Spark: Modular Spiking Neural Networks

Nowadays, neural networks act as a synonym for artificial intelligence. Present neural network models, although remarkably powerful, are inefficient both in terms of data and energy. Several alternative forms of neural networks have been proposed to address some of these problems. Specifically, spiking neural networks are suitable for efficient hardware implementations. However, effective learning algorithms for spiking networks remain elusive, although it is suspected that effective plasticity mechanisms could alleviate the problem of data efficiency. Here, we present a new framework for spiking neural networks — *Spark* — built upon the idea of modular design, from simple components to entire models. The aim of this framework is to provide an efficient and streamlined pipeline for spiking neural networks (performance summary is shown subfigures a & b).

We showcase this framework by solving the sparse-reward cartpole problem without any auxiliary readout models and using a simple plasticity mechanism. Our solution consists of a simple A vs B model; activity on A is biased to inhibit activity on B and vice versa (subfigure c). Note that, *Spark* allows us to define self-recurrent complex architectures extremely easily. We trained 25 agents for a total of 1000 episodes each, followed by a testing of 250 episodes (subfigures d, e, f & g). The majority of our agents (16 out of 25) are capable of solving the problem within the first 50-80 episodes, while the remaining agents showed a slower training process; we allow agents to continue training for 1000 episodes to test the stability of our solution.

As shown with the cartpole example, we believe that a framework compatible with traditional ML pipelines may accelerate research in the area, specifically for continuous and unbatched learning, akin to the one animals exhibit.

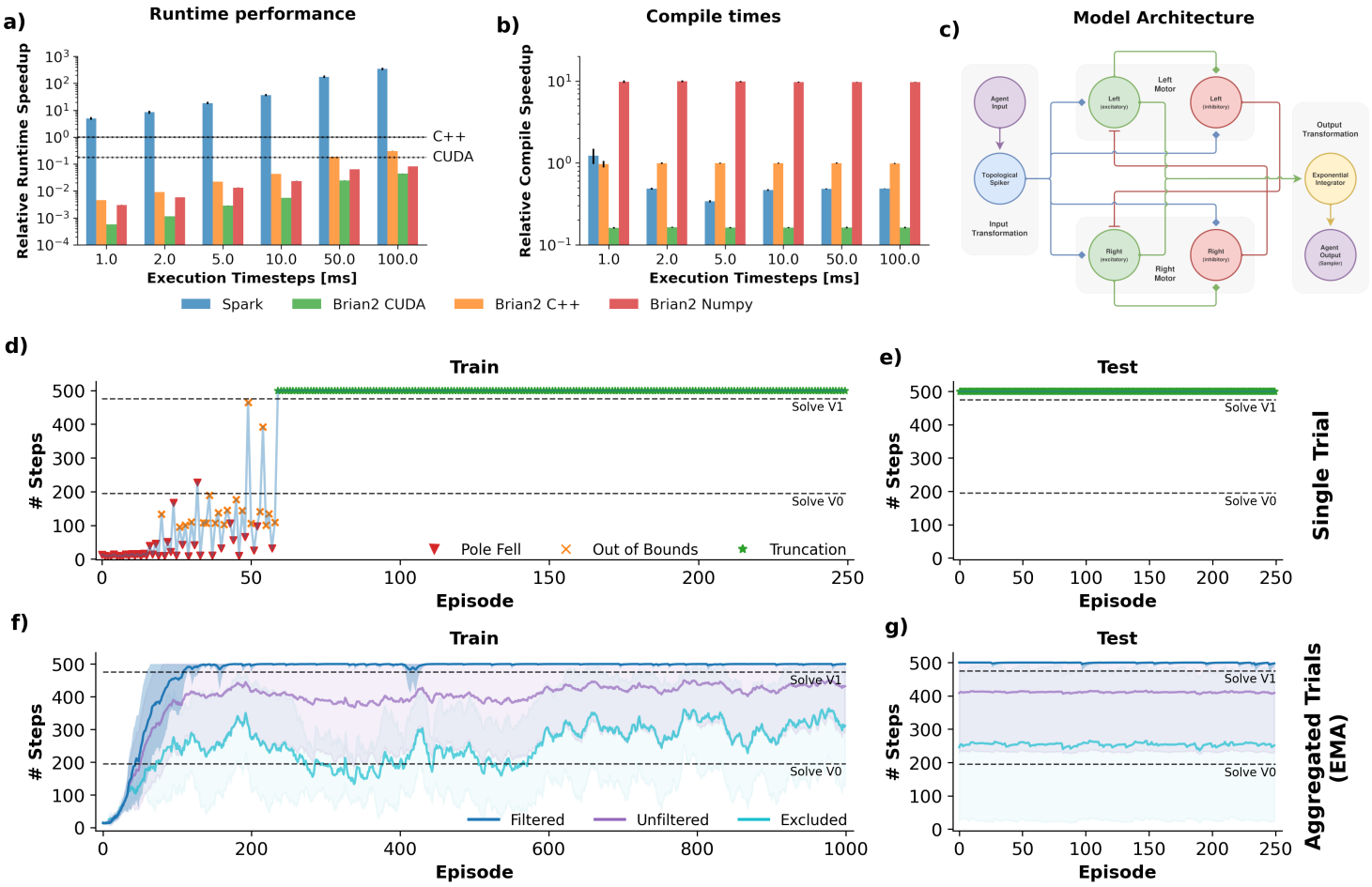

Figure 1: (a) Average relative speedup of 1 seconds simulations, 10 network instantiations, 5 runs per instantiation. A network with 8 self-recurrent neuron pools with 1024 neurons and non-trivial connectivity among them was used for this benchmark. Speedups are normalized with respect to the C++ single execution. Execution timesteps indicate how many timesteps each model computes before a new input signal is presented to the network and a new output signal is registered from the network. (b) Average compilation times for the models in (a). (c) Summary of the model architecture used to solve the cartpole problem. The model consists of a simple A vs B architecture; activity on A inhibits activity on B and vice versa. Agent output consists of a traced "measurement" of the activity of each excitatory subpopulation. (d) Training trajectories of a single SNN agent for the sparse-reward cartpole problem, with episode end annotations. (e) Test performance for the agent in (d) after 1000 episodes. (f) Exponential moving average for the training trajectories of 25 SNN agent for the sparse-reward cartpole problem. Filtered consist of a subset of 16 agents that managed to stabilize within the first 50-80 episodes. Unfiltered is the entire agent population. Excluded is the remaining 9 agents for which no stabilization was observed in the first 500 episodes. (g) Test performance for the agents in (f) after 1000 training episodes.