognition (MAPR). IEEE, 2022. doi:10.1109/MAPR56351.2022.9925007.
43. Han, B.C.; Kim, H.C.; Kang, M.J. Comparison of value-based Reinforcement Learning Algorithms in Cart-Pole Environment. *International Journal of Internet, Broadcasting and Communication* **2023**, *15*, 166–175. doi:10.7236/IJIBC.2023.15.3.166.
44. Xu, J. How to Beat the CartPole Game in 5 Lines – A Simple Solution without Artificial Intelligence. TowardsDataScience, https://towardsdatascience.com/how-to-beat-the-cartpole-game-in-5-lines-5ab4e738c93f, 2021.