BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

Implementing Spike-Timing Dependent Plasticity on the Izhikevich model of a Spiking Neuron

ADITYA AHUJA

Under the supervision of Dr. Basabdatta Sen Bhattacharya

December 2, 2019

Abstract

Spike-Timing Dependent Plasticity (STDP)is a learning rule that explains how differences in spike timings between Pre and Post synaptic spikes leads to changes in synaptic weights. STDP is important as it is the major learning rule through which a network of spiking neurons learn to associate their synaptic weights with external reward. In this project, we implement the STDP learning rule on SpineCreator - a graphical framework for modeling and analysing Neuronal models.

Key words: STDP, SpineCreator, Plasticity, Izhikevich Neuron

CONTENTS

1 THE IZHIKEVICH NEURON

The Izhikevich neuron was introduced by Eugene M. Izhikevich in 2003 [2]. While not based on biophysical parameters, it is still able to model the various spiking patterns that have been observed in biological neurons.

At its core, the Izhikevich neuron is described by four parameters : a,b,c,d that help model important neuron biophysical variables. The equations that describe these variables are:

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{1.1}$$
$$u' = a(bv - u) \tag{1.2}$$
$$v = c \tag{1.3}$$
$$u = c + d \tag{1.4}$$

where equations (1.3),(1.4) hold if $v >= 30mV$.

2 STDP

As explained earlier, STDP is the proposed theory which tries to relate temporal spike differences to changes in synaptic weights.

It is an extension of the Hebbian learning rule, as it imposes a time based ordering on the learning. A Pre-synaptic spike followed by a Post-synaptic spike strengthens the synapse. Conversely, A Post-synaptic spike followed by a Pre-synaptic one weakens the synapse.

We quantify these relations with the following equations :

$$\Delta G = +A_+ * e^{(-x/\tau_+)} \quad \text{if } x >= 0 \tag{2.1}$$
$$\Delta G = -A_- * e^{(x/\tau_-)} \quad \text{otherwise} \tag{2.2}$$

Thus by considering the relative timing of the spikes, the STDP learning rule is able to determine the update that should be done to the synaptic weights.

3 SpineCreator

3.1 Introduction

SpineCreator [1] is a Graphical Framework for the creation, simulation and visualization of layered spiking neurons.

The framework consists of the following sections :-

1. Components
2. Network
3. Experiments
4. Graphs
5. Visualizations

Further subsections explore each section of SpineCreator in brief, explaining its utility. We begin our explanation in a bottom-up manner, explaining the basic sections first, and ending with the later sections.

3.2 Components

The Components sections contains the interface needed to create various network components that would be used as basic blocks for further network layers. All the components can be classified into the following three categories :

1. Neuron Body (NB)
2. Post-Synapse (PS)
3. Weight Update (WU)

Each of these components can be used to build a different part of a spiking neural network.

For example, the Neuron Body component can be used to set up the basic differential equations that the Neuron should follow and can contain further mathematical constraints between several pre-defined variables and parameters.

Components also help describe "ports", which can be used to send signals back and forth between network components. For example a spike may be sent out from the Post Synapse component through an "Event Port", and may be received in the Neuron Body component through the corresponding receive port.

3.3 NETWORK

The Network section contains tools to build the Network structure using previously defined components. We can use the Neuron Body components to instantiate populations of neurons. The Post Synapse and Weight Update components can be used to form the synapses between the Neuron Bodies.

Once the components have been suitably instantiated, we can set up the values for the parameters and the state variables defined withing the components previously. We can also map ports from one component to another and define additional connections between components.

3.4 EXPERIMENTS

The Experiments section contains the tools needed to simulate the previous defined network. Different experiments can be set up and their corresponding information, such as the simulation duration and time-step can be specified.

Further one can define the Input and Output to the Network being simulated. The Output specified here must consist of output ports defined previously in the components section. Once specified, these outputs are monitored for the duration of the experiment and later stored in .csv files for further analysis.

External inputs can also be provided to the network components by hosting a server on a local machine.

3.5 GRAPHS

This section can be used for analysis and interpretation of the experiments that were defined in the previous section. It build graphs using the .csv and .log files generated during the simulations.

3.6 FURTHER RESOURCES

A more detailed and practical overview of the SpineCreator framework can be found at the SpineML homepage `http://spineml.github.io/`.

Tutorials that can be found at : `http://spineml.github.io/spinecreator/tutorial/` are also especially helpful.

4 IMPLEMENTATION

4.1 OVERVIEW

The implementation consisted of a the following components :

1. **NB: Izhikevich** - Implements the equations used to define an Izhikevich neuron as described in section 1.
2. **PS: AMPA** - Implements the AMPA postsynapse.
3. **PS: GABA** - Implements the GABA postsynapse.
4. **WU: STDP** - Implements the 5 state model that was used to model STDP on the Izhikevich neurons.

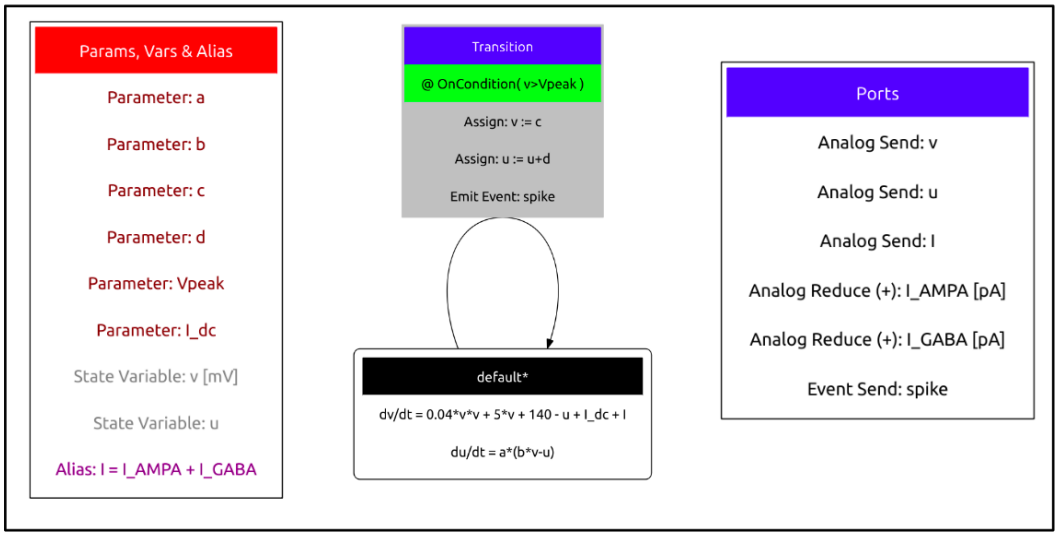

Figure 4.1: SpineCreator implementation of Izhikevich Neuron Body.

Of the above, the non-trivial part was designing the Weight Update STDP component. The rest of this section thus focuses on the design and implementation of the STDP weight update component.

4.2 STATE DESCRIPTIONS

A five state model was developed to model the weight update component of Izhikevich neurons. These states known natively as 'regimes' in SpikeCreator describe the differential equations that a model obeys. The regimes also feature triggers that allow one to program for incoming events, impulses or analog inputs. On encountering such triggers, one can program for state(regime) change or other assignments.

The designed component consists of the following five states:

1. **Initial state**: Initial regime that the denotes the initial state of the component.

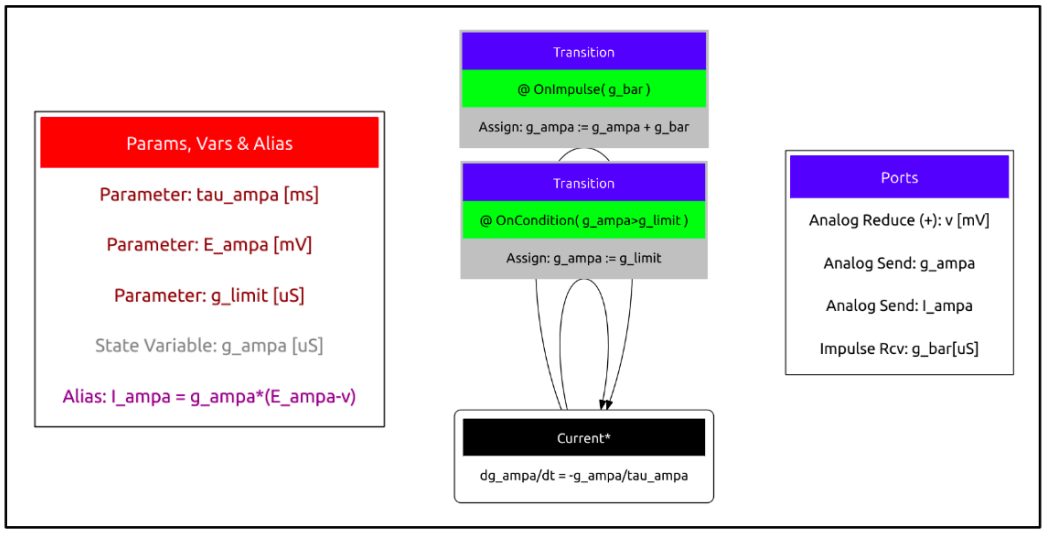

Figure 4.2: SpineCreator implementation of AMPA Postsynapse.

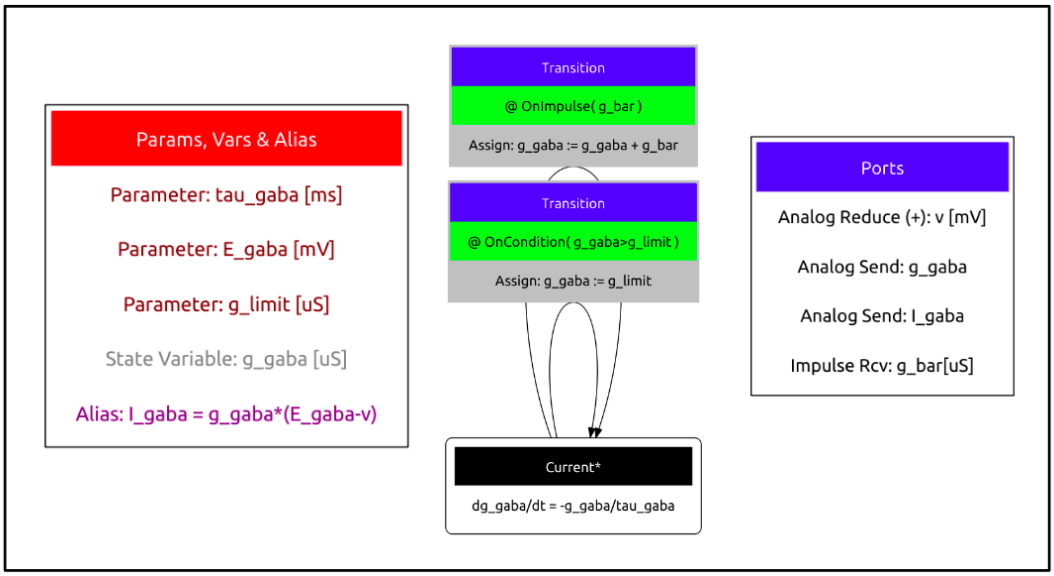

Figure 4.3: SpineCreator implementation of GABA Postsynapse.

2. **Pre state**: State when the component has recently received a 'Pre' spike.
3. **Post state**: State when the component has recently received a 'Post' spike.
4. **Pre-Post state**: State after the component has received the 'Pre' and 'Post' spikes in a short temporal window.
5. **Post-Pre state**: State after the component has received the 'Post' and 'Pre' spikes in a short temporal window.

4.3 Weight Update

When the weight update component reaches the 'Pre-Post' or the 'Post-Pre' regime (State 8/9 in the above algorithm), it calculates the change in synaptic weight and emits it as an impulse to the post-synapse. This change is formally modelled as $\bar{g}$, the change in conductance per spike.

```
[1] Open up event reveiving ports to listen for spike input from
    Pre and Post Synaptic Neurons.

[2] Set up an impluse forwarding port, in order to transfer the
    change in synaptic weight post an update.

[3] Set state to "Initial", and begin listening for both Pre and
    Post Spikes.

[4] In case of Pre Spike, goto 6.
[5] In case of Post Spike, goto 7.

[6]
      [a] Change state to 'Pre' State.
      [b] Note down the time of transition.
      [c] If a Post Spike is encountered goto 8.
      [d] Else go back to "Inital" state, i.e. goto 3.

[7]
      [a] Change state to 'Post' State.
      [b] Note down the time of transition.
      [c] If a Pre Spike is encountered goto 9.
      [d] Else go back to "Inital" state, i.e. goto 3.

[8]
      [a] Change state to 'Pre-Post' State.
      [b] Note down the time of transition.
      [c] Apply the weights update rule.
      [d] Emit the Weight change impulse.
      [e] Go back to 'Initial' state, i.e. goto 3.

[9]
      [a] Change state to 'Post-Pre' State.
      [b] Note down the time of transition.
      [c] Apply the weights update rule.
      [d] Emit the Weight change impulse.
      [e] Go back to 'Initial' state, i.e. goto 3.
```

Figure 4.4: Weight Update State Transition Algorithm [simplified]

4.4 Connections and Ports

On receiving the $\bar{g}$ impulse from the weight update component, the post-synapse updates its conductance g_{syn}, adding the obtained value to it (upto a maximum g_{limit}). Thus, the individual weight update impulses in effect get summed up for each effective update, leading to the overall equation :

$$g_{syn}(t_o) = g_{syn}(t_o - \Delta t) + \Sigma \bar{g}_{syn}$$

where g_{syn} is the membrane conductance at the $t_o{}^{\text{th}}$ time step, Δt is the time per time-step and $\bar{g}_{syn}$ is the conductance update.

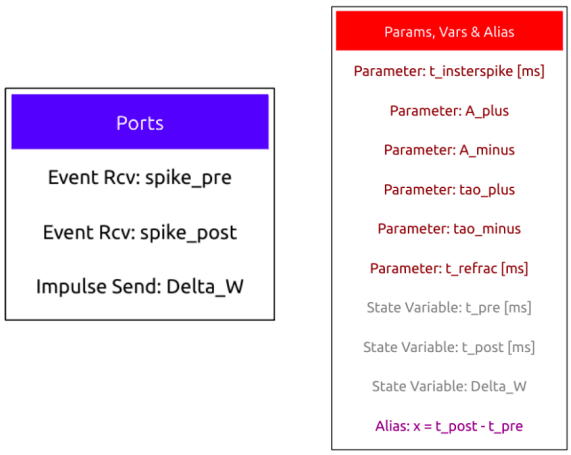

Figure 4.5: Weight Update : Ports, Parameters and Variables.

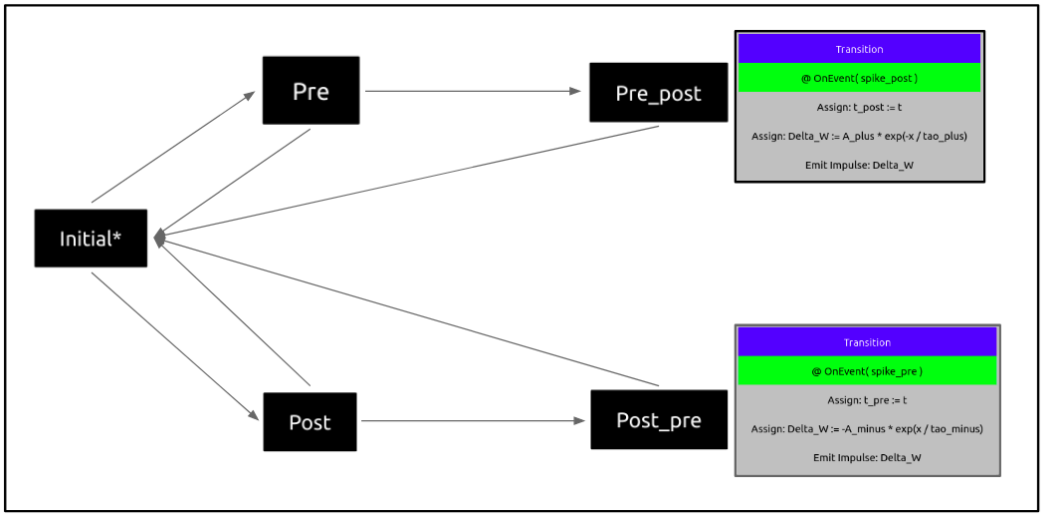

Figure 4.6: Schematic diagram of valid state transitions.

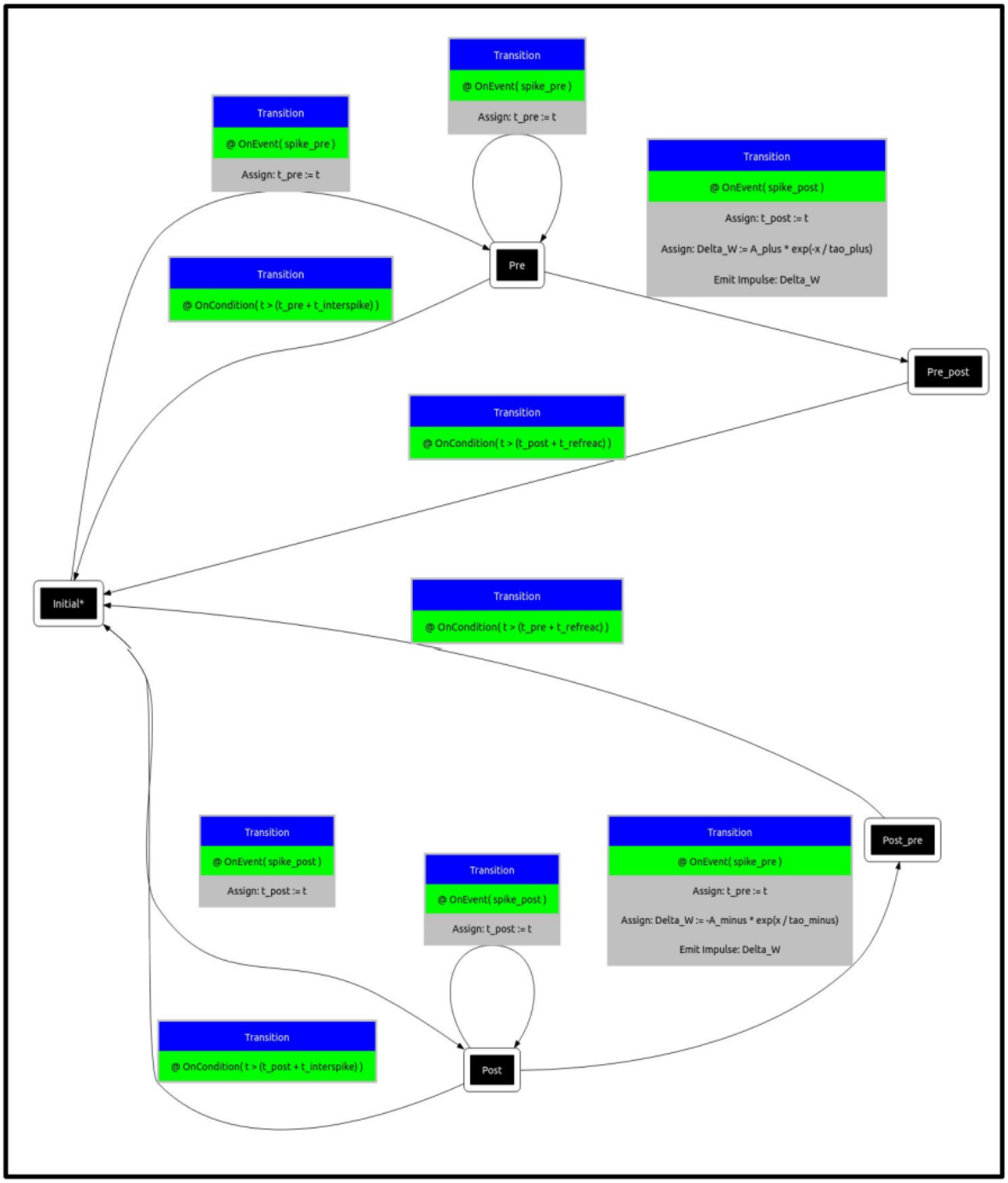

Figure 4.7: Original State Transition graph.

5 TESTING THE IMPLEMENTATION

A simple network was built to test the STDP implementation. It consisted of two Neuron populations that were connected to 4 different poisson spike sources. Each population was connected to an 'Input' source at frequency of 300 Hz, and a noise source of 80 HZ.

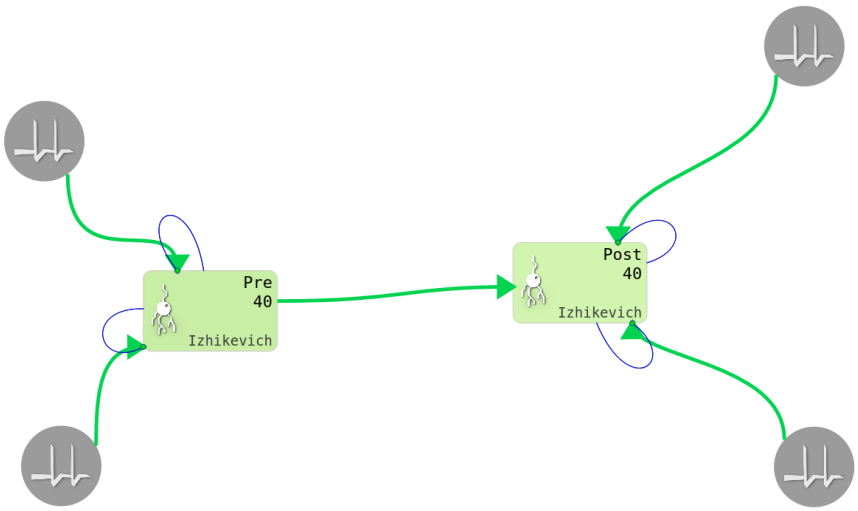

Figure 5.1: Network diagram for verifying STDP

The experimental setup to verify the STDP implementation was as below :

1. `At t = 0000ms: Set both Noise sources at 80 Hz, and the Input sources at 0 Hz.`
2. `At t = 0200ms: Set both the Input sources at 300 Hz.`
3. `At t = 0800ms: Set both the Input sources at 0 Hz.`
4. `At t = 1000ms: Start training poisson spikes to the Neurons through the input sources.`
5. `At t = 1800ms: Stop the training poisson spikes.`
6. `At t = 2000ms: Set only the Pre Input spike to 300 Hz.`
7. `At t = 1000ms: Set all Input sources to 0 Hz.`

At the time of this report, the implementation needs further analysis and testing.

6 Further Work

Further work would focus on the following areas :

1. Robust testing and verification of the implemented STDP algorithm.
2. Introducing Neuromodulation into the STDP algorithm [3].
3. Using the algorithm as a component for modeling biological neural models, such as that of the Basal Ganglia [4], or the Lateral Geniculate Nucleus[5].

7 Acknowledgments

This project would not have been possible with Prof. Basabdatta Sen Bhattacharya, who has constantly mentored and guided me. I would also like to thank Dr. Sebastian James for his various inputs and advice on working with the SpineCreator framework.

References

[1] A. J. Cope, P. Richmond, S. S. James, K. Gurney, and D. J. Allerton. Spinecreator: A graphical user interface for the creation of layered neural models. *Neuroinformatics*, 15(1):25–40, 2017.

[2] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[3] E. M. Izhikevich. Solving the distal reward problem through linkage of stdp and dopamine signaling. *Cerebral cortex*, 17(10):2443–2452, 2007.

[4] B. Sen-Bhattacharya, S. James, O. Rhodes, I. Sugiarto, A. Rowley, A. B. Stokes, K. Gurney, and S. B. Furber. Building a spiking neural network model of the basal ganglia on spinnaker. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):823–836, 2018.

[5] B. Sen-Bhattacharya, T. Serrano-Gotarredona, L. Balassa, A. Bhattacharya, A. B. Stokes, A. Rowley, I. Sugiarto, and S. Furber. A spiking neural network model of the lateral geniculate nucleus on the spinnaker machine. *Frontiers in neuroscience*, 11:454, 2017.