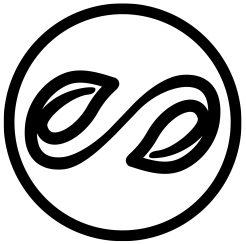
WaspNet.jl

A Julian Spiking Neural Network Simulator

Sam Buercklin[1*], Charles Freeman[1], John L. Sherwood[1]



We present `WaspNet.jl`, a framework intended for spiking neural network (SNN) simulations for computational neuroscience and machine learning. `WaspNet.jl` provides efficient implementations for SNN primitives and common neuronal models. This framework is robust to the neuron model, supporting both spiking and non-spiking neurons, allowing for novel hybrid topologies. We exhibit GPU acceleration with `CuArrays.jl` and demonstrate capabilities of `WaspNet.jl` with a variety of example experiments.

[1]LeafLabs, Cambridge, MA
[*]buercklin@leaflabs.com

```
Pkg.add("WaspNet")
```

https://github.com/leaflabs/WaspNet.jl

Spiking Neural Networks (SNNs)

A SNN functions similar to an artificial neural network (ANN) with layers comprising neurons and weights and signals passed between layers. By comparison to ANNs, an SNN:

- Communicates through discrete, temporally sparse binary messages called spikes.
- Solves an ordinary differential equation to determine when to emit a spike, instead of computing a scalar function of neuron inputs (e.g. ReLU, tanh).
- Encodes information with spike timing patterns and neuron firing rates.

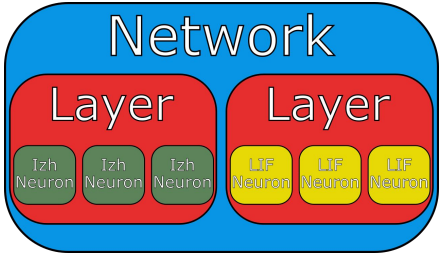


Figure 1: Graphical representation of WaspNet.jl abstractions.

In a `WaspNet.jl` Network, layers possess a set neurons of a single type. Networks contain sets of various layers. There are no restrictions on the inter-layer neuron types, despite potentially very different modes of operation.

WaspNet.jl Abstractions

`Neuron`: Black box mapping an input and time to an output. Typically solves an ODE with extra rules for spike emission, although non-spiking neurons can this description as well. Example neuron models include the Leaky Integrate & Fire (LIF) [1] or Izhikevich [2] models.

`Layer`: Collection of neurons and associated input weights for other layers. Computes neuron inputs and stores neuron outputs.

`Network`: Orchestrates communication between constituent layers and handles network input.

Example Implementation: The code below builds a 4 layer, 4-neuron-wide network and drives it with random inputs.

```
N_neurons = 4
N_in = N_neurons

neurons = [WaspNet.Izh() for _ in 1:N_neurons]

layers = Array{Layer, 1}()
for j in 1:4
    weights = randn(N_neurons, N_neurons)
    push!(layers, Layer(neurons, weights))
end
network = Network(layers, N_in)

f(t) = randn(N_in)
result = simulate!(network, f, 0.001, 5.)
```

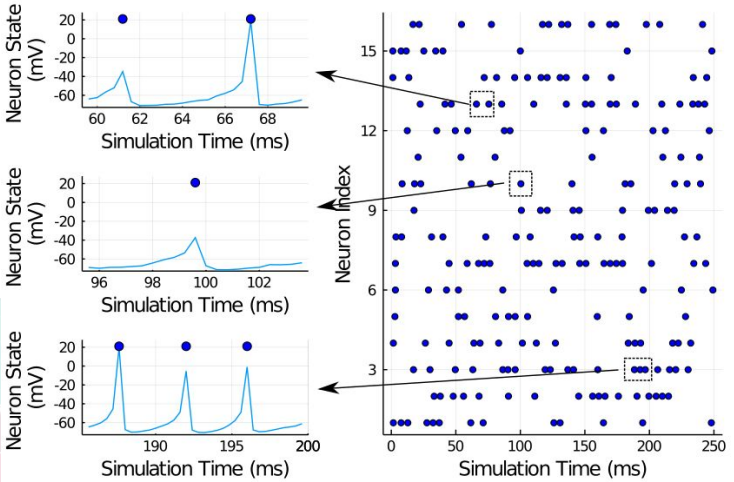


Figure 2: Results of the 4 Layer, 16 Neuron simulation.

Left: Three excerpt traces of the internal state of three neurons are shown, with spike times denoted by blue markers. Typically the internal state is **not** communicated in the network, just spike events as indicated by the markers.

Right: A raster event plot indicating all of the times that each neuron spiked. Markers on the LHS represent the related boxed markers or spikes on the RHS. By analyzing the event plot, for example by calculating average spike rates in window of time, we can use this SNN to solve problems similar to an ANN.

Arbitrary Connectivity and Heterogeneous Networks

SNNs are frequently recurrent and nonlinearly connected. `WaspNet.jl` supports arbitrary connectivity and recurrence by specifying connected layers and weights between any two layers, including themselves.

Networks comprise as many neuron types as there are layers, similar to biology: different neurons specialized to specific behaviors can be inserted where necessary, regardless of their internal dynamics.

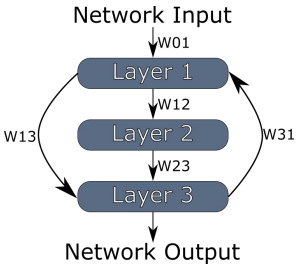


Figure 3: An example non-feed-forward SNN topology

The code below generates a network similar to the diagram to the left. Input connected layers and weights for each connection are specified in the same order. If no connections are given, the layer and network assume a feed-forward structure.

```
# Wij is a weight matrix from layer i to layer j
# N_in is the number of inputs to drive the network
L1 = Layer(neurons1, conns = [0, 3], W = [W01, W31])
L2 = Layer(neurons2, W12)
L3 = Layer(neurons3, conns = [1, 2], W = [W13, W23])
net = Network([L1, L2, L3], N_in)
```

Neuronally-wrapped Neurons

By wrapping neurons within neurons, we can augment existing models with new features, such as:

- Automatically filtering neuronal spike trains to calculate spike rates
- Introducing multiple time scales to a problem
- Pre-processing synthetic data to be more biologically plausible

In Figure 5 we show code for adding noise and decay to a generic neuron model, helping the simulation mimic experimental data.

Odorant Discrimination using Reservoir Computing

We have used WaspNet.jl to implement a SNN with a single recurrent layer (often called a reservoir computer (RC)) capable of classifying multiple odorants. This network is driven with a novel odorant dataset [3] to generate event data, and the average spiking rate of each neuron over time is used to classify the odorants.

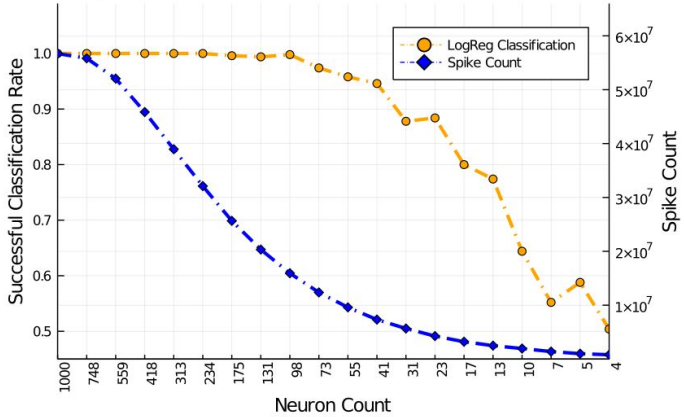
Specifically, we are interested in the efficacy of a RC as a function of neuron count. The power consumption of a SNN can be partially modeled as a function of emitted spikes, which are temporally sparse communications.

To optimize the power consumption, we begin with large networks and prune them by removing neurons with low activity [Figure 4, right]. We developed pruning methods to this end, and found that classification rate can be preserved while suppressing network spike count.

Our pruning methods require specifying indices of the layers and neurons to remove.

```
new_net = prune(in_net, layer_idxs, neuron_idxs)
```

Figure 4: Pruning Low-Activity Neurons



Noisy Neuron-wrapped Neurons

This code creates a new neuron type meant for wrapping an existing neuron. This wrapper injects noise into the output signal of the inner neuron and adds an exponential decay to the spike, akin to a convolution of the spike train with an exponential kernel. This wrapper works with any existing neuron which outputs binary spikes.

```
mutable struct NoiseDecay<:AbstractNeuron
    inner_neuron
    output
    state
end
function WaspNet.update!(neuron::NoiseDecay, input, dt, t)
    new_output = update!(neuron.inner_neuron, input, dt, t)
    neuron.output = new_output
    if new_output == 1
        neuron.state = 15. + randn()
    else
        neuron.state = neuron.state*0.8 + randn()
    end
    return new_output
end

noisy = NoiseDecay(WaspNet.Izh(), 0., 0.)
result = simulate!(noisy, (t) -> 0.1, 0.001, 0.08,
    track_state=true)
```

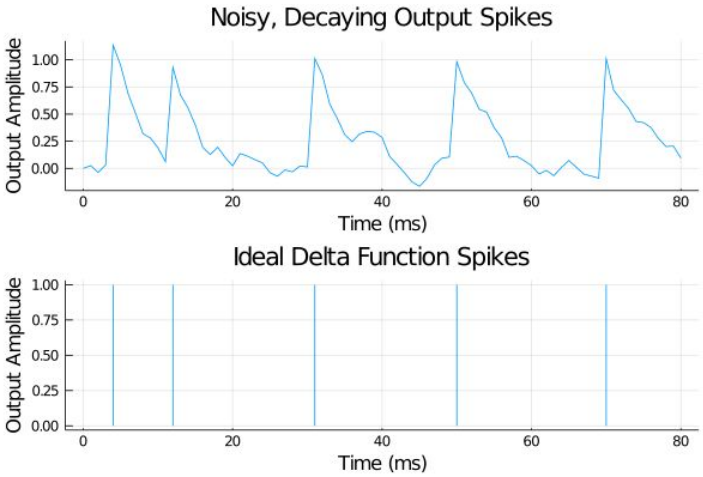


Figure 5: Noisy synthetic spikes vs ideal neuron spikes.

Here, we present noisy synthetic data using the Noisy Neuron wrapper (top), as well as its ideal counterpart (bottom). The former is interesting as a numerical representation of neuronal action potentials, while the latter is useful for low-power machine learning and studying SNNs as dynamical systems.

Ongoing Work

– CUDA support
  - Faster input calculations with accelerated matrix ops
  - Converting neuron updates to CUDA kernels for parallelization
– Unify “Training” abstraction
  - Biologically inspired methods (STDP)
  - Gradient-like methods

Closing Remarks

We have presented `WaspNet.jl` along with sample code and simulations. `WaspNet.jl` is available on the Julia Package Registry under `WaspNet` as well as GitHub. We welcome any and all feedback.

Citations

[1] Abbott, Larry F. "Lapicque's introduction of the integrate-and-fire model neuron (1907)." *Brain research bulletin* 50.5-6 (1999): 303-304.
[2] Izhikevich, Eugene M. "Simple model of spiking neurons." *IEEE Transactions on neural networks* 14.6 (2003): 1569-1572.
[3] Saha, Debajit, et al. "A spatiotemporal coding mechanism for background-invariant odor recognition." *Nature neuroscience* 16.12 (2013): 1830-1839.

Acknowledgments

Barani Raman (Washington University St. Louis)
Debajit Saha, Xander Farnum (Michigan State University)

This material is based upon work supported by the Defence Advanced Research Projects Agency (DARPA) under Agreement No. HR00111990036; DARPA-PA-18-02-03 Microscale Bio-mimetic Robust Artificial Intelligence Networks (AIE; μBRAIN)