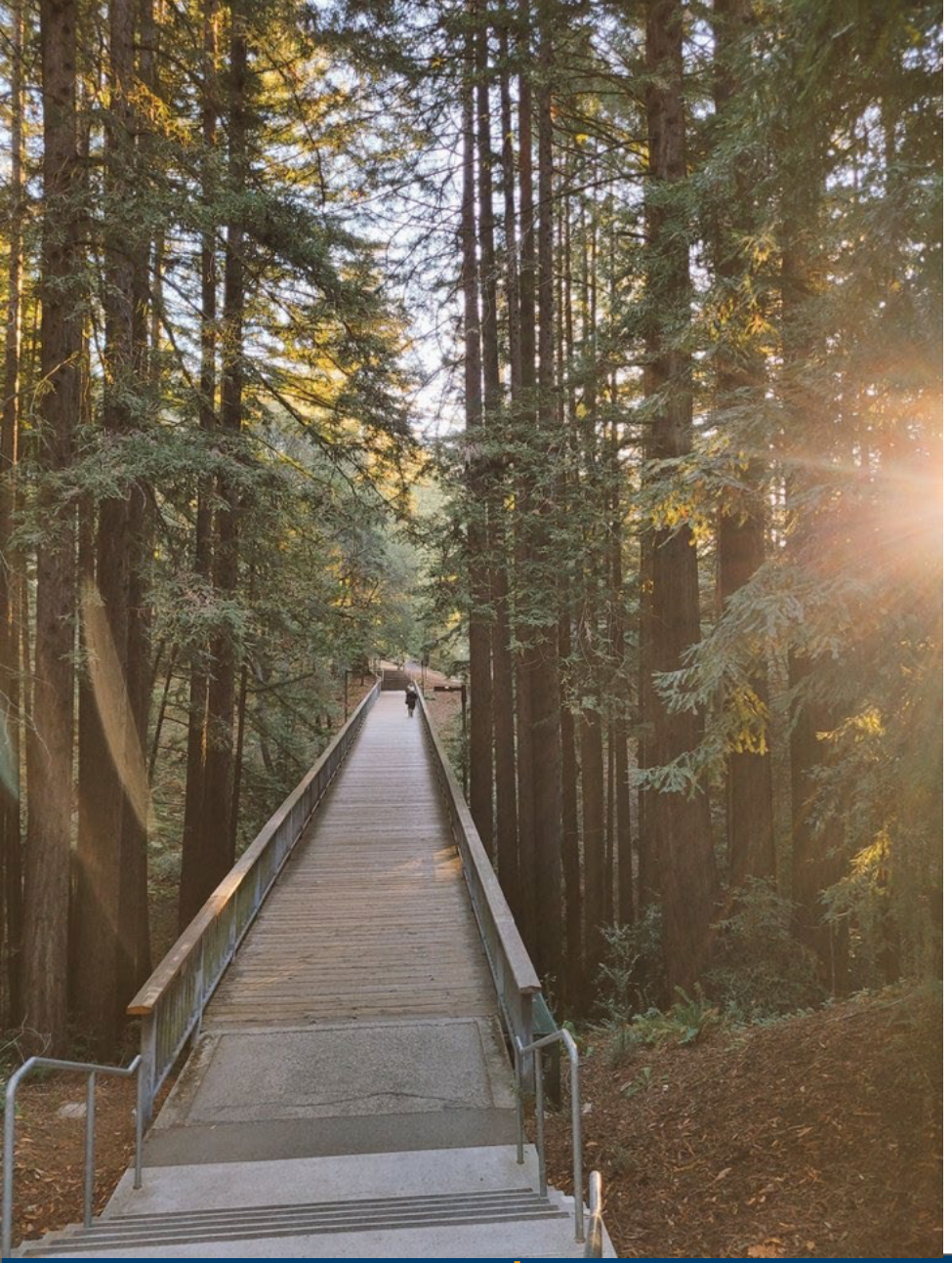


Training Brain-Inspired Spiking Neural Networks Using Lessons from Deep Learning

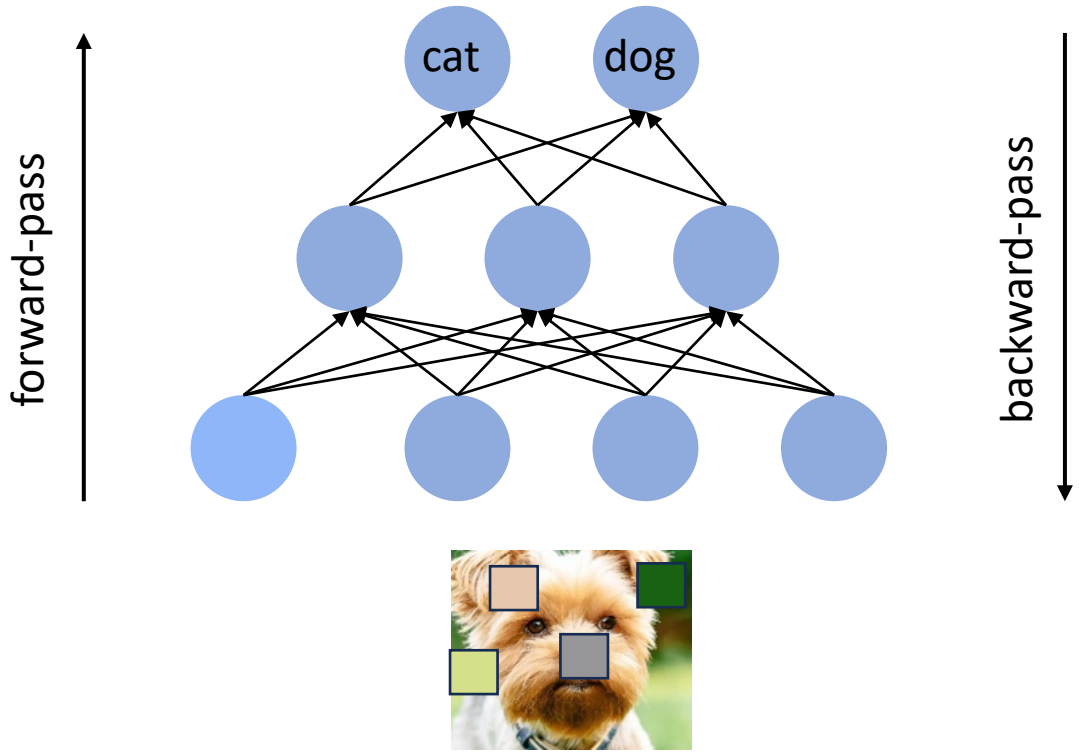
Jason K. Eshraghian
Assistant Professor, ECE, UC Santa Cruz

18 March 2024

The Energy Cost of AI

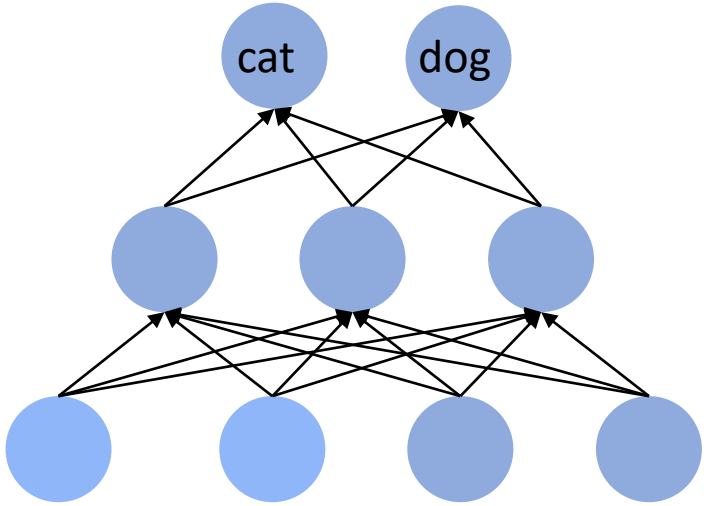
ground truth: **dog**

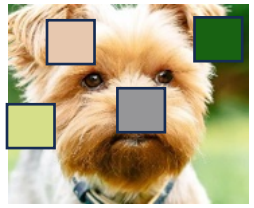
error: ++



Training

Approx. $5 million





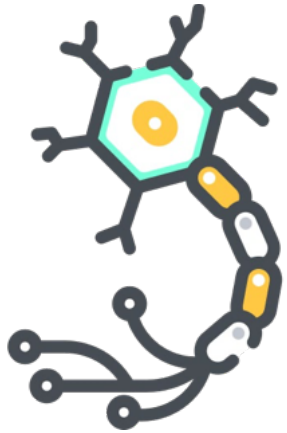
Inference

Estimate: >$100,000 p/day

How much should we plagiarize the brain?

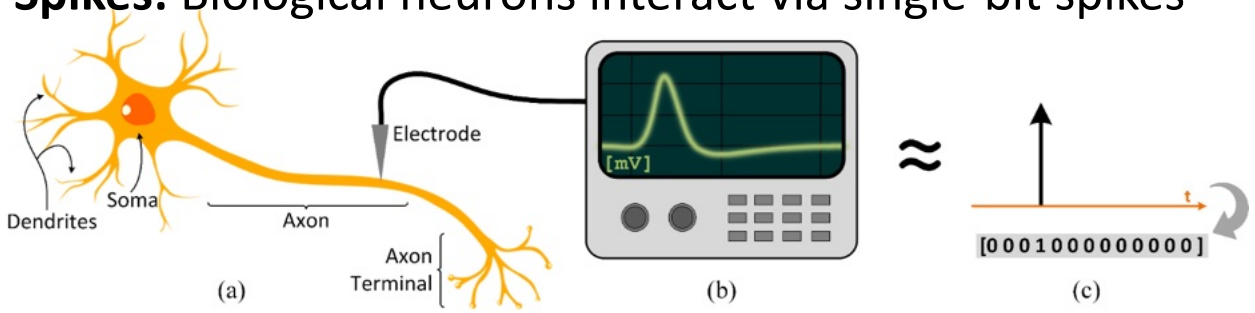
High-level: Interconnections and systems

Low-level: Neurons and Ion Channels



Toward Biological Networks: The Three S's

Spikes: Biological neurons interact via single-bit spikes

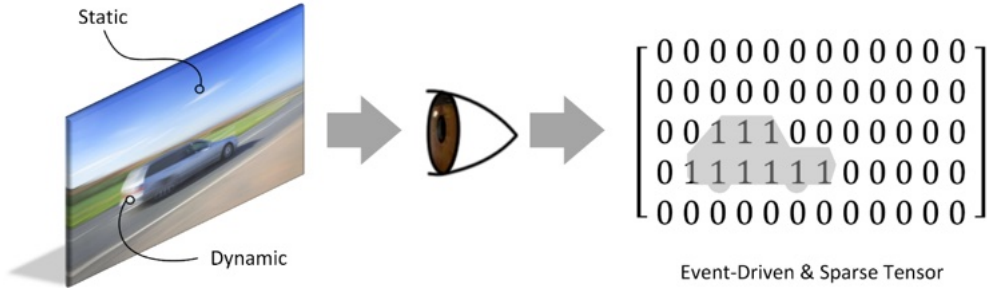


Sparsity: Biological neurons spend most of their time at rest, setting most activations to *zero* at any given time

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5]$$

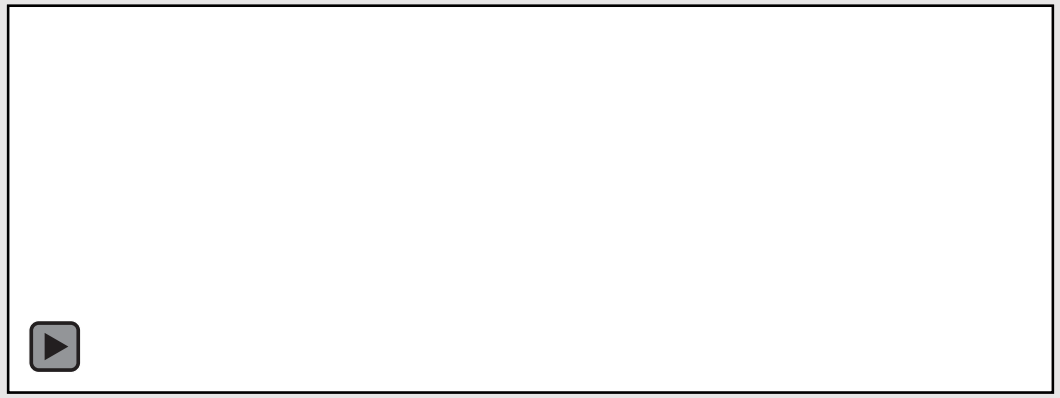
"7 at position 10; 5 at position 20"

Static Suppression (aka Event-driven Processing): The sensory periphery only processes information when there is *new* information to process

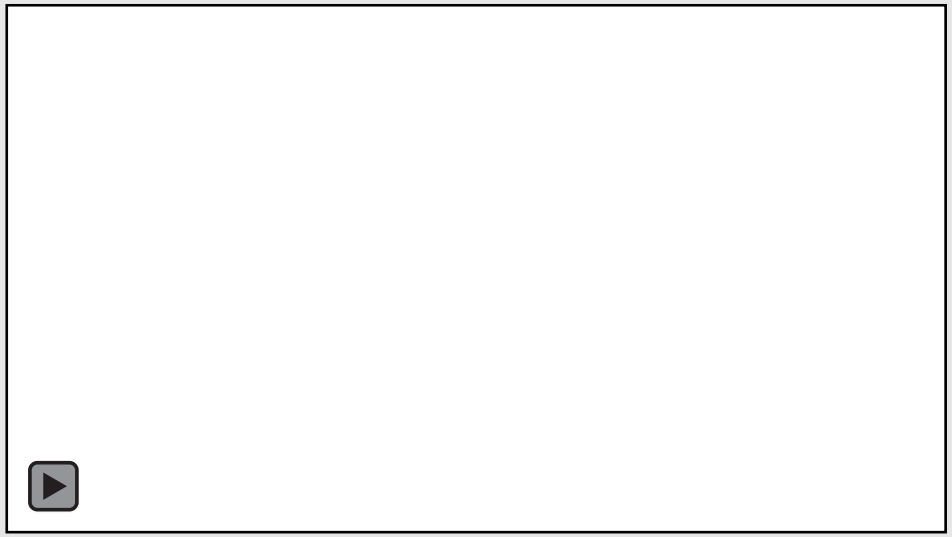


Event-based Vision Applications

Jugular Venous Pressure Estimation, *UCSC & UWA*



Microfluidic Analysis, *UCSC & USyd*



Slow-mo Lasers, *Prophesee*

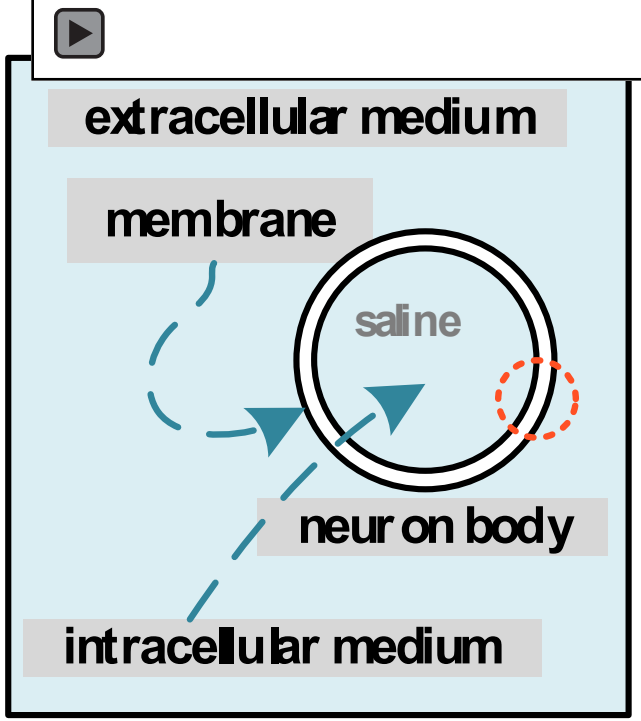
Tutorial Outline

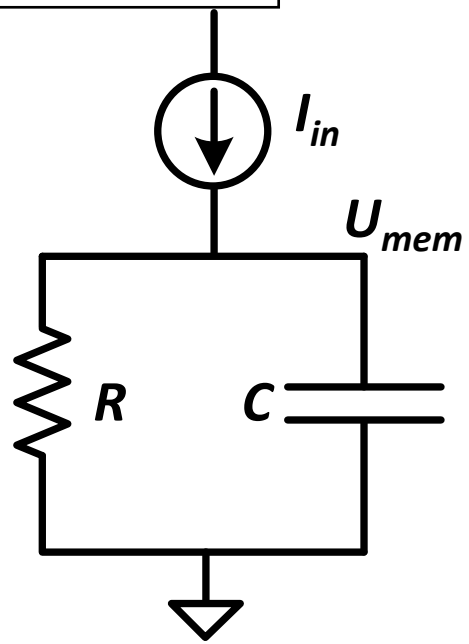
1. **Spiking Neurons**
2. **How to Train Your Spiking Neural Net**
   a. Local Learning
   b. Shadow Training
   c. Spike Time Learning
   d. Backprop Through Time and Surrogate Gradients
   e. Spike Time Learning
   f. Real-Time Recurrent Learning
3. **Fixed Precision Training**
4. **SNNs in the Wild**
5. **Perspectives: How can we do better?**

Tutorial Outline

1. **Spiking Neurons**
2. **How to Train Your Spiking Neural Net**
   a. Local Learning
   b. Shadow Training
   c. Spike Time Learning
   d. Backprop Through Time and Surrogate Gradients
   e. Spike Time Learning
   f. Real-Time Recurrent Learning
3. **Fixed Precision Training**
4. **SNNs in the Wild**

egrate-and-Fire Neuron

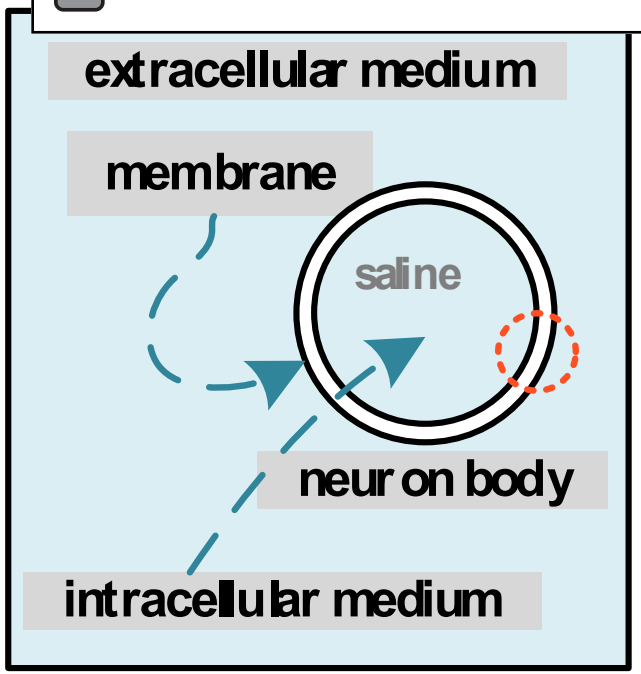


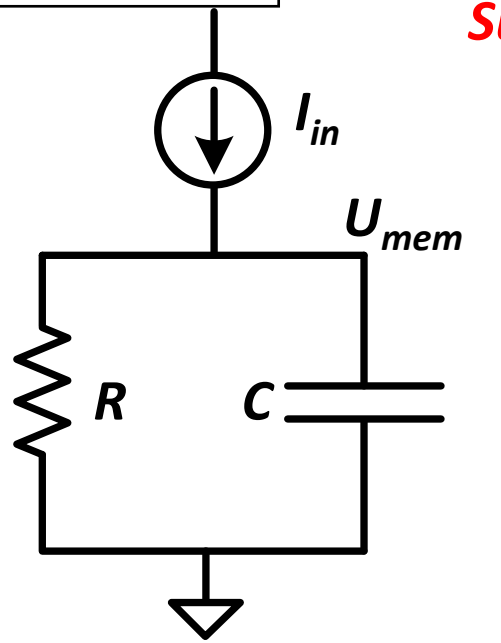


- Bilipid thin-film membrane surrounded by ions: **capacitive**
- Ion-leakage/transfer: **resistive**
- The leaky integrate-and-fire neuron is just a 1st-order low-pass filter, i.e., an **RC circuit**

The neuroscientists stole from electrical engineers so it's time to steal back from them

egrate-and-Fire Neuron





KCL: $I_{\text{in}}(t) = I_R + I_C$

Ohm's Law: $I_R(t) = \frac{U_{\text{mem}}(t)}{R}$

Capacitance: $Q = CU_{\text{mem}}(t)$

$$\frac{dQ}{dt} = I_C(t) = C\frac{dU_{\text{mem}}(t)}{dt}$$

Substitution: $$I_{\text{in}}(t) = \frac{U_{\text{mem}}(t)}{R} + C\frac{dU_{\text{mem}}(t)}{dt}$$

$$\Longrightarrow RC\frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

[V/Time] [V] [V]

Which means RC=τ [time]

$$\tau\frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

The Leaky Integrate-and-Fire Neuron

For a derivative of a function to be of the same form as the original function, i.e., $\frac{dU_{\text{mem}}(t)}{dt} \propto U_{\text{mem}}(t)$ ***this implies the solution is exponential with a time constant τ***

Say the neuron starts at U_0 with no input. The solution is:

$$U_{\text{mem}}(t) = U_0 e^{-\frac{t}{\tau}}$$

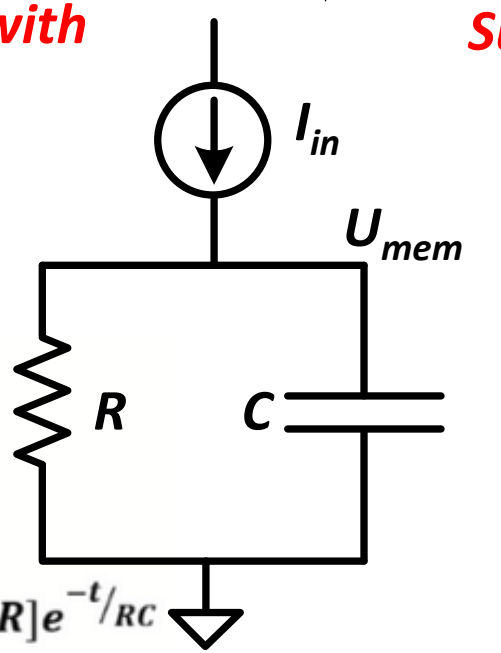
More generally:

$$\Rightarrow U_{\text{mem}}(t) = I_{in}(t)R + c_1 e^{-t/RC}$$

where at t=0, $U_{\text{mem}}(t) = U_0$

$$\Rightarrow c_1 = U_0 - I_{in}(t)R$$

$$\Rightarrow U_{\text{mem}}(t) = I_{in}(t)R + [U_0 - I_{in}(t)R]e^{-t/RC}$$



KCL: $I_{\text{in}}(t) = I_R + I_C$

Ohm's Law: $I_R(t) = \frac{U_{\text{mem}}(t)}{R}$

Capacitance: $Q = CU_{\text{mem}}(t)$

$$\frac{dQ}{dt} = I_C(t) = C\frac{dU_{\text{mem}}(t)}{dt}$$

Substitution: $I_{\text{in}}(t) = \frac{U_{\text{mem}}(t)}{R} + C\frac{dU_{\text{mem}}(t)}{dt}$

$$\Longrightarrow RC\frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

[V/Time] [V] [V]

Which means RC=τ [time]

$$\tau\frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

The Leaky Integrate-and-Fire Neuron

For a derivative of a function to be of the same form as the original function, i.e., $\frac{dU_{\text{mem}}(t)}{dt} \propto U_{\text{mem}}(t)$ ***this implies the solution is exponential with a time constant τ***

Say the neuron starts at U_0 with no input. The solution is:

$$U_{\text{mem}}(t) = U_0 e^{-\frac{t}{\tau}}$$

More generally:

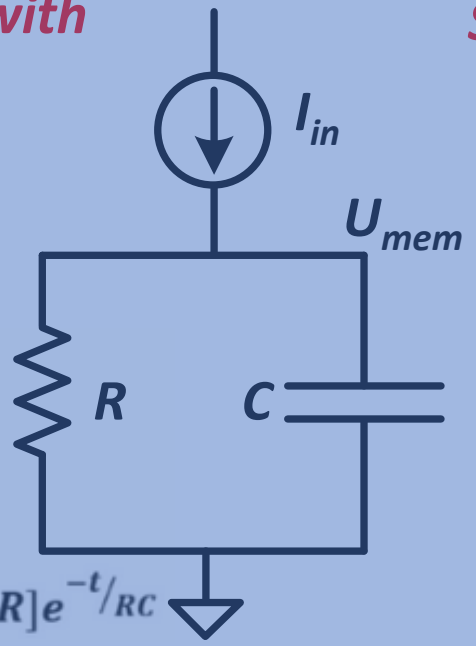
$$\Rightarrow U_{\text{mem}}(t) = I_{in}(t)R + c_1 e^{-t/RC}$$

where at t=0, $U_{\text{mem}}(t) = U_0$

$$\Rightarrow c_1 = U_0 - I_{in}(t)R$$

$$\Rightarrow U_{\text{mem}}(t) = I_{in}(t)R + [U_0 - I_{in}(t)R]e^{-t/RC}$$

Don't be scared.



KCL: $I_{\text{in}}(t) = I_R + I_C$

Ohm's Law: $I_R(t) = \frac{U_{\text{mem}}(t)}{R}$

Capacitance: $Q = CU_{\text{mem}}(t)$

$$\frac{dQ}{dt} = I_C(t) = C\frac{dU_{\text{mem}}(t)}{dt}$$

Substitution:

$$I_{\text{in}}(t) = \frac{U_{\text{mem}}(t)}{R} + C\frac{dU_{\text{mem}}(t)}{dt}$$

$$\Longrightarrow RC\frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

[V/Time] [V] [V]

Which means RC=τ [time]

$$\tau\frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

The Leaky Integrate-and-Fire Neuron

For a derivative of a function to be of the same form as the original function, i.e., $\frac{dU_{\text{mem}}(t)}{dt} \propto U_{\text{mem}}(t)$ ***this implies the solution is exponential with a time constant τ***

Say the neuron starts at U_0 ***with no input. The solution is:***

$$U_{\text{mem}}(t) = U_0 e^{-\frac{t}{\tau}}$$

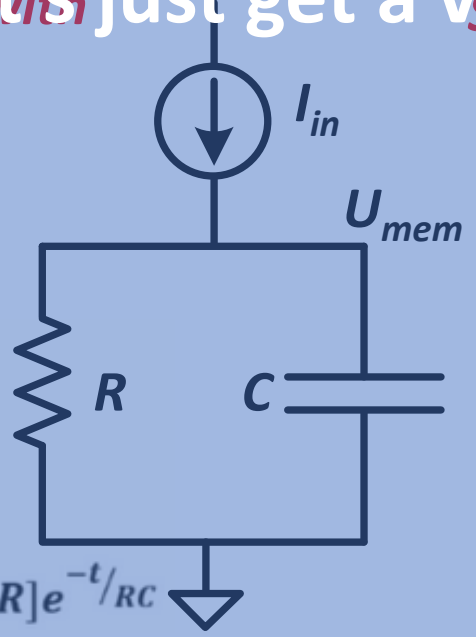
More generally:

$$\Rightarrow U_{\text{mem}}(t) = I_{in}(t)R + c_1 e^{-t/RC}$$

where at t=0, $U_{\text{mem}}(t) = U_0$

$$\Rightarrow c_1 = U_0 - I_{in}(t)R$$

$$\Rightarrow U_{\text{mem}}(t) = I_{in}(t)R + [U_0 - I_{in}(t)R]e^{-t/RC}$$



Don't be scared.

Let's just get a vibe of the math.

KCL: $I_{\text{in}}(t) = I_R + I_C$

Ohm's Law: $I_R(t) = \frac{U_{\text{mem}}(t)}{R}$

Capacitance: $Q = CU_{\text{mem}}(t)$

$$\frac{dQ}{dt} = I_C(t) = C\frac{dU_{\text{mem}}(t)}{dt}$$

Substitution:

$$I_{\text{in}}(t) = \frac{U_{\text{mem}}(t)}{R} + C\frac{dU_{\text{mem}}(t)}{dt}$$

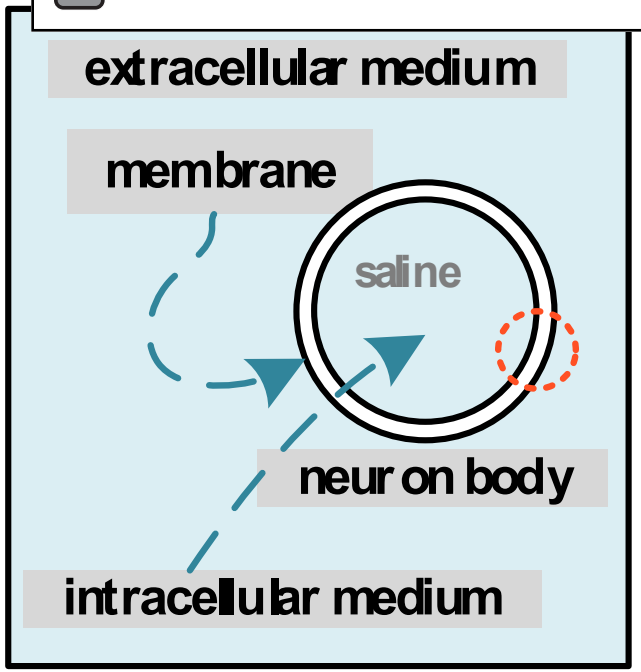
$$\Longrightarrow RC\frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

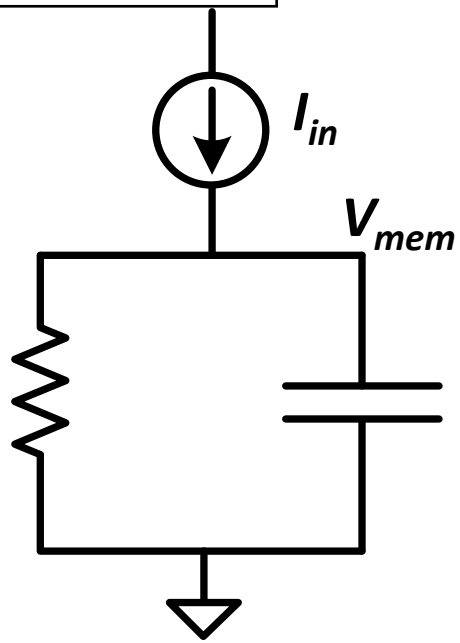
[V/Time] [V] [V]

Which means RC=τ [time]

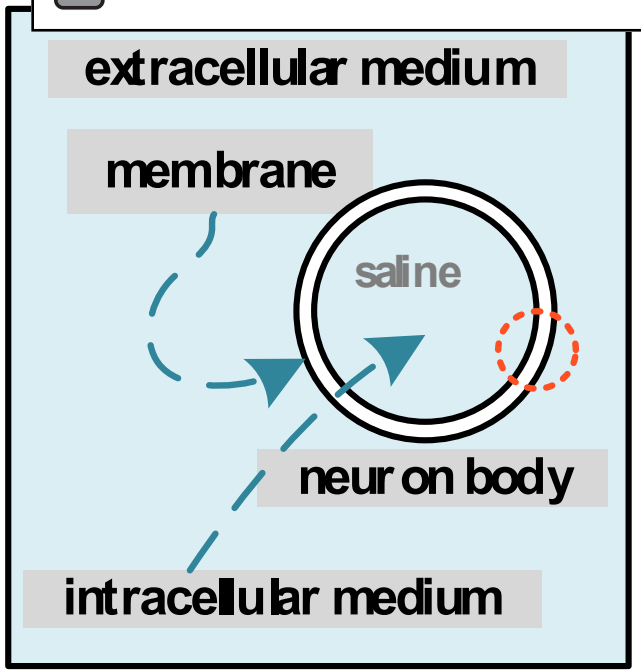
$$\tau\frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

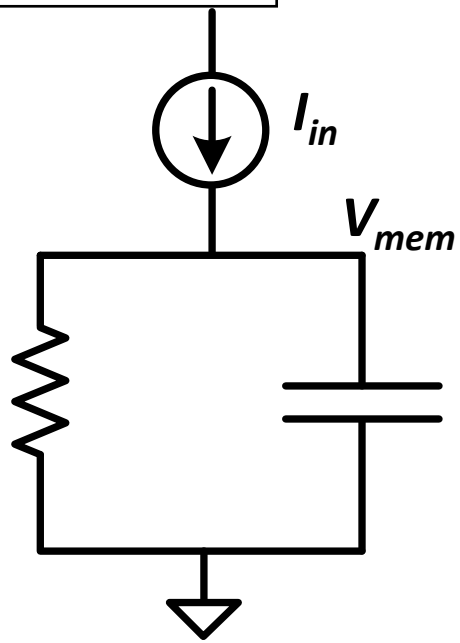
egrate-and-Fire Neuron



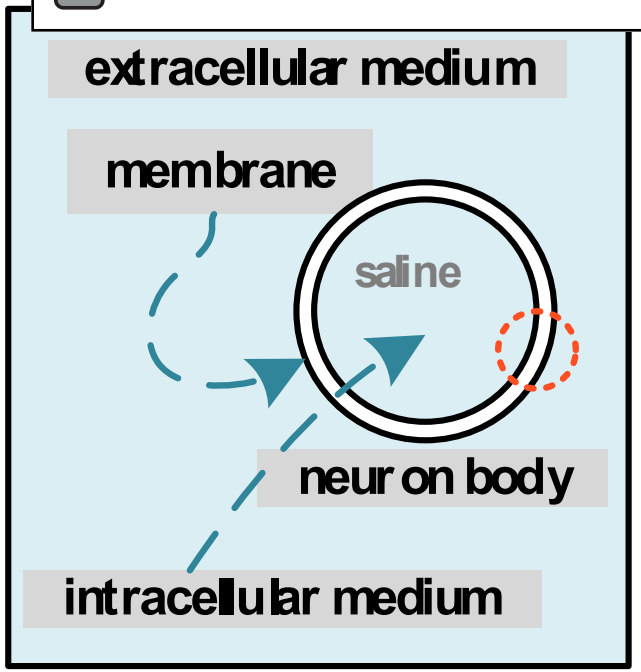


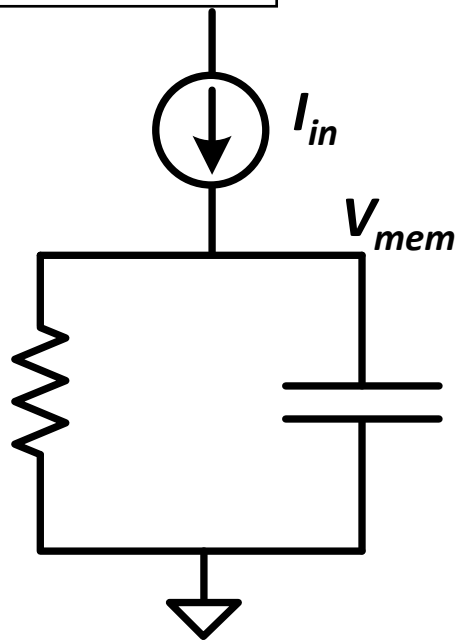
egrate-and-Fire Neuron



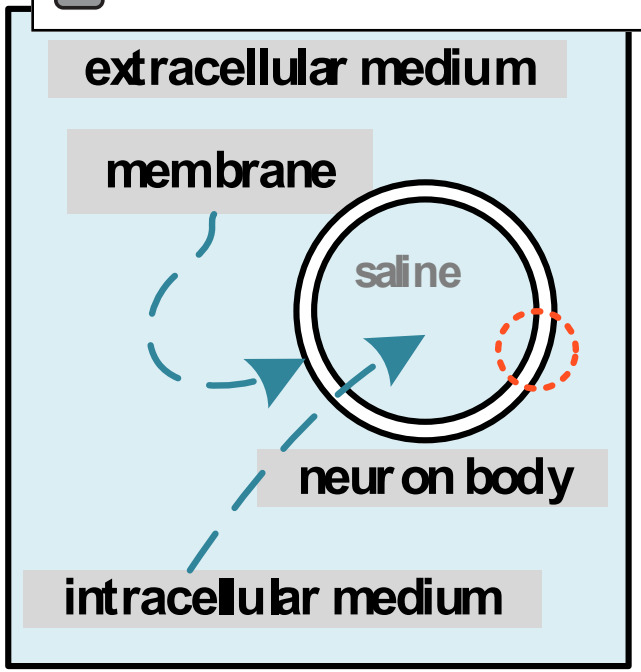


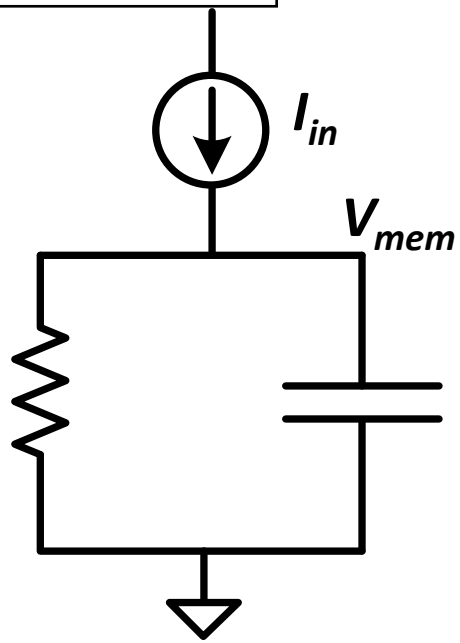
egrate-and-Fire Neuron



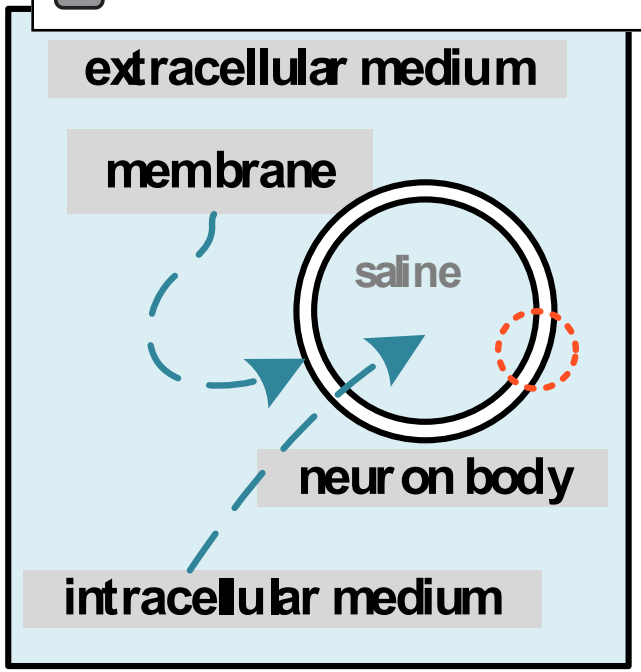


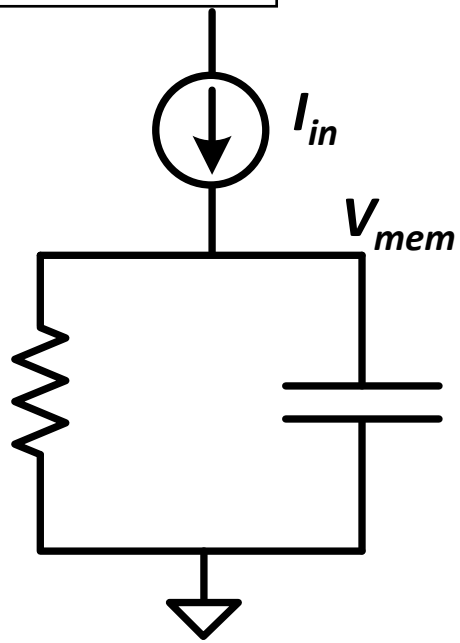
egrate-and-Fire Neuron



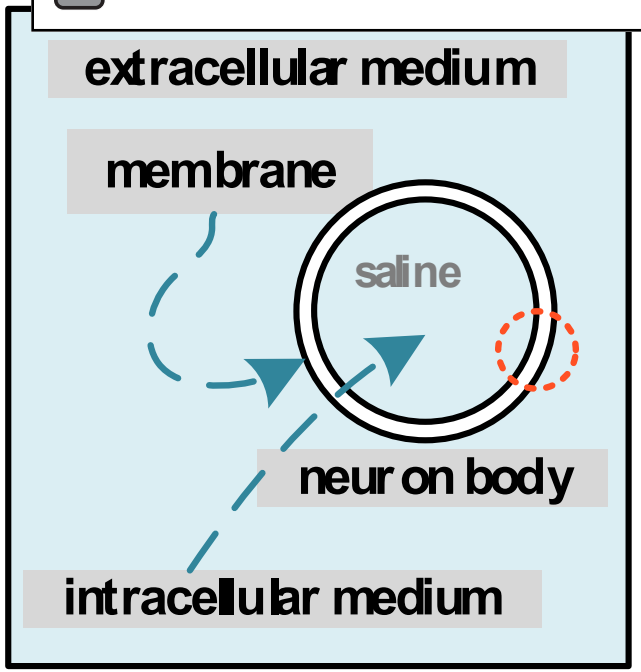


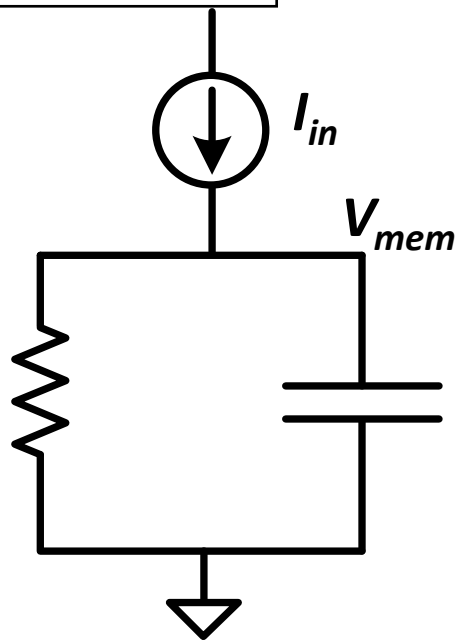
egrate-and-Fire Neuron



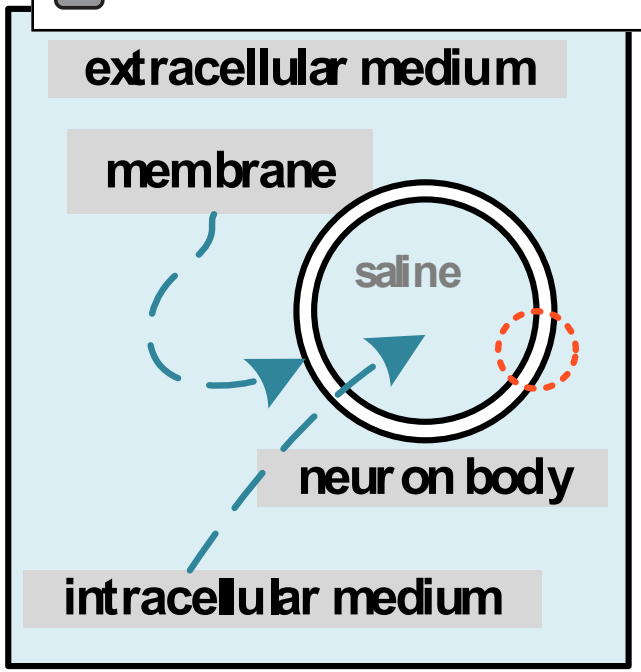


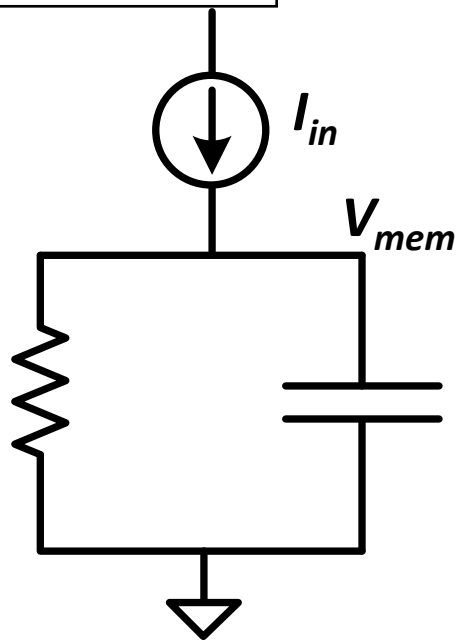
egrate-and-Fire Neuron



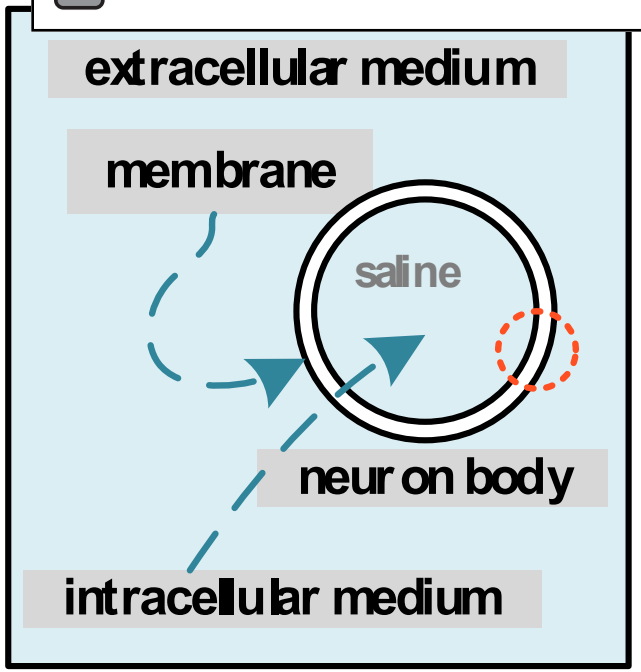


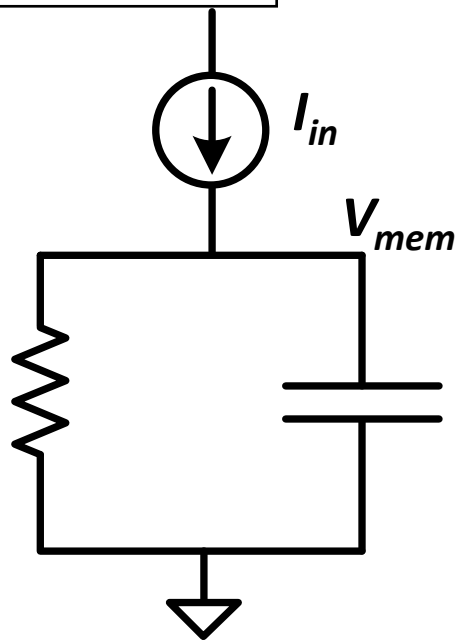
egrate-and-Fire Neuron





egrate-and-Fire Neuron





$$V\,[t+1] = \beta V[t] + (1-\beta) I_{in}[t]$$

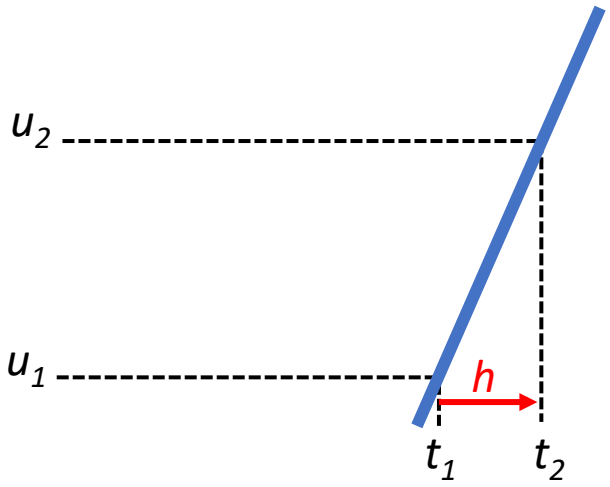
Note: $\beta = \left(1 - {}^{\Delta t}/_{\tau}\right)$, τ=RC, Δt=1, R=1

The Leaky Integrate-and-Fire Neuron

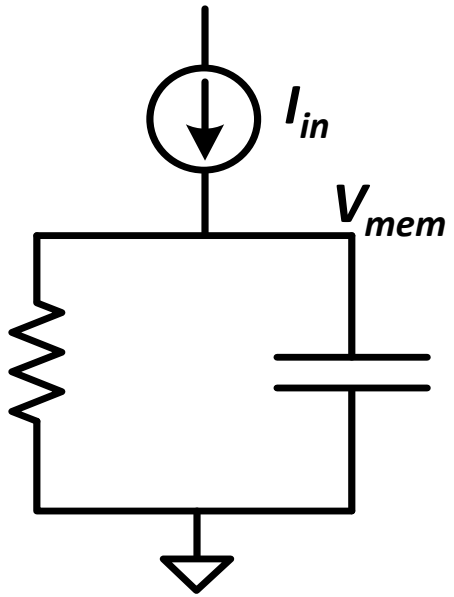
Making it deep learning compatible: discretize it in time

Limit definition of the derivative

$$\frac{dU}{dt} = \lim_{h \to 0} \frac{U(t+h) - U(t)}{h}$$



$$\tau \frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

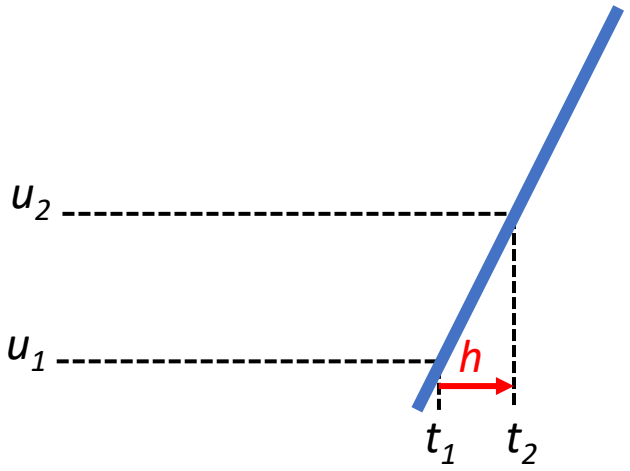


The Leaky Integrate-and-Fire Neuron

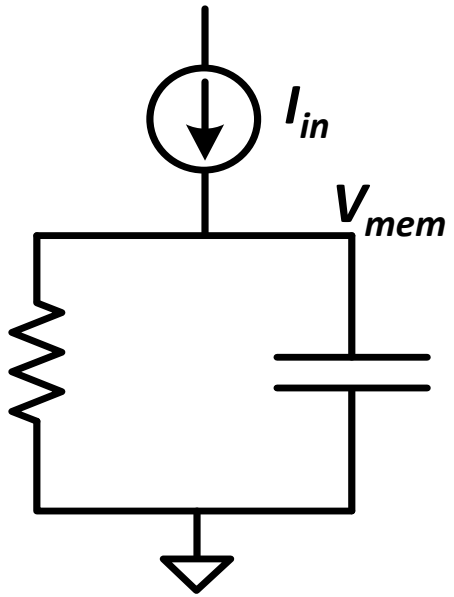
Making it deep learning compatible: discretize it in time

Limit definition of the derivative

$$\frac{dU}{dt} = \lim_{h \to 0} \frac{U(t+h) - U(t)}{h}$$



$$\tau \frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

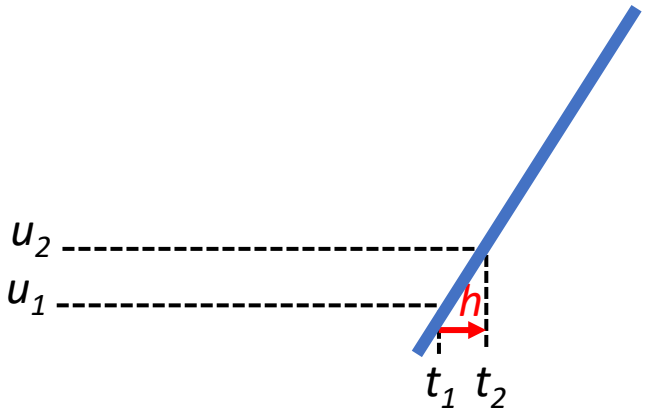


The Leaky Integrate-and-Fire Neuron

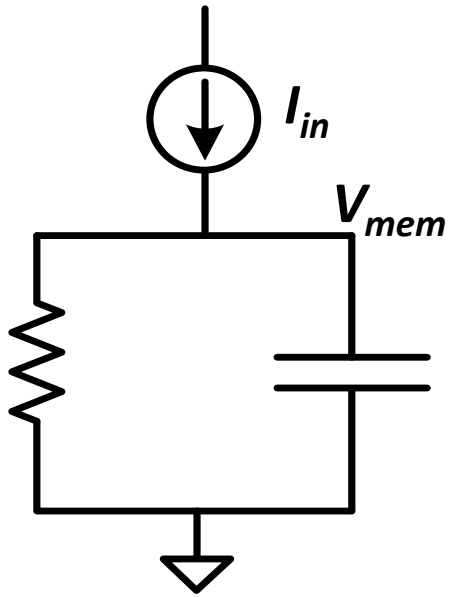
Making it deep learning compatible: discretize it in time

Limit definition of the derivative

$$\frac{dU}{dt} = \lim_{h \to 0} \frac{U(t+h) - U(t)}{h}$$



$$\tau \frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

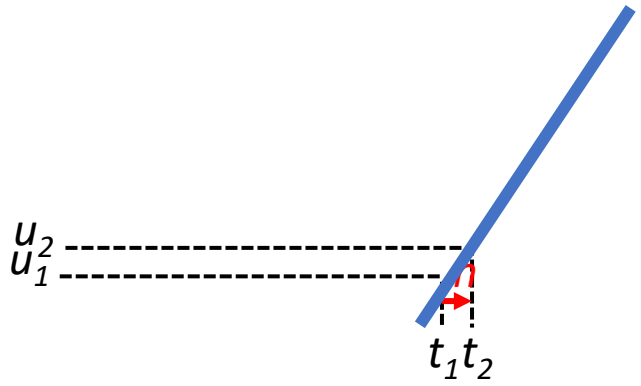


The Leaky Integrate-and-Fire Neuron

Making it deep learning compatible: discretize it in time

Limit definition of the derivative

$$\frac{dU}{dt} = \lim_{h \to 0} \frac{U(t+h) - U(t)}{h}$$



Forward-Euler Method:

A numerical method to solve ODEs based on the current state of the system + discretizing time

$$\tau \frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

First, let's solve this derivative without taking the limit $\Delta t \to 0$:

$$\tau \frac{U(t+\Delta t) - U(t)}{\Delta t} = -U(t) + RI_{\text{in}}(t)$$

For a small enough Δt, this is a fair approximation

Isolate the membrane potential at the next step

$$U(t+\Delta t) = U(t) + \frac{\Delta t}{\tau}\big(-U(t) + RI_{\text{in}}(t)\big)$$

```
def leaky_integrate_neuron(U, time_step=1e-3, I=0, R=5e7, C=1e-10):
  tau = R*C
  U = U + (time_step/tau)*(-U + I*R)
  return U
```

The Leaky Integrate-and-Fire Neuron

Making it deep learning compatible: discretize it in time

This is a lot. Let's keep simplifying.

$$U(t+\Delta t) = (1 - \frac{\Delta t}{\tau})U(t) + \frac{\Delta t}{\tau} I_{\text{in}}(t) R$$

Assume no input current: I_{in} = 0A

$$U(t+\Delta t) = (1 - \frac{\Delta t}{\tau})U(t)$$

Let the ratio of subsequent U be the decay rate of membrane potential

$$U(t+\Delta t) = \beta U(t)$$

From this eqn, this implies:

$$\beta = (1 - \frac{\Delta t}{\tau})$$

$$\tau \frac{dU_{\text{mem}}(t)}{dt} = -U_{\text{mem}}(t) + RI_{\text{in}}(t)$$

First, let's solve this derivative without taking the limit $\Delta t \to 0$:

$$\tau \frac{U(t+\Delta t) - U(t)}{\Delta t} = -U(t) + RI_{\text{in}}(t)$$

For a small enough Δt, this is a fair approximation

Isolate the membrane potential at the next step

$$U(t+\Delta t) = U(t) + \frac{\Delta t}{\tau}\big(-U(t) + RI_{\text{in}}(t)\big)$$

```
def leaky_integrate_neuron(U, time_step=1e-3, I=0, R=5e7, C=1e-10):
  tau = R*C
  U = U + (time_step/tau)*(-U + I*R)
  return U
```

The Leaky Integrate-and-Fire Neuron

Making it deep learning compatible: discretize it in time

This is a lot. Let's keep simplifying.

$$U(t+\Delta t) = (1 - \frac{\Delta t}{\tau})U(t) + \frac{\Delta t}{\tau} I_{\text{in}}(t) R$$

Assume no input current: I_{in} = 0A

$$U(t+\Delta t) = (1 - \frac{\Delta t}{\tau})U(t)$$

Let the ratio of subsequent U be the decay rate of membrane potential

$$U(t+\Delta t) = \beta U(t)$$

From this eqn, this implies:

$$\beta = (1 - \frac{\Delta t}{\tau})$$

2 bold assumptions to condense math:

- ***Let t represent time-steps in a sequence rather than continuous time***
- ***Assume R=1 (normalization? Or just being lazy?)***

$$\beta = (1 - \frac{1}{C}) \implies (1-\beta) I_{\text{in}} = \frac{1}{\tau} I_{\text{in}}$$

Input current is weighted by (1-β)

$$U[t+1] = \beta U[t] + (1-\beta) I_{\text{in}}[t+1]$$

Deep learning needs something learnable though

$$U[t+1] = \beta U[t] + W X[t+1]$$

$$S[t] = \begin{cases} 1, & \text{if } U[t] > U_{\text{thr}} \\ 0, & \text{otherwise} \end{cases}$$

The Leaky Integrate-and-Fire Neuron

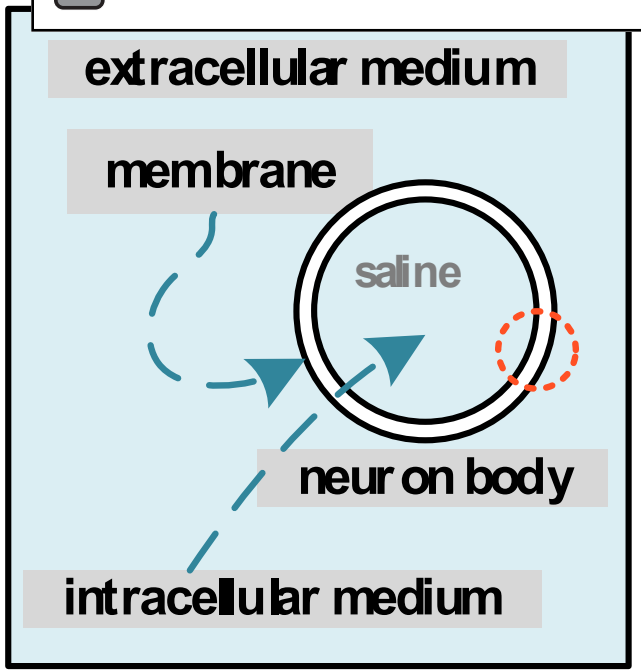
Making it deep learning compatible: discretize it in time

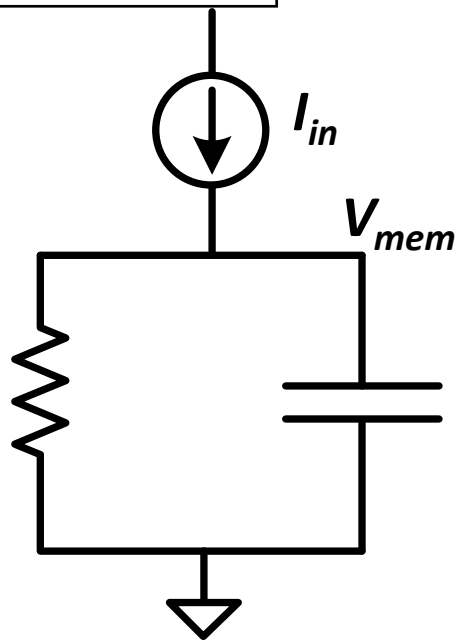
$$U[t+1] = \underbrace{\beta U[t]}_{\text{decay}} + \underbrace{WX[t+1]}_{\text{input}} - \underbrace{S[t]U_{\text{thr}}}_{\text{reset}}$$

$$S[t] = \begin{cases} 1, & \text{if } U[t] > U_{\text{thr}} \\ 0, & \text{otherwise} \end{cases}$$

```
def leaky_integrate_and_fire(mem, x, w, beta, threshold=1):
  spk = (mem > threshold) # if membrane exceeds threshold, spk=1, else, 0
  mem = beta * mem + w*x - spk*threshold
  return spk, mem
```

egrate-and-Fire Neuron

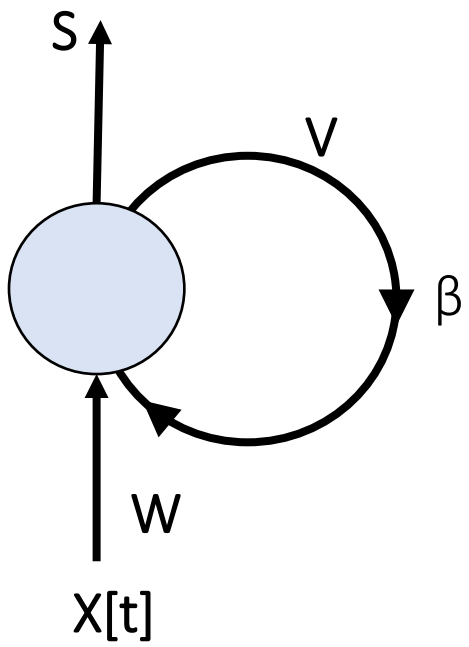




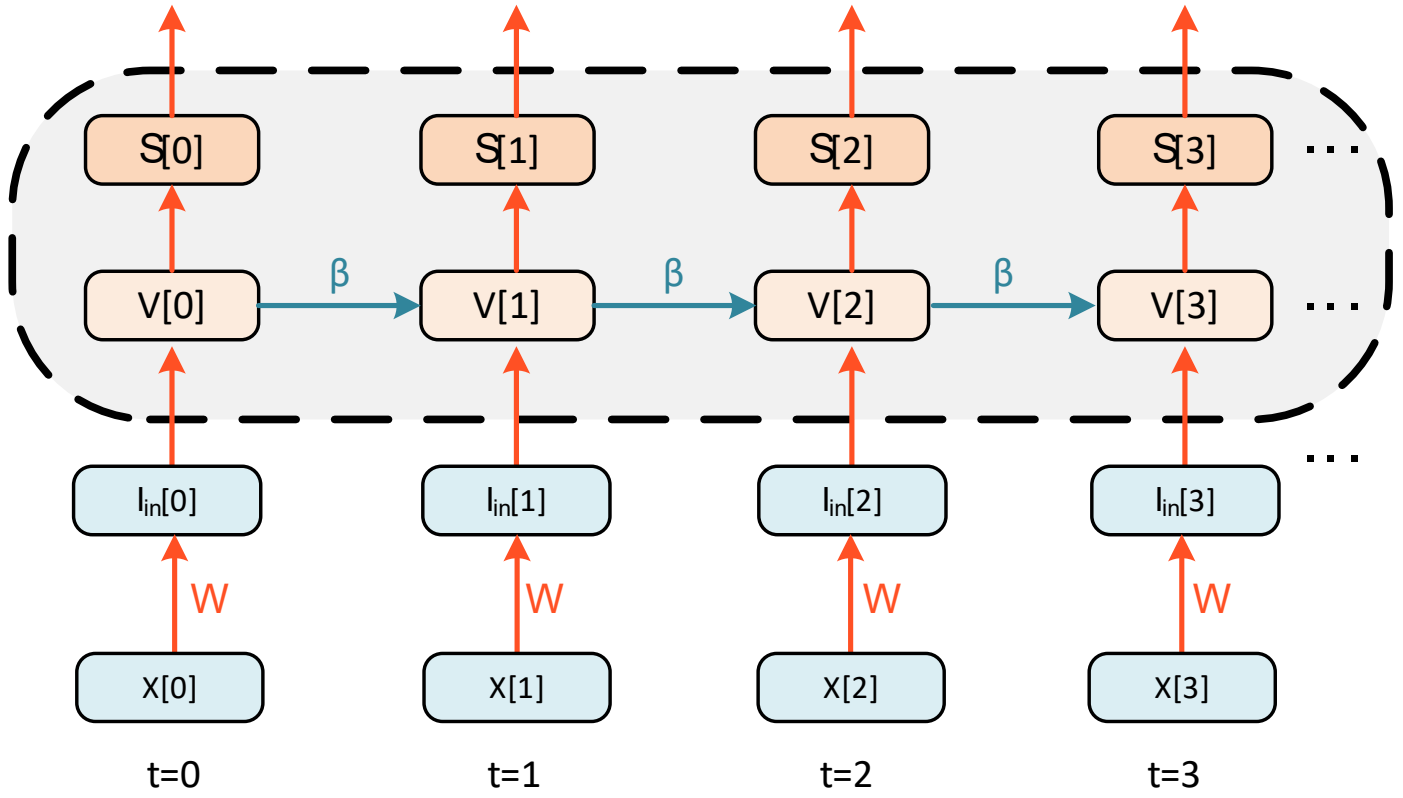
$$V\,[t+1] = \beta V[t] + (1-\beta) I_{in}[t]$$

Note: $\beta = \left(1 - {}^{\Delta t}/_{\tau}\right)$, τ=RC, Δt=1, R=1

A Recurrent Representation



Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

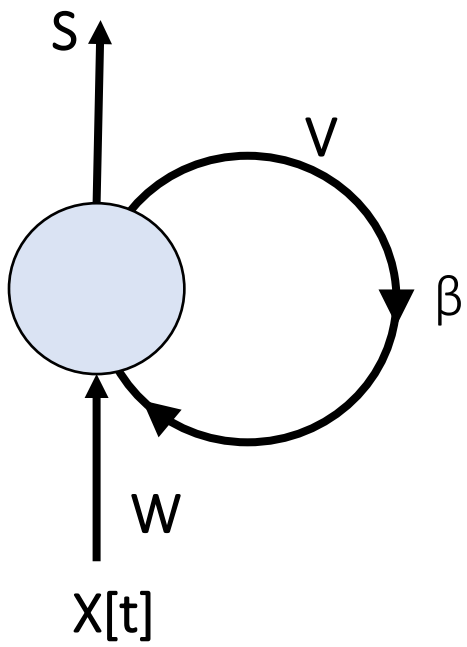
Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + (1-\beta) I_{in}[t]$$

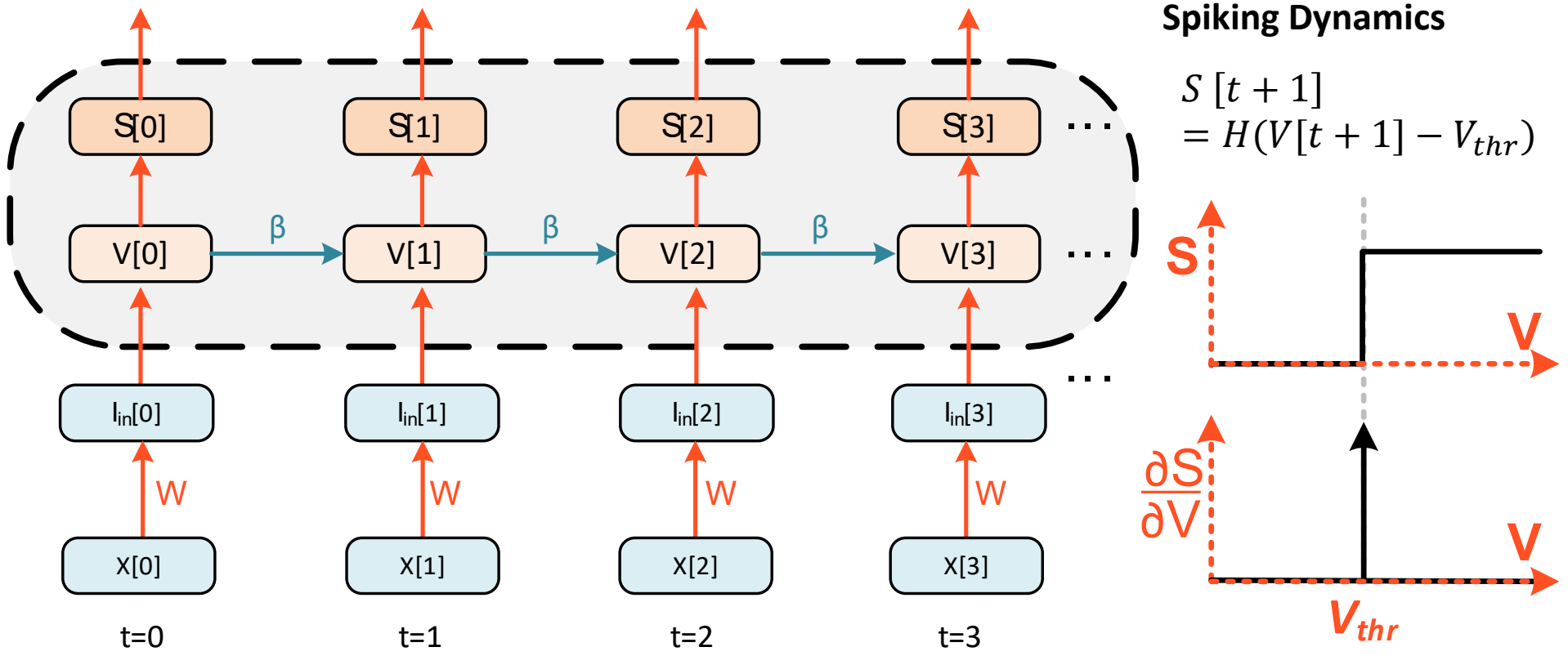
Note:
$(1-\beta)$ is removed, and
$I_{in}[t] = WX[t]$

The leaky integrate-and-fire neuron is now compatible with all the tricks and hacks that go with training deep learning models.

A Recurrent Representation



Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

The leaky integrate-and-fire neuron is now compatible with all the tricks and hacks that go with training deep learning models.

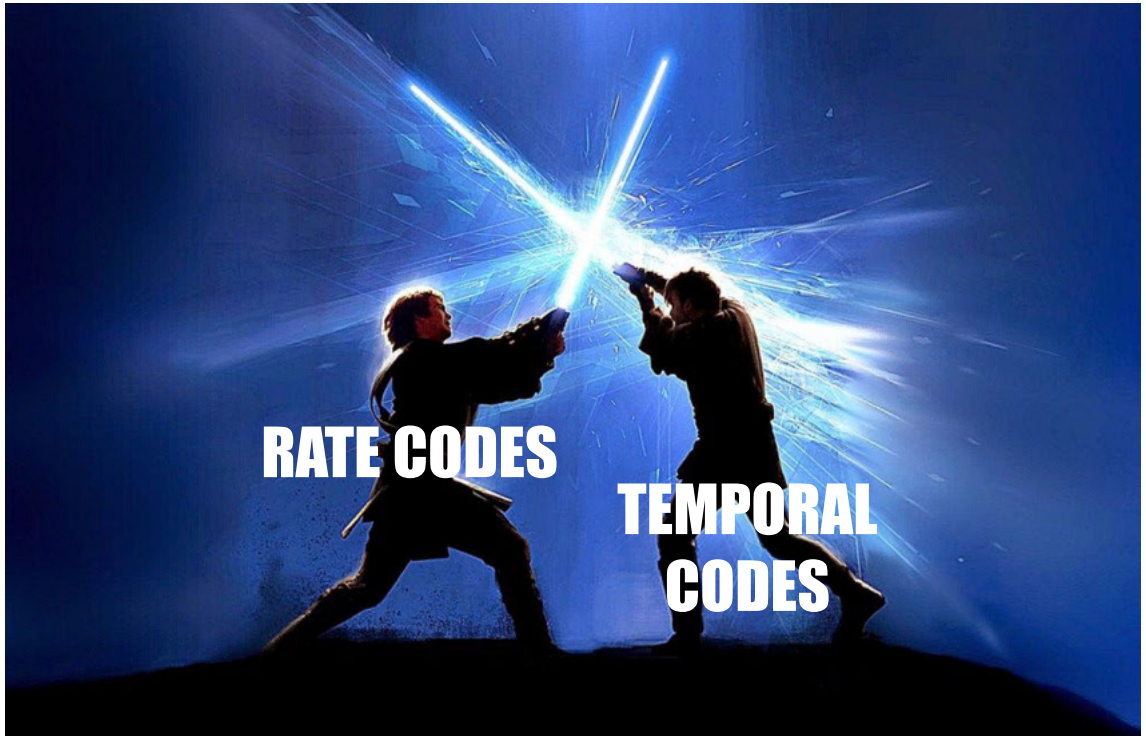
The Neural Code

The war between rate codes and temporal codes

- Once spikes have been passed to the model, an SNN will modify this representation as the spikes propagate to the output
- **How do we decode the output?**

Deep Learning:

- Rates are better for error and noise tolerance, and enable faster convergence when using backprop – *most common approach in modern SNN training; not very brain-like, but good accuracy.*
- Timing is better for sparsity (therefore power efficiency) and latency – *arguably more brainlike; see "What is the other 85% of V1 doing?"*



Olshausen & Field, Problems in Syst. Neurosci., 2004.

Tutorial Outline

1. Spiking Neurons
2. **How to Train Your Spiking Neural Net**
   a. Local Learning
   b. Shadow Training
   c. Spike Time Learning
   d. Backprop Through Time and Surrogate Gradients
   e. Spike Time Learning
   f. Real-Time Recurrent Learning
3. Fixed Precision Training
4. SNNs in the Wild
5. Perspectives: How can we do better?

Training Spiking Neural Networks

Local Learning: STDP

Spike-Timing-Dependent Plasticity (STDP) is a learning rule that adjusts connection strength based on the timing of their spikes.

Connections are strengthened if:

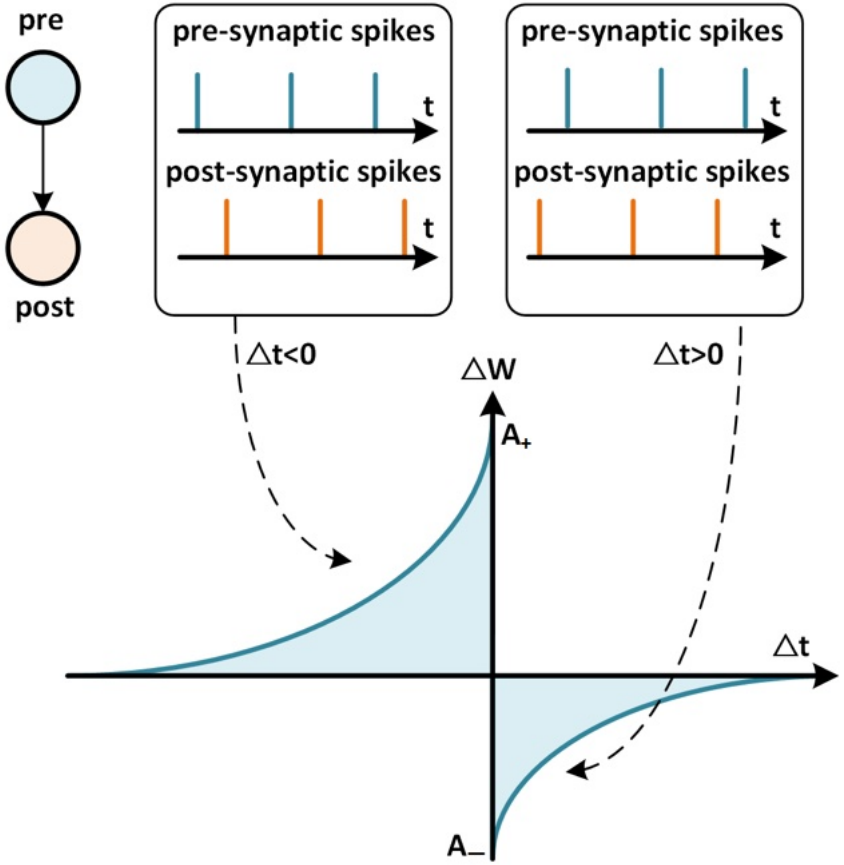
1. The neurons fire together, and
2. Follow a certain sequence

Pros

- Biologically Motivated
- Computationally Cheap

Cons

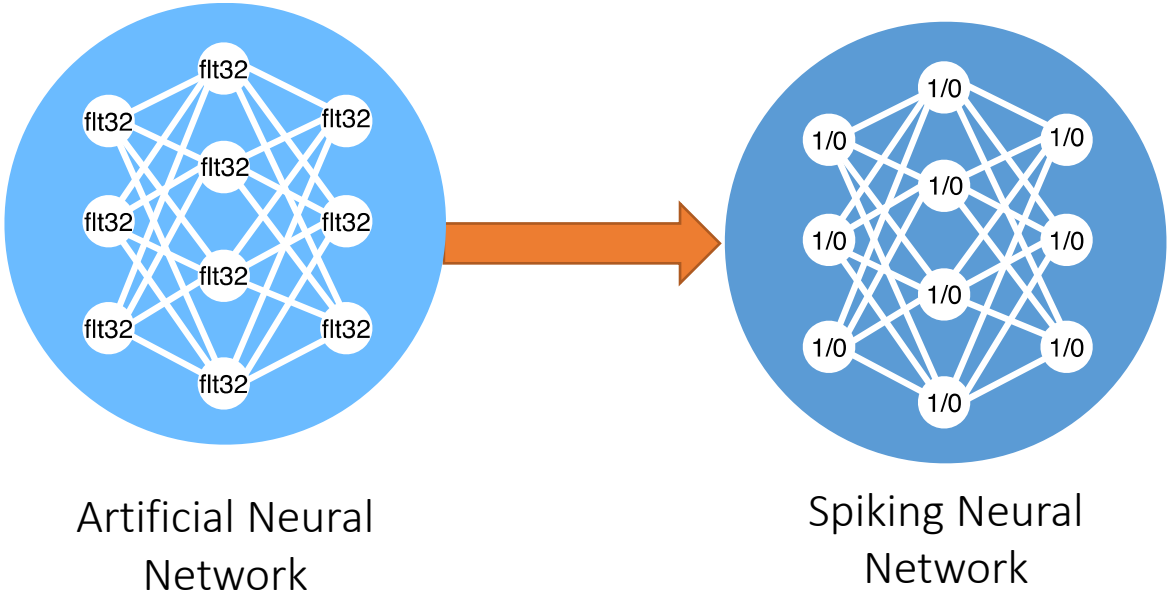
- Not very functional in the deep learning context



Bi & Poo, J. Neuroscience, 1998.

Training Spiking Neural Networks

Shadow Training: ANN-to-SNN Conversion



Artificial Neural Network

Spiking Neural Network

Pros

- The first approach to reach state-of-the-art performance in most complex tasks using SNNs (pre-2018)
- End up with a pair of networks: one for performance, one for power (with some caveats)

Cons

- SNN is an approximation of the ANN and sets an upper-bound on performance
- Sequential ANN conversion is underexplored – these models are only tested on image datasets
- Biologically implausible

B. Rueckauer et al., Frontiers in Neuroscience 11(682), Dec. 2017.
https://github.com/NeuromorphicProcessorProject/snn_toolbox

Training Spiking Neural Networks

Spike Time Learning

Take the derivative of the membrane potential with respect to time **only at the firing time**.

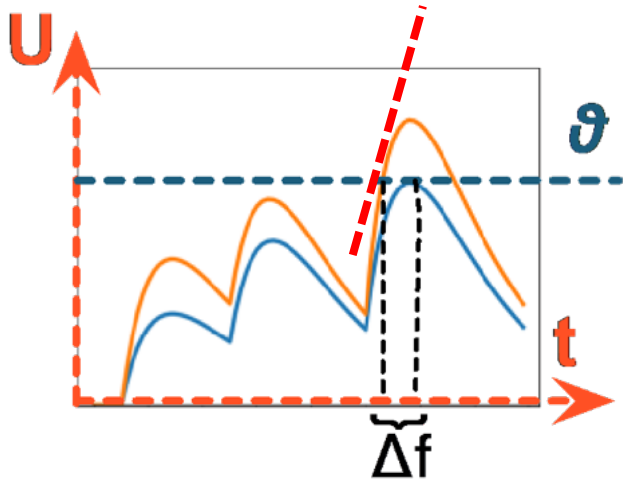
Then train the **firing time** using regression.

Pros

- First ever use of backprop on SNNs!
- Sparse spiking activity

Cons

- Potentially too sparse – one spike is learnt at a time
- If no spiking, then no learning
- Must use differentiable voltage dynamics (this rules out the simplest leaky integrate-and-fire neuron on previous slides)
- Has not yet performed well on challenging datasets
- **Very** sensitive to hyperparameters

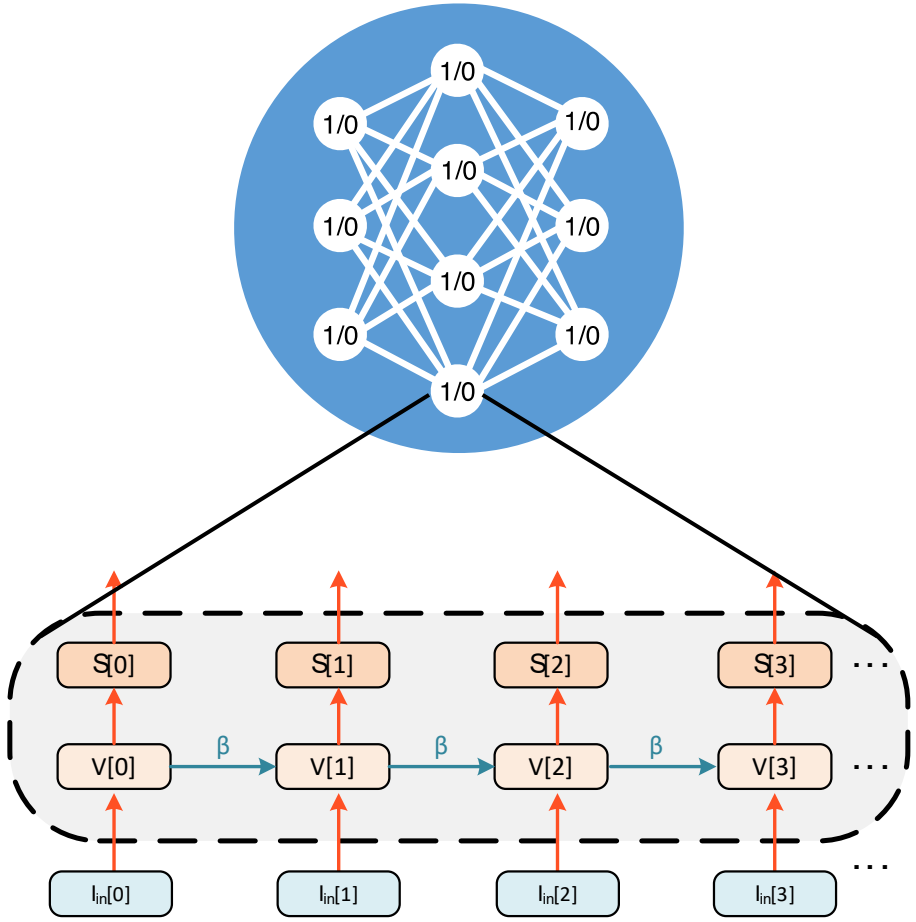


Over the past 20 years, this approach has been extended to handle alternative cases, such as multiple spikes, alternative neuron models, etc. (E.g., *Event-prop*)

S. Bohte et al., Neurocomputing 48(1), Oct. 2002.

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons from Deep Learning" arXiv, 2021.

Training Spiking Neural Networks



Backprop Through Time

Pros

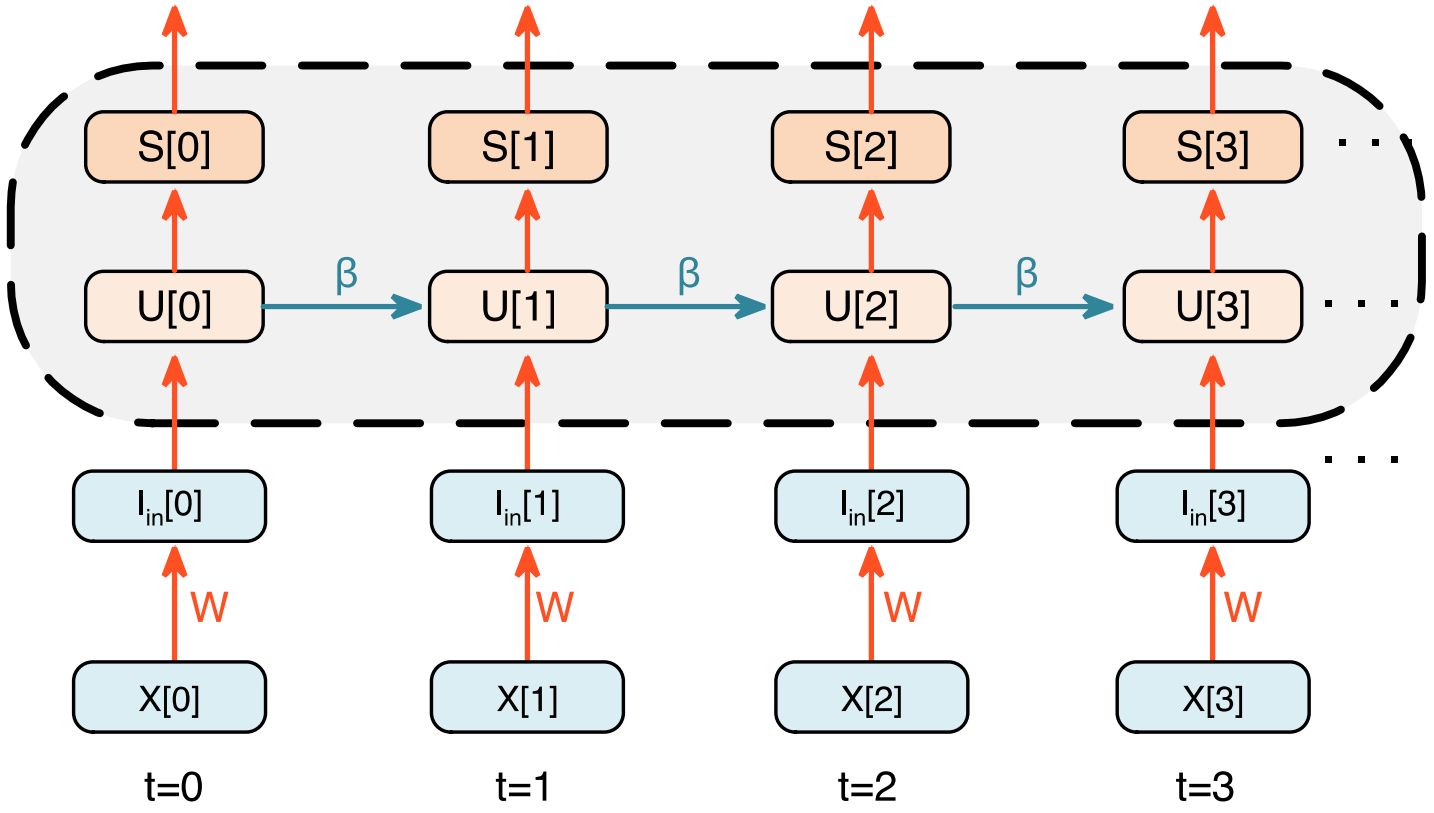
- Not bound by the upper-limit of ANN performance – reaches state-of-the-art on most tasks!
- Compatible with autodifferentiation tools
- Many deep learning tricks can be ported across

Cons

- Computationally expensive on non-neuromorphic hardware
- Somewhat sensitive to hyperparameters
- Biologically implausible: temporal and spatial credit assignment problems

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

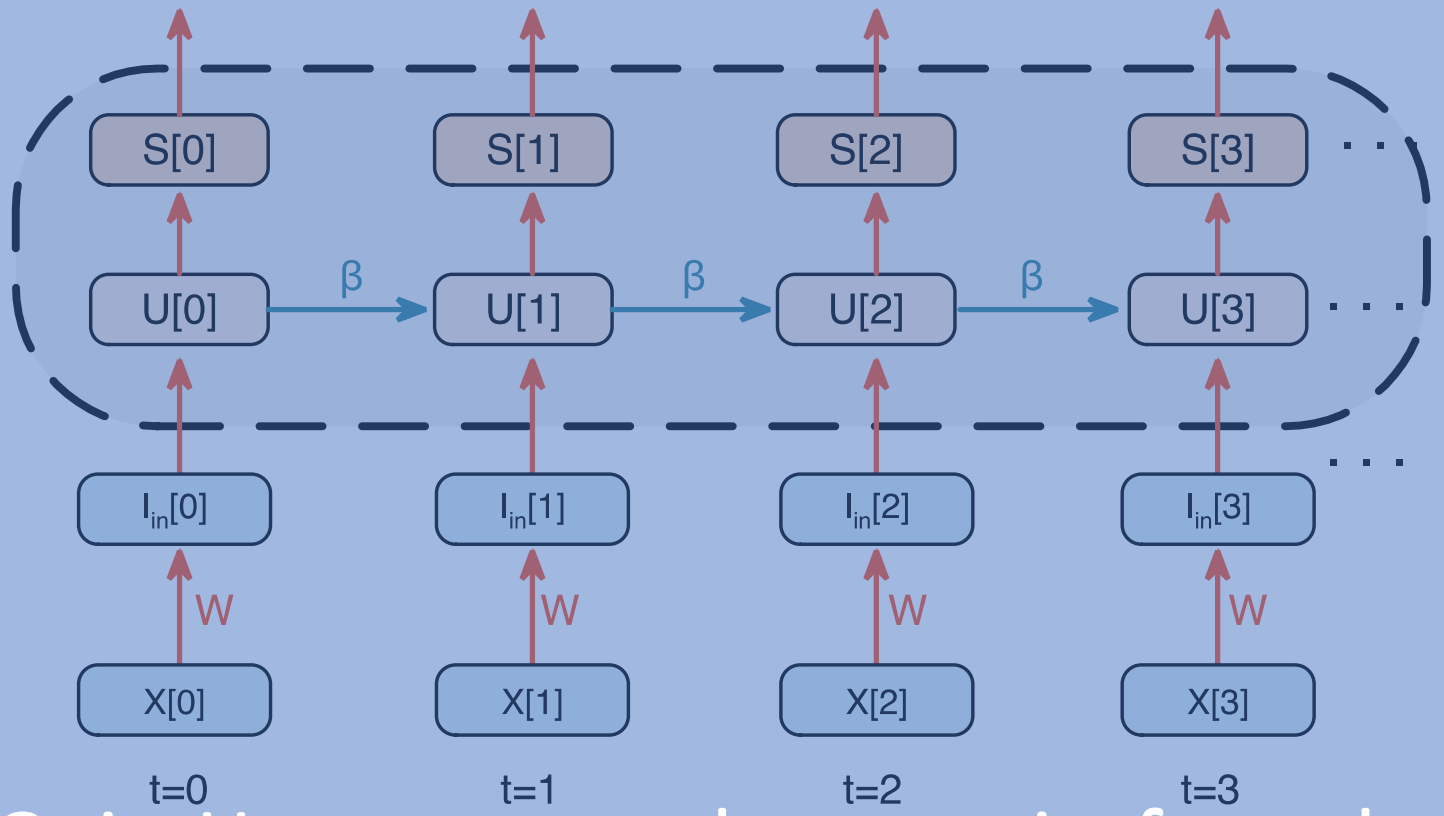
Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

Membrane Potential Dynamics

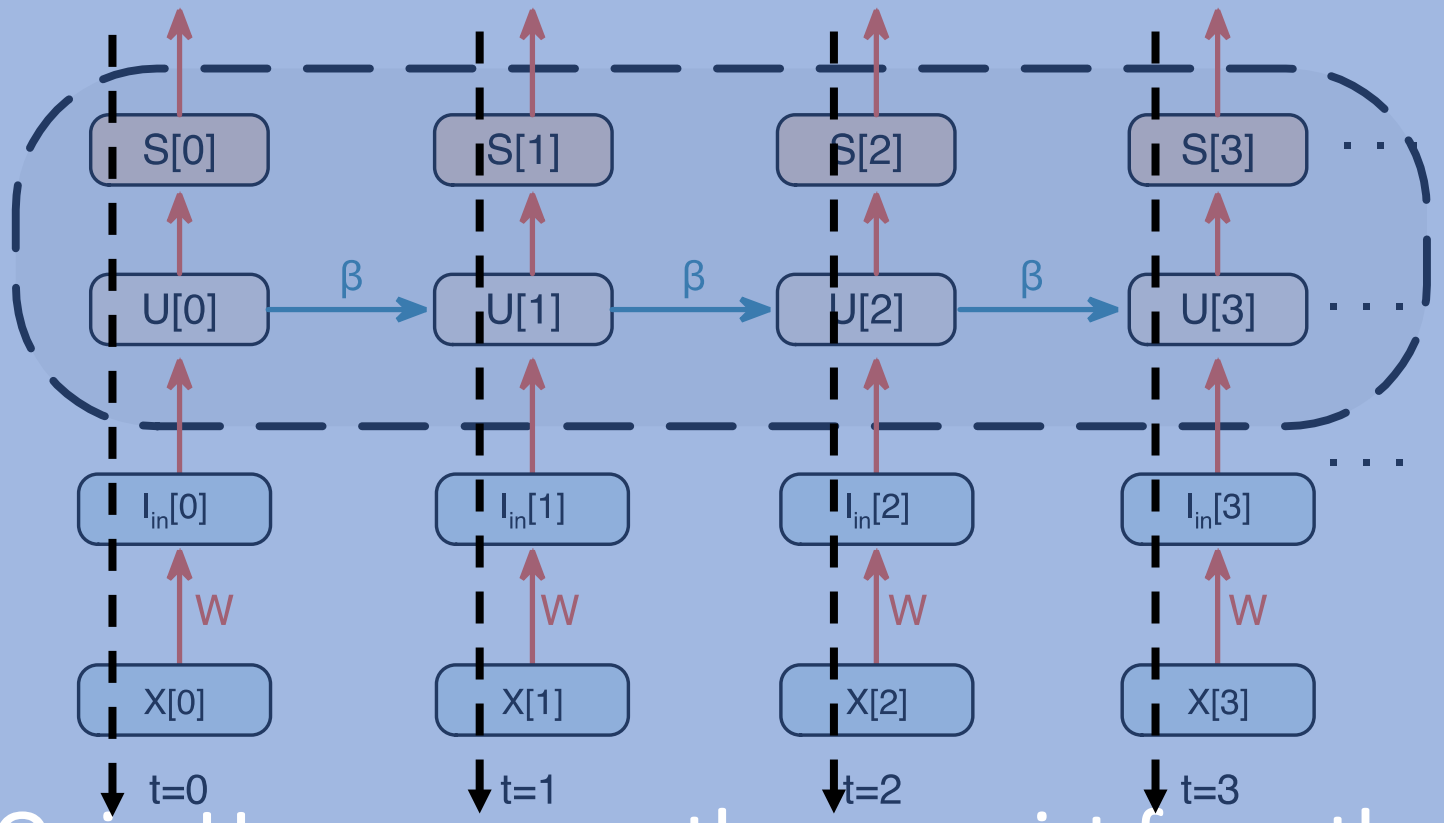
$$V[t+1] = \beta V[t] + X[t]W$$

Pop Quiz: How many pathways exist from the outputs to W?

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

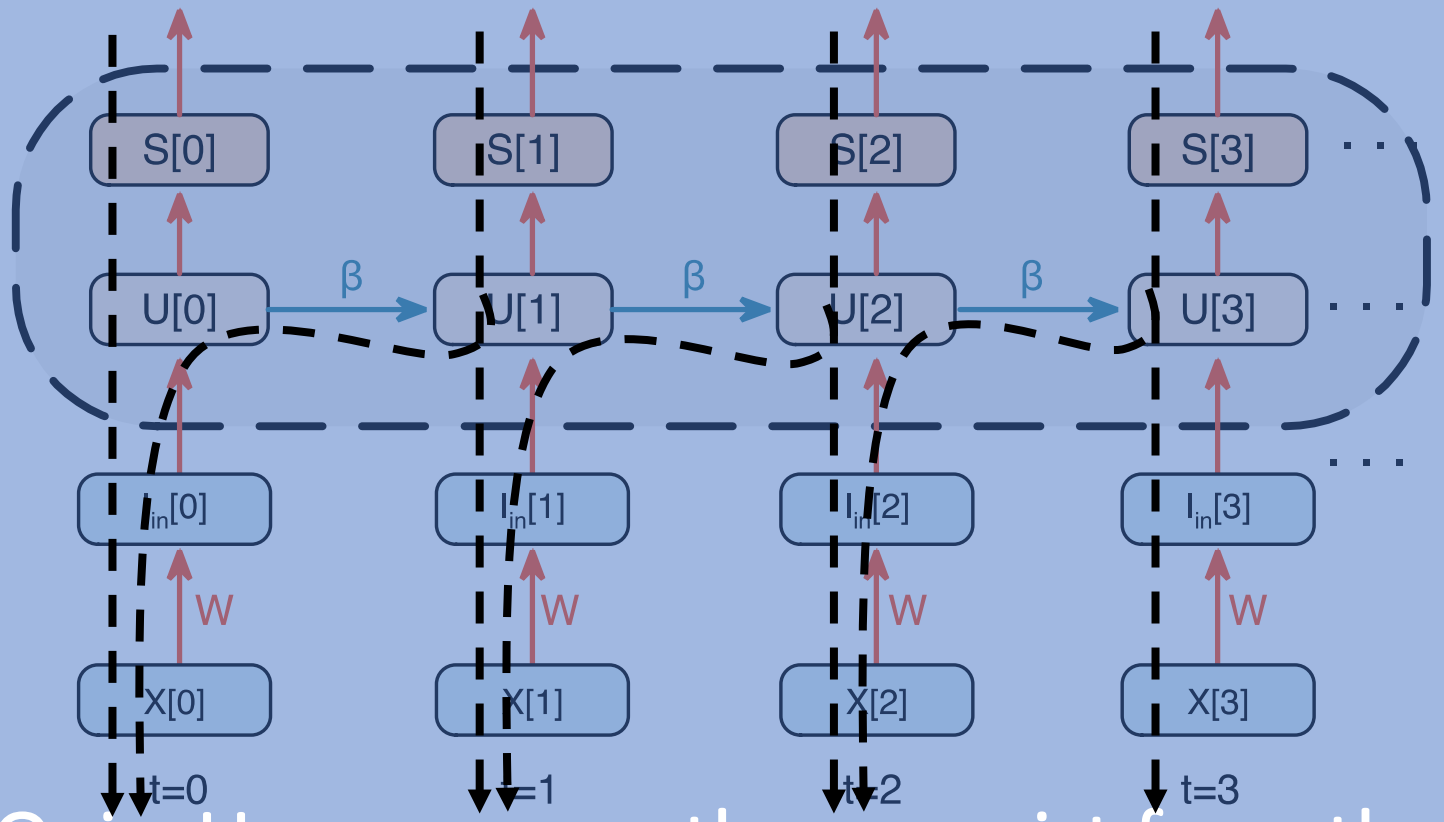
Pop Quiz: How many pathways exist from the outputs to W?

Hint #1: Here are 4 pathways

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

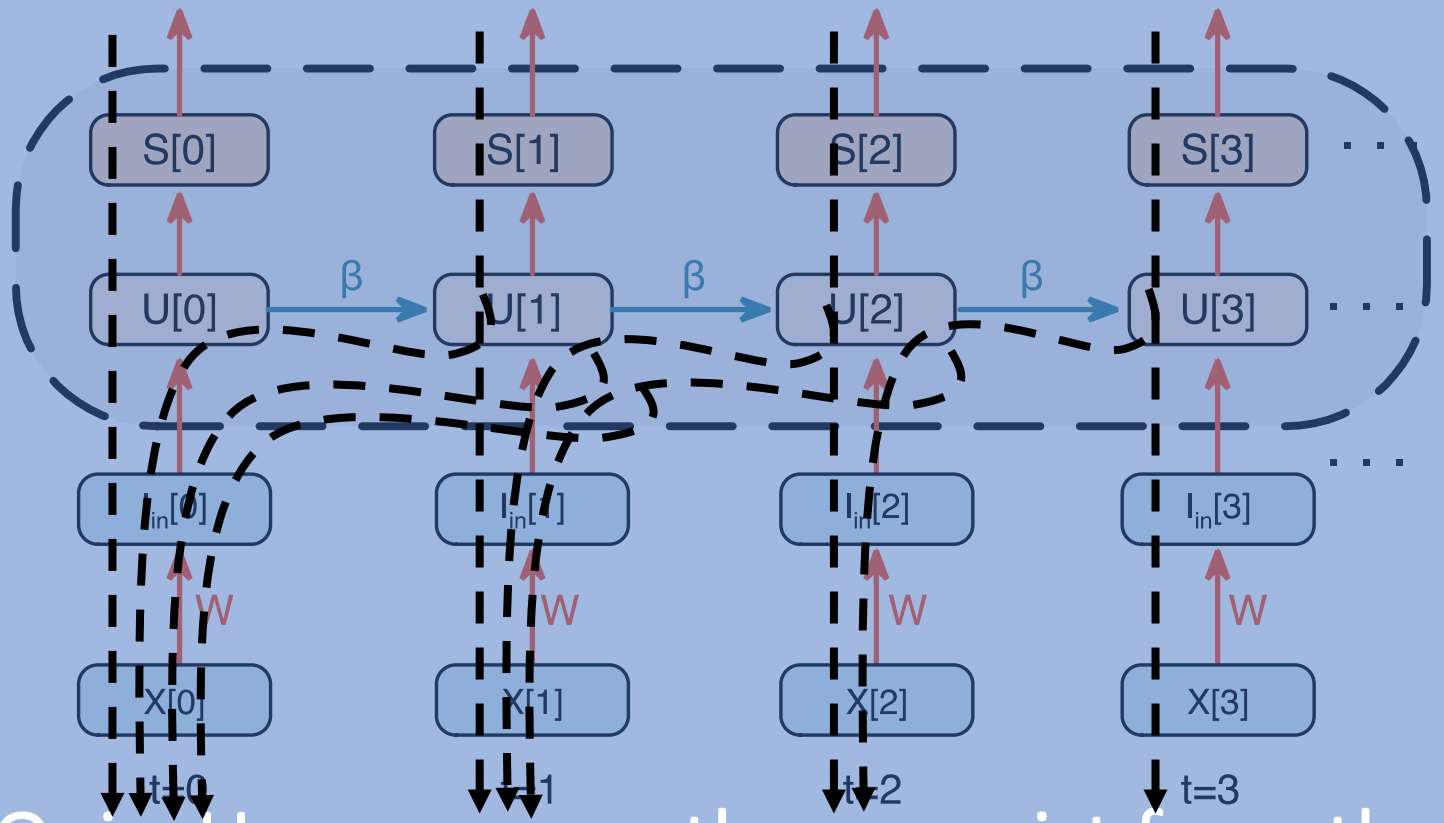
Pop Quiz: How many pathways exist from the outputs to W?

Hint #2: Here are 7 pathways

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

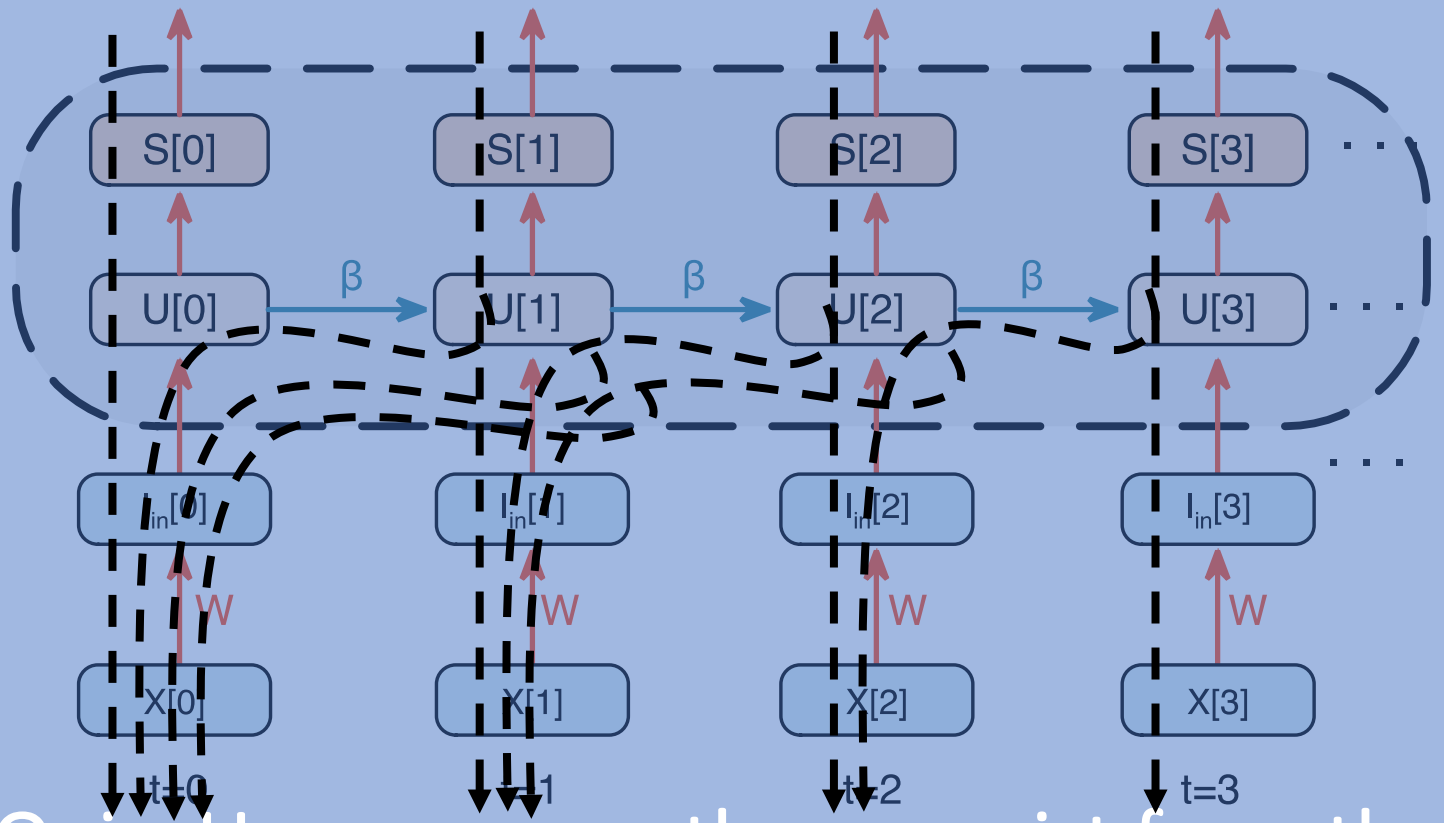
Pop Quiz: How many pathways exist from the outputs to W?

Answer: 10

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

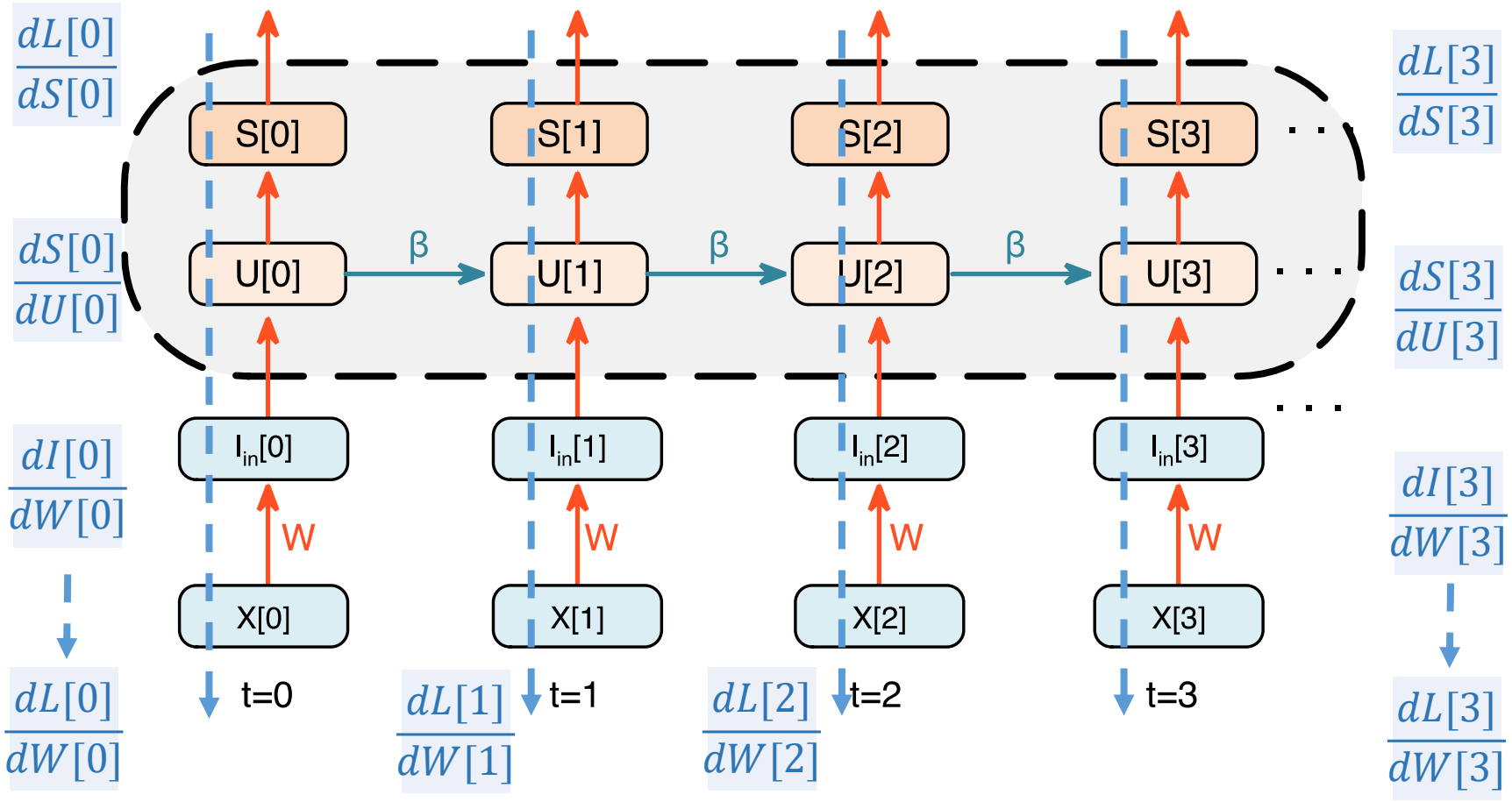
Pop Quiz: How many pathways exist from the outputs to W?

Answer: 10 - and this is just one neuron 😭

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

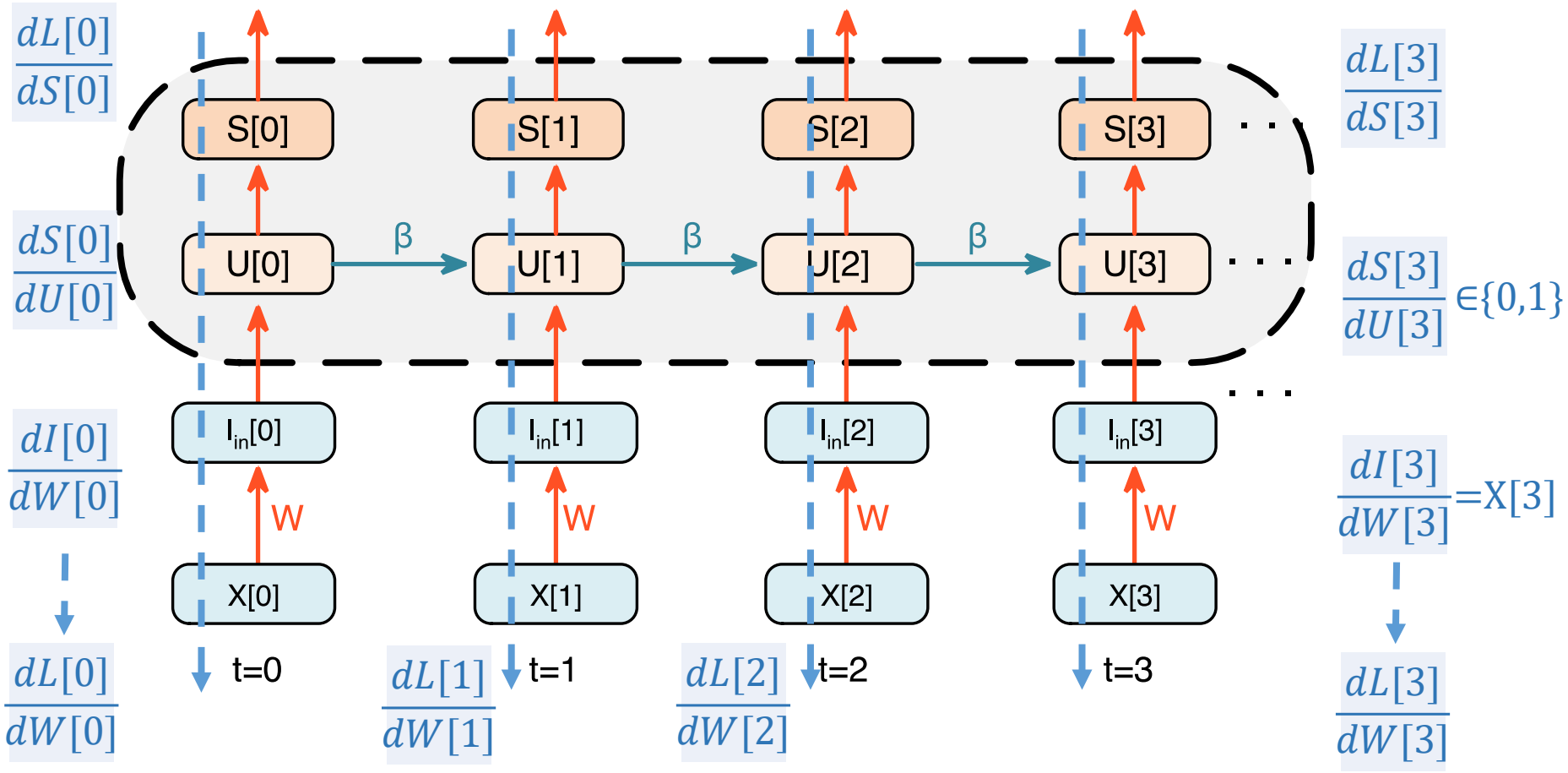
Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

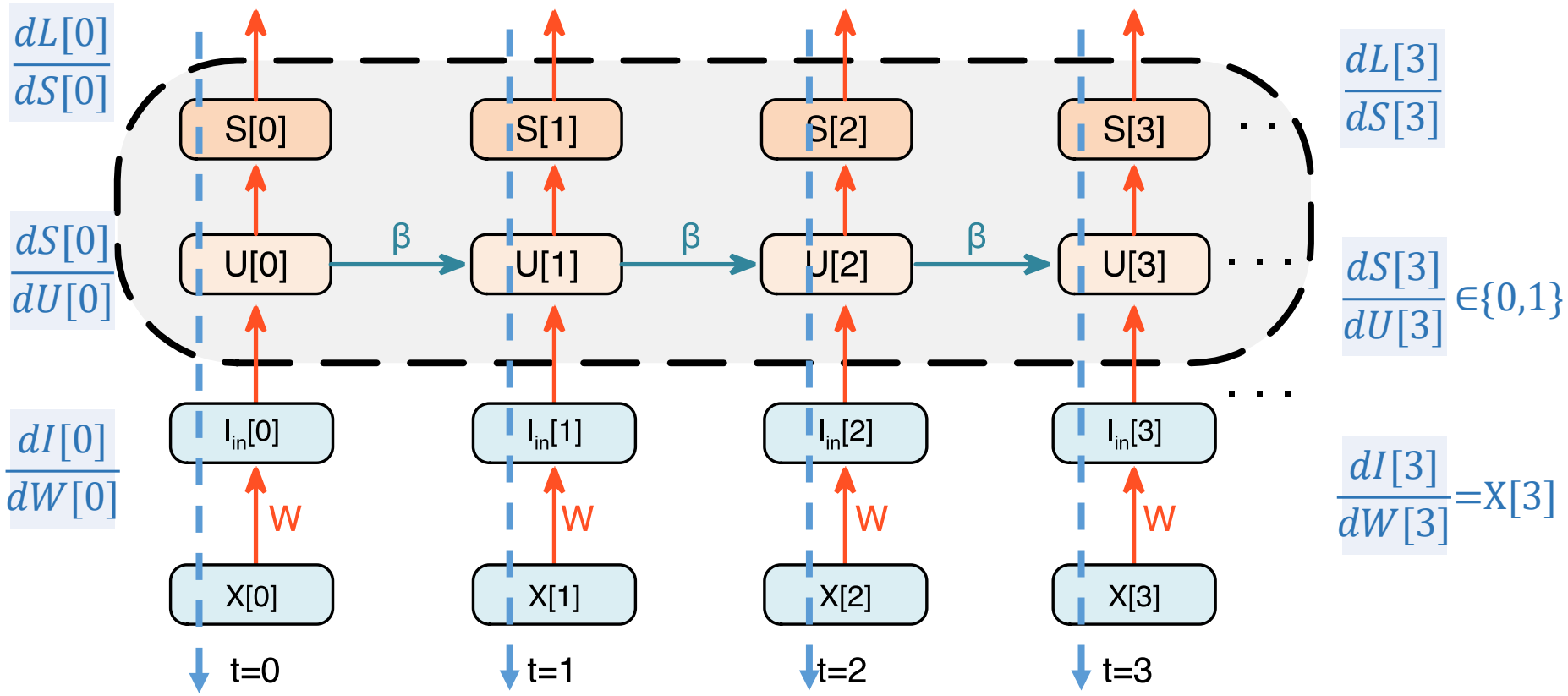
Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

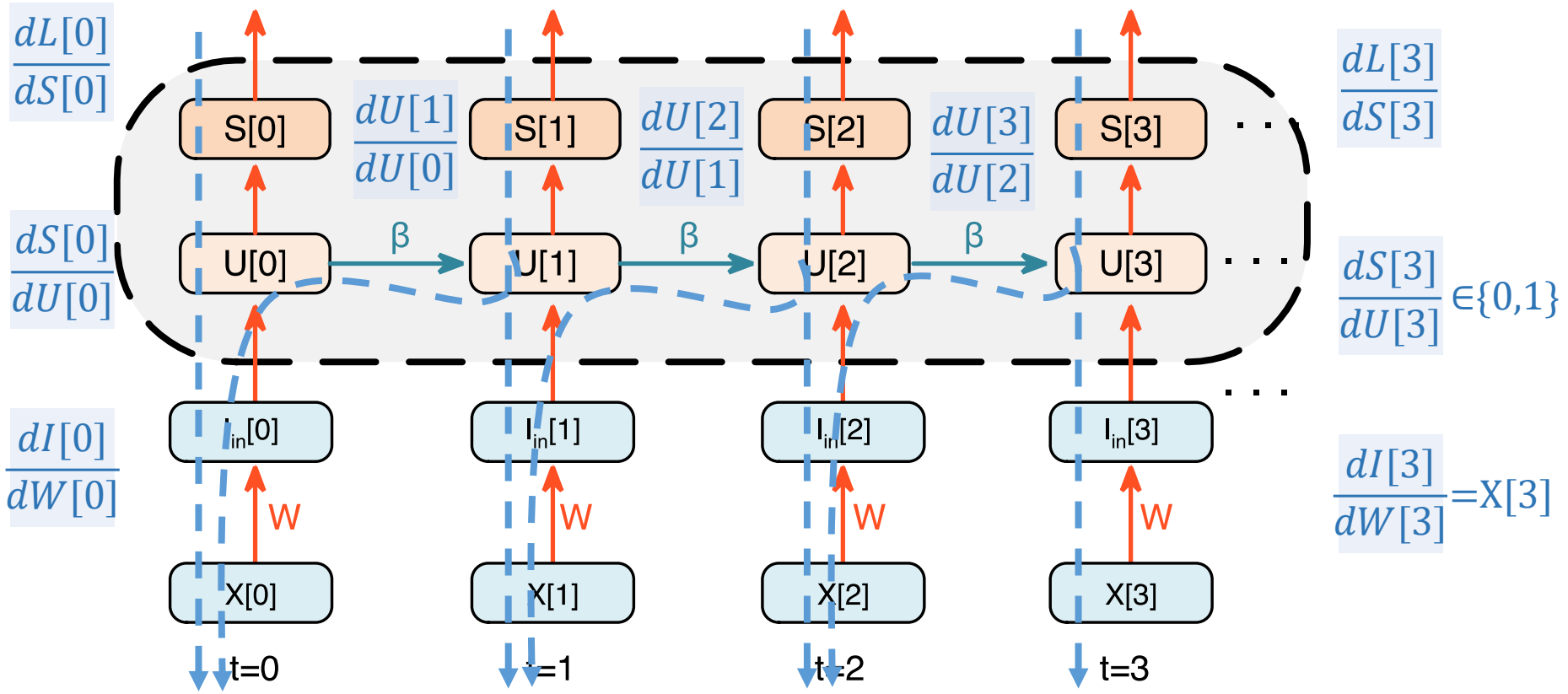
Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

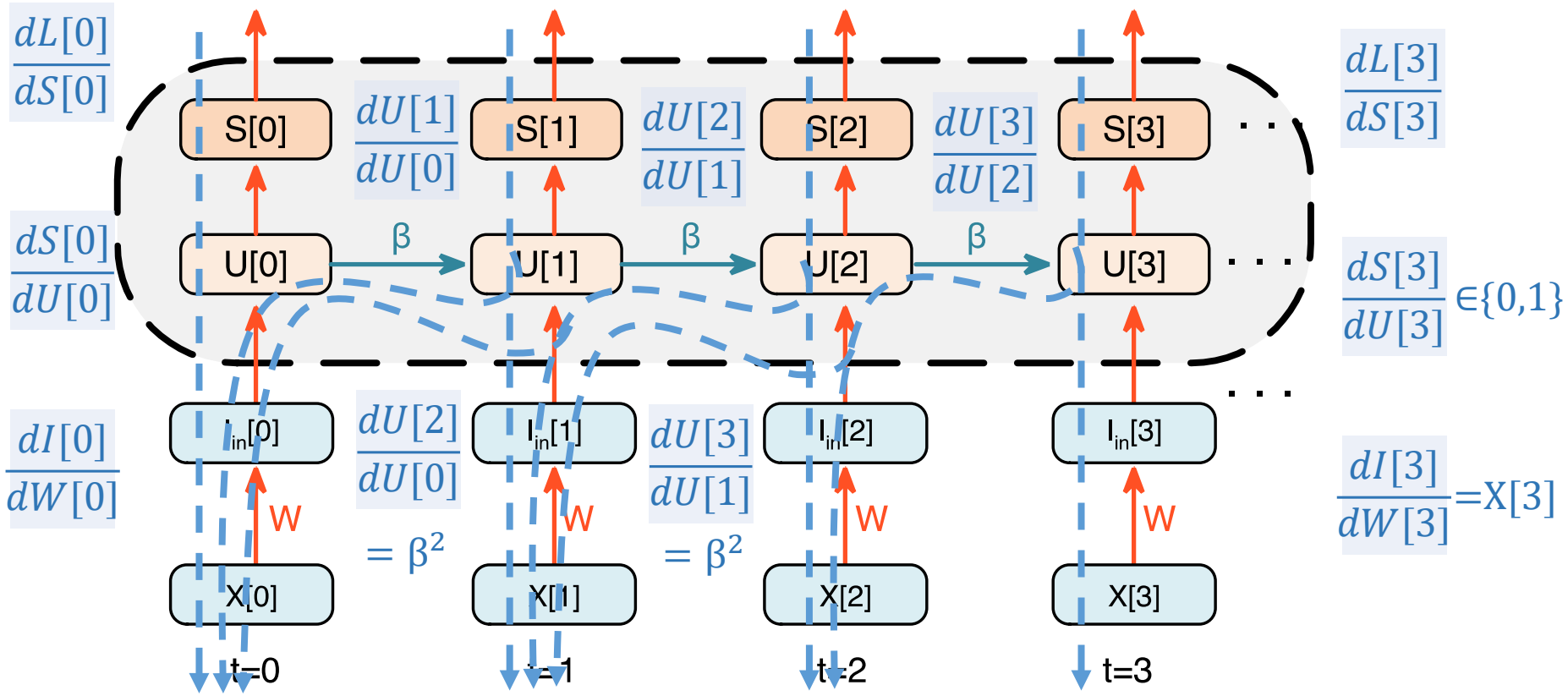
Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

$$S[t+1] = H(V[t+1] - V_{thr})$$

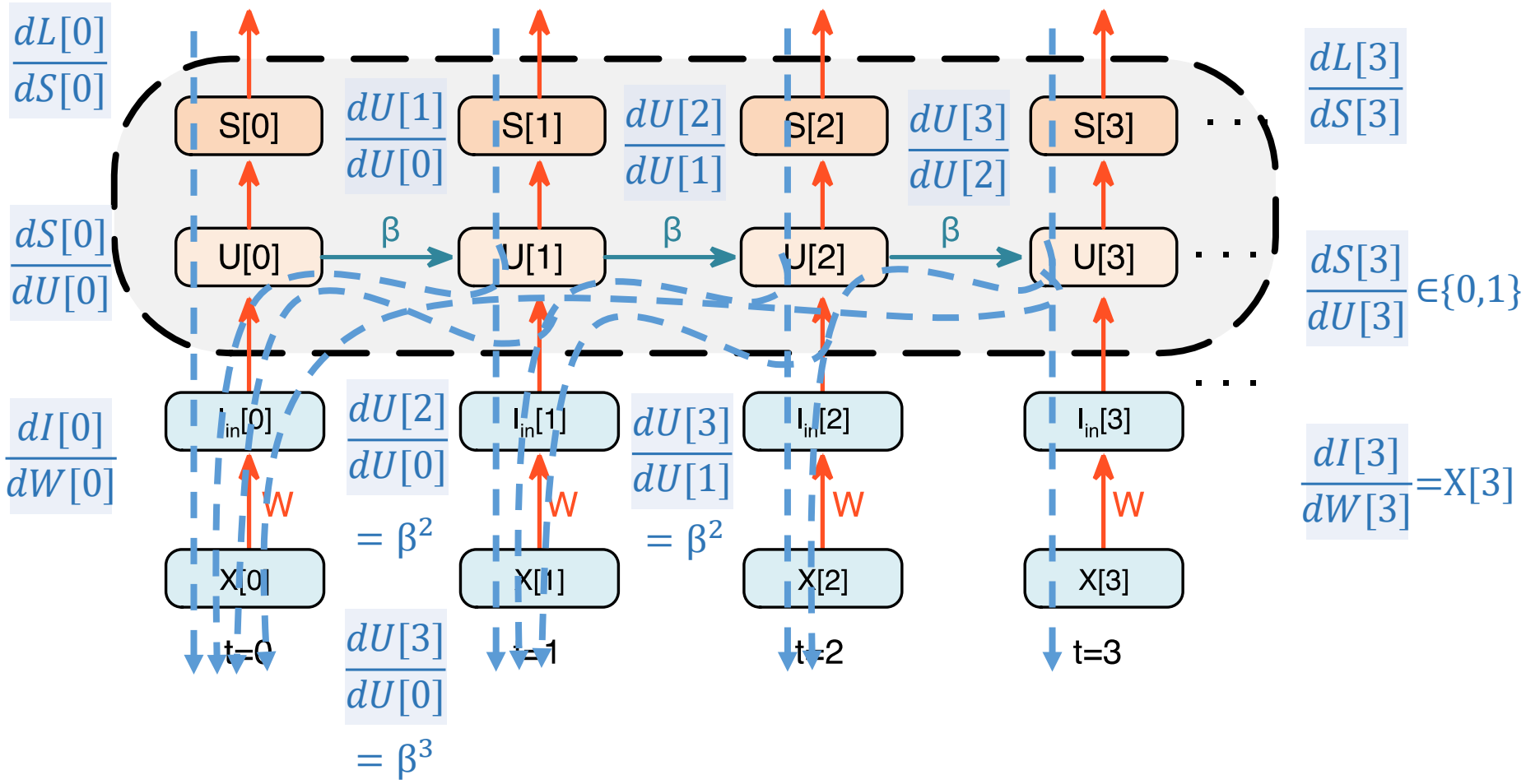
Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spiking Dynamics

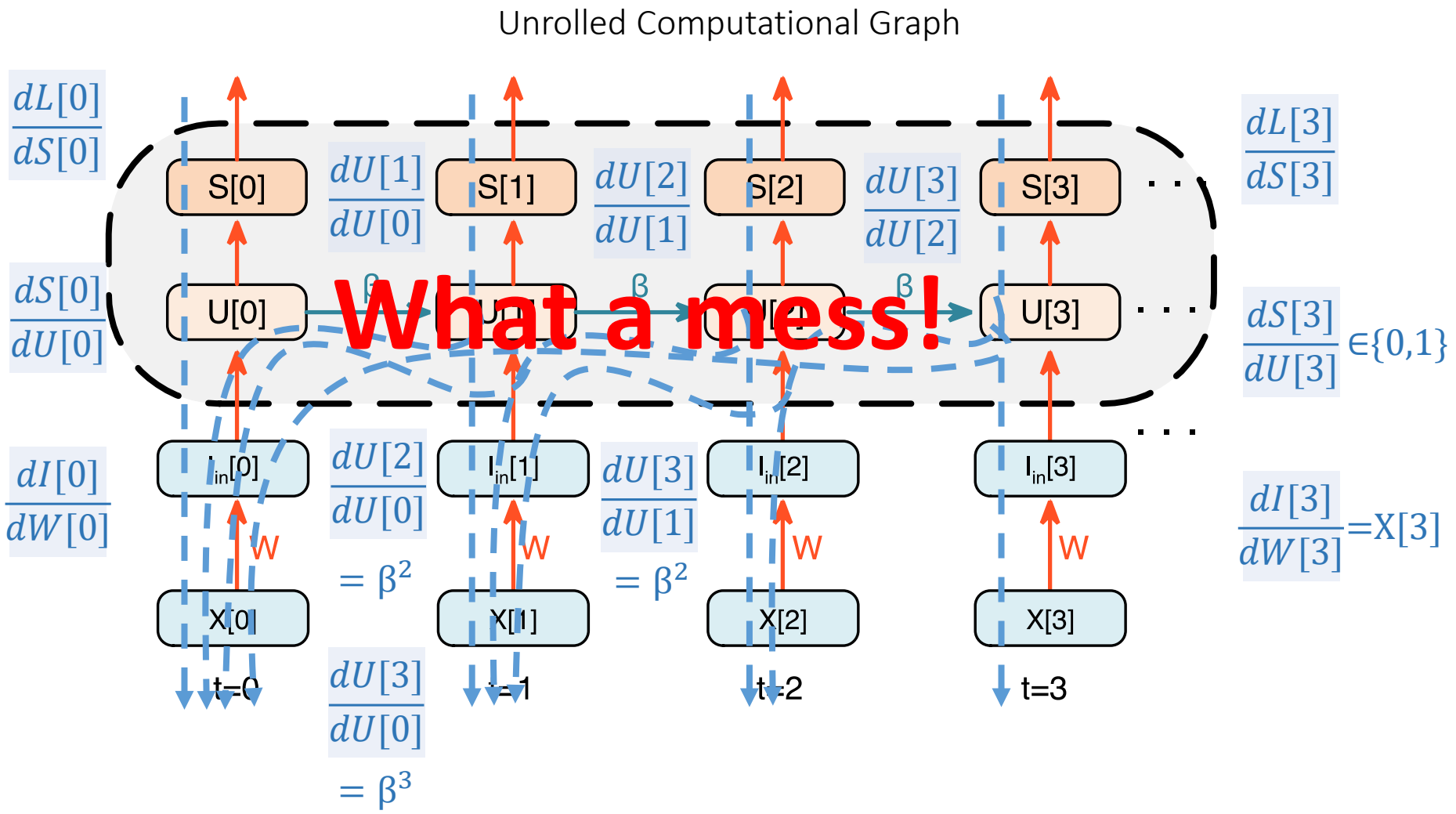
$$S[t+1] = H(V[t+1] - V_{thr})$$

Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time



Spiking Dynamics

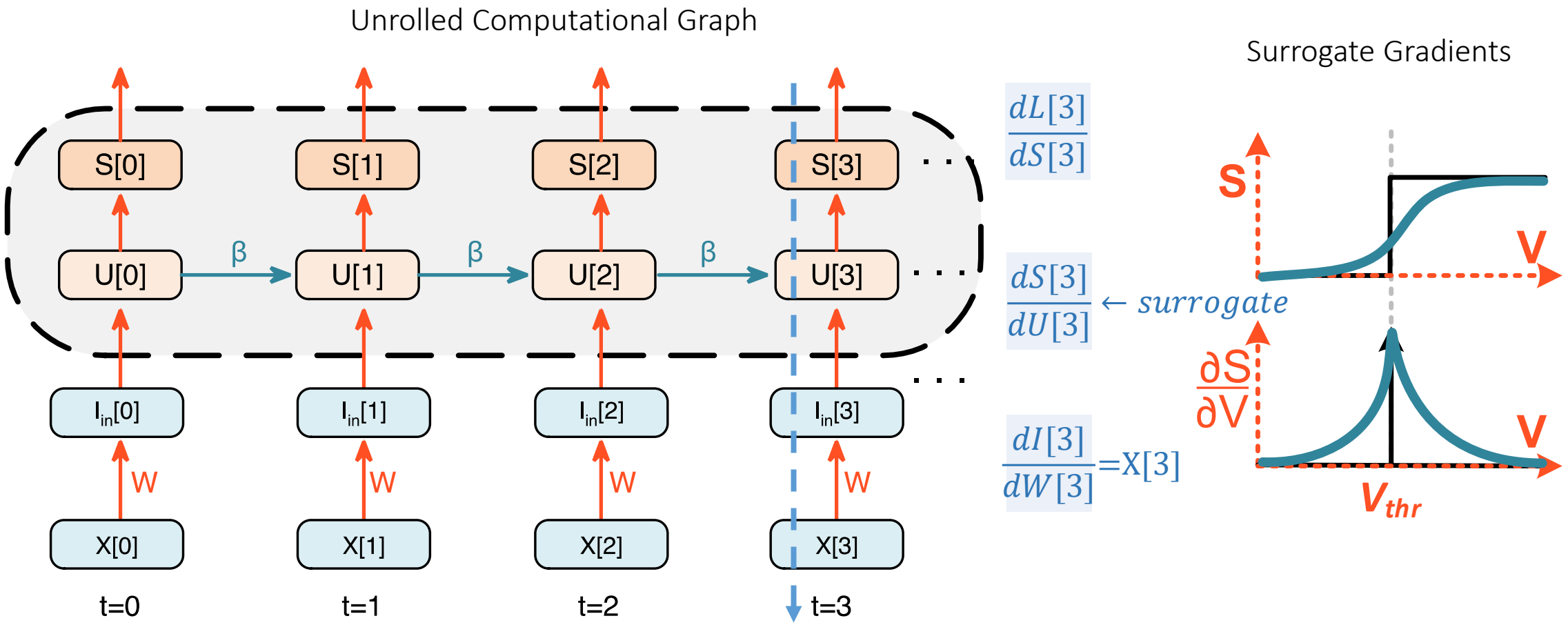
$$S[t+1] = H(V[t+1] - V_{thr})$$

Membrane Potential Dynamics

$$V[t+1] = \beta V[t] + X[t]W$$

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

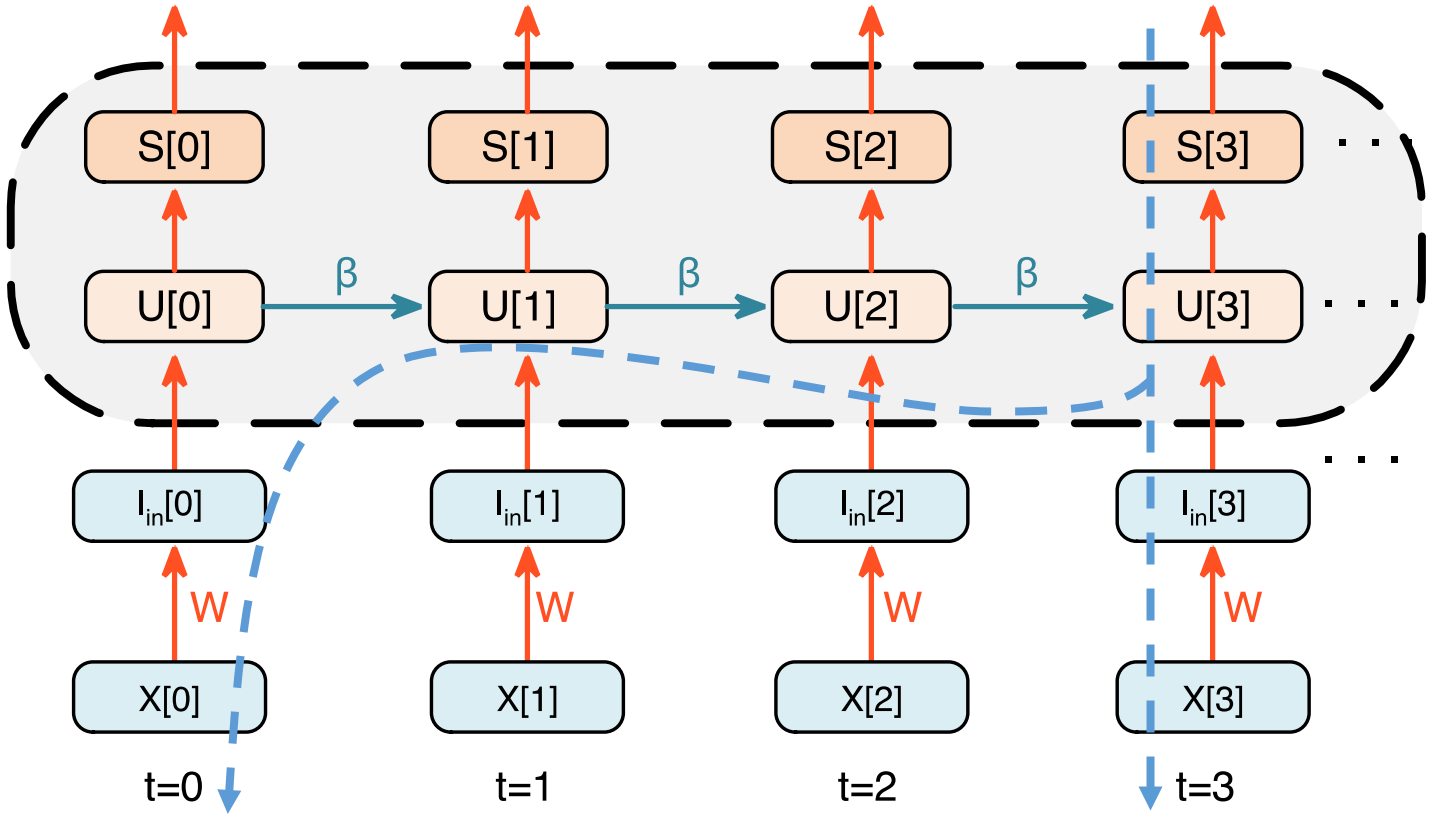
Spiking Backprop Through Time



J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Spiking Backprop Through Time

Unrolled Computational Graph



Spatial Credit Assignment
How does a loss assign 'blame' to a weight that is spatially far?

Temporal Credit Assignment
All states throughout history must be stored to run the BPTT algorithm i.e., the entire graph must be stored
Memory complexity: **O(nT)**

Update Locking
Weights are fixed during the forward-pass and only updated during training

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Real-Time Recurrent Learning

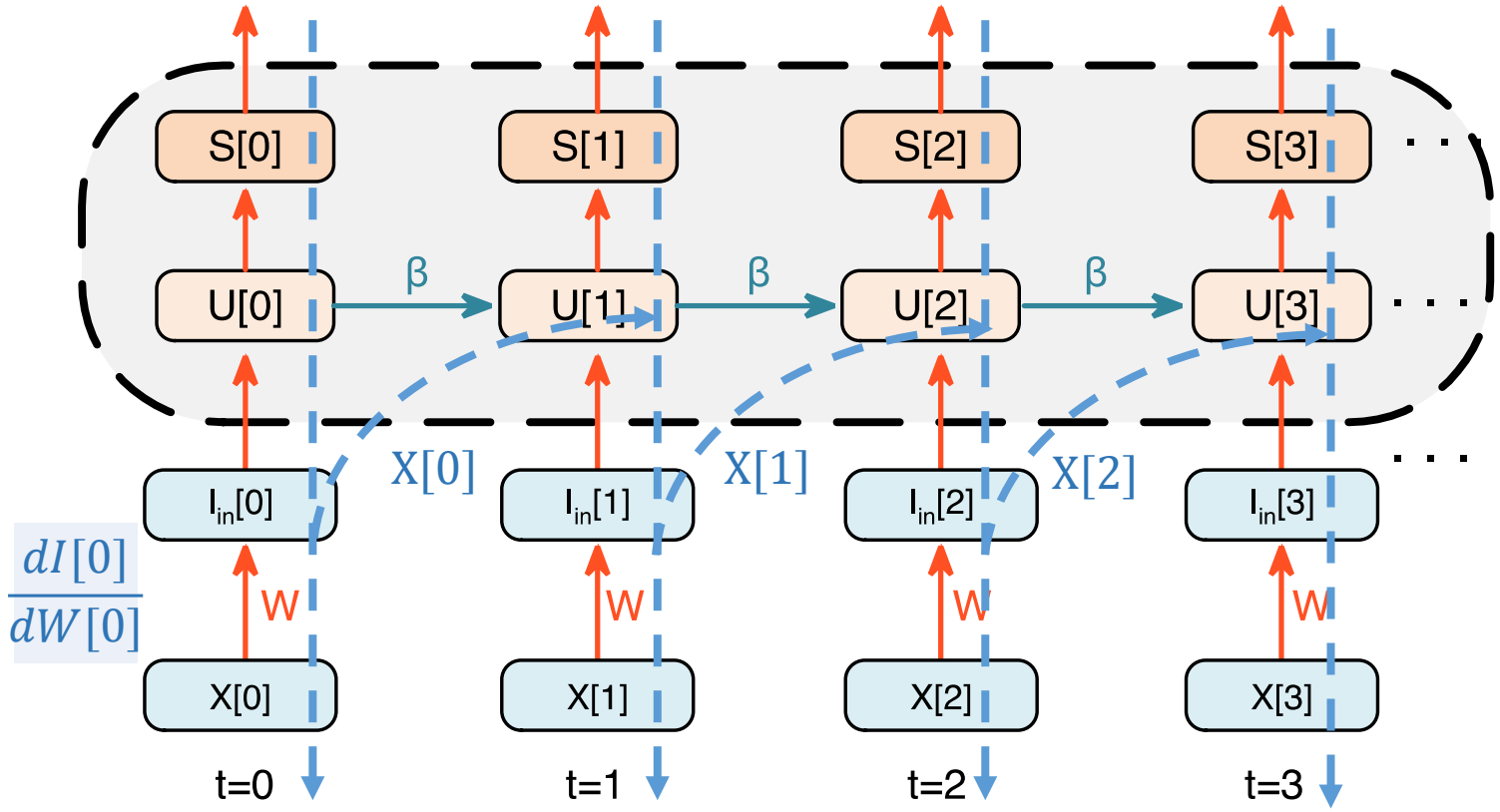
Problem with Backprop Through Time
The entire graph must be stored
Memory complexity: **O(nT)**

Forward-Mode learning addresses this –
gradients are snowballed forward in time

Real-Time Recurrent Learning
(Williams & Zipser, 1989)

RTRL was buried and somewhat forgotten because it comes at a cost of computational complexity: **$O(n^3)$**

Addressing temporal credit assignment problems of BPTT



J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Real-Time Recurrent Learning

How is RTRL modified?

- Adds feedback connections in the forward-pass
- Removes feedback connections during error propagation
- Surrogate gradients

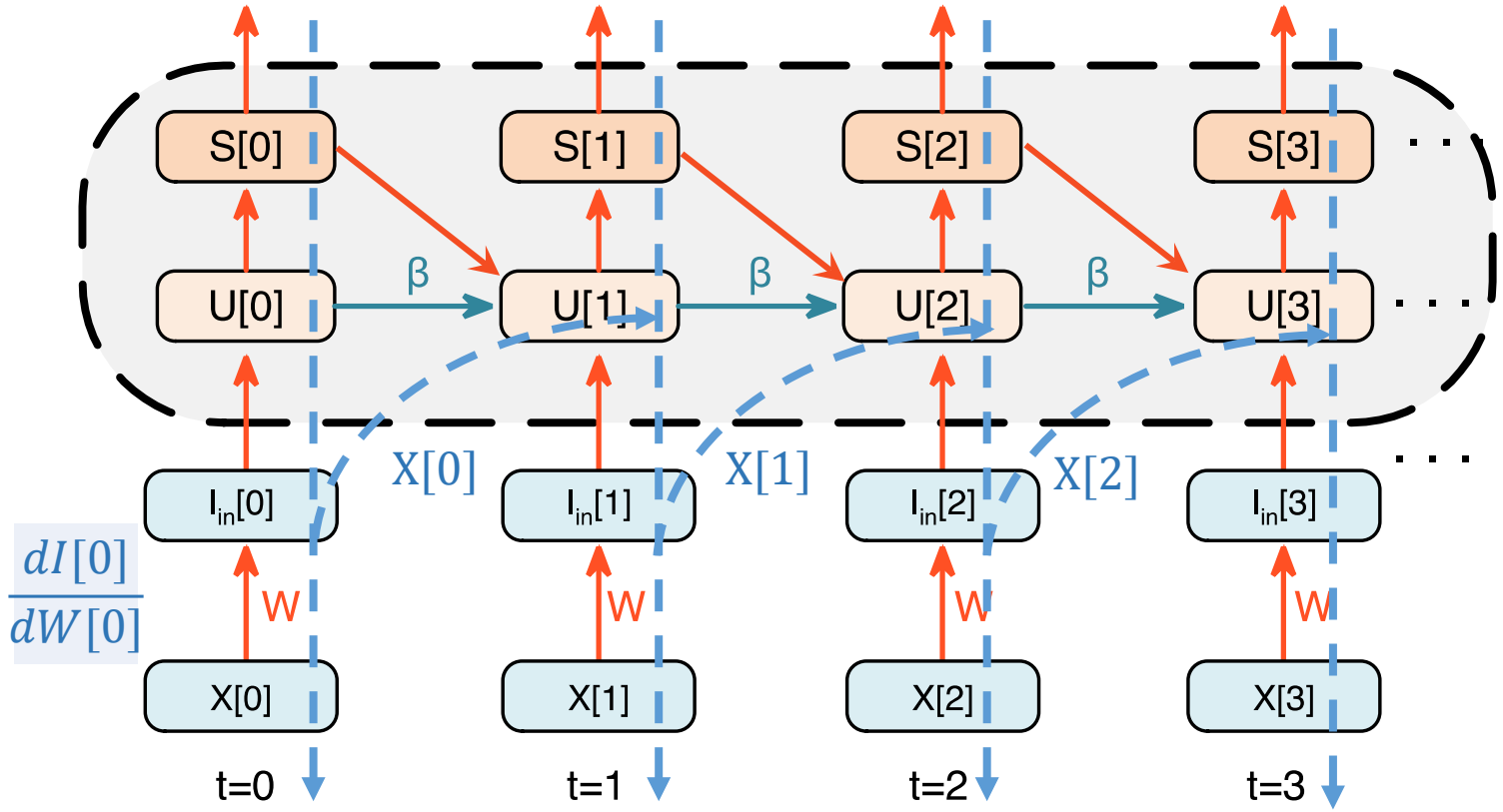
Pros (beyond RTRL benefits)

- Projecting data into a high-dimensional recurrent space acts like a reservoir

Cons

- Applying unbiased gradients to recurrent weights would be better for learning
- Analysis is constrained to a single layer

RTRL variants: e-prop (Bellec et al., 2020)



J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

G. Bellec et al., "A solution to the learning dilemma for recurrent networks of spiking neurons", Nat. Comms., 2020.

Real-Time Recurrent Learning

How is RTRL modified?

- Adds greedy local losses at **every** layer (Bengio et al., 2006)
- Errors only need to be propagated back to a single layer
- Surrogate gradients

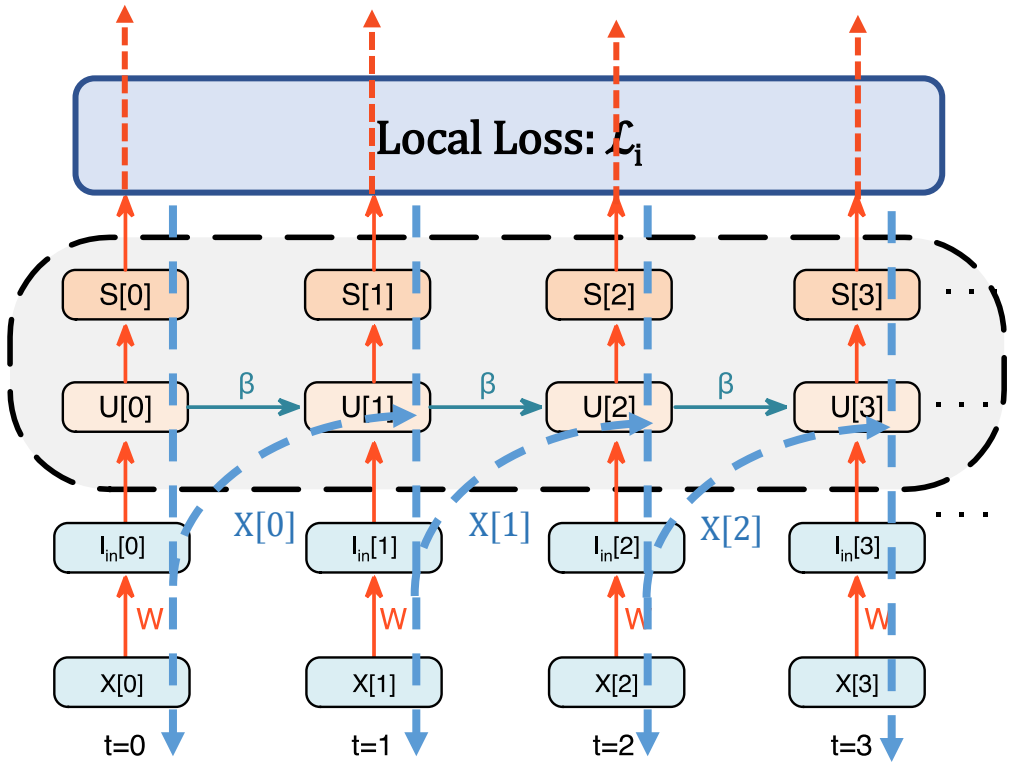
Pros (beyond RTRL benefits)

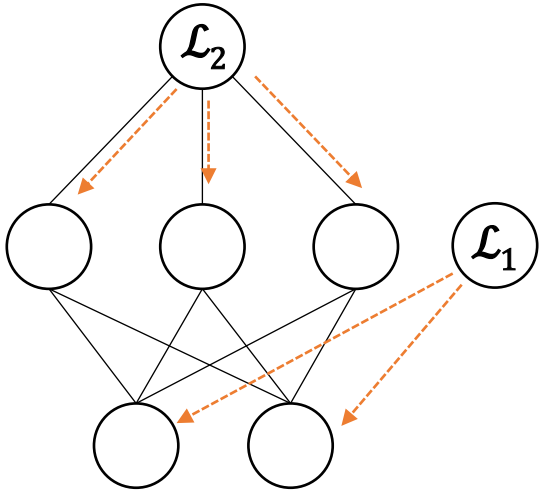
- Addresses spatial credit assignment using local losses from multiple objectives

Cons

- Not many problems can be cast into a form with a definable local loss
- Greedy local learning prioritizes immediate gains without considering an overall objective: often leads to suboptimal solutions

RTRL variants: decolle (Kaiser et al., 2020)





J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

J. Kaiser, H. Mostafa, E. Neftci, "Synaptic plasticity dynamics for deep continuous online learning", Front. Neurosci., 2020.

Real-Time Recurrent Learning

How is RTRL modified?

- OSTL + Direct random target projection (Frenkel & Lefebvre et al., 2019)
- Surrogate gradients

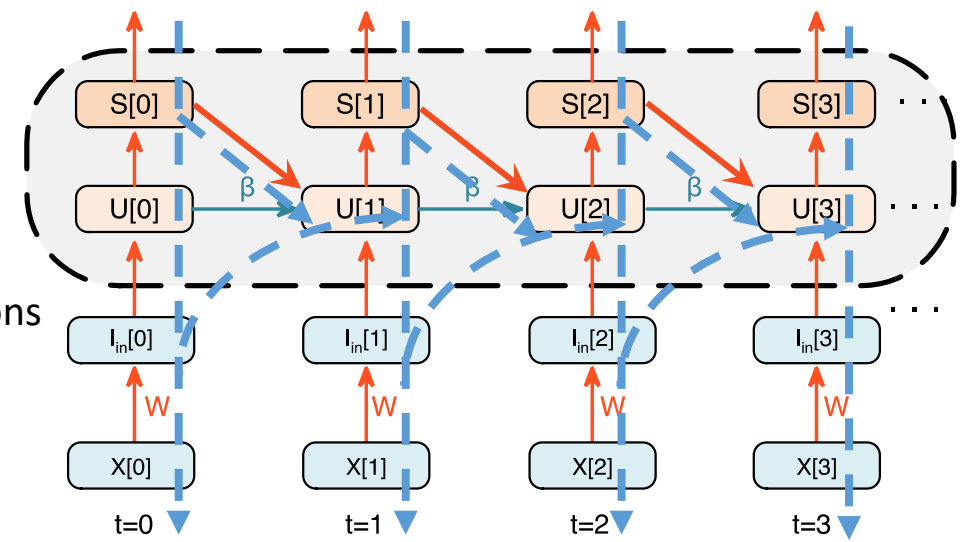
Pros (beyond RTRL benefits)

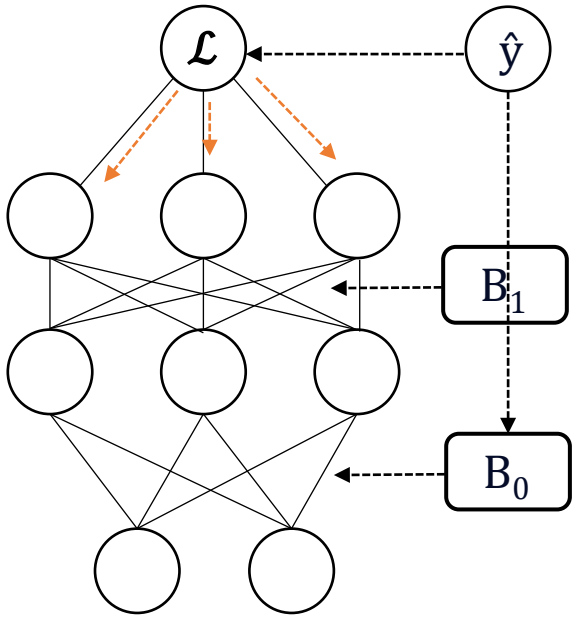
- Addresses spatial credit assignment using local signals from a target $\hat{y}$
- Able to generalize to multiple recurrent connections

Cons

- DRTP prioritizes immediate gains without considering an overall objective: often leads to suboptimal solutions

RTRL variants: OSTTP (Ortner & Pes, et al., 2023)





J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Ortner, Pes, Gentinetta, Frenkel, Pantazi, et al., "Online Spatiotemporal Learning with Target Projection", IEEE AICAS, 2023.

Real-Time Recurrent Learning

How is RTRL modified?

- OSTL + Direct random target projection (Frenkel & Lefebvre et al., 2019)
- Temporal gradient components do not propagate across layers (maroon) while spatial gradient components do (blue)
- Surrogate gradients

Pros (beyond RTRL benefits)

- Addresses spatial credit assignment using local signals from a target $\hat{y}$
- Able to generalize to multiple recurrent connections

Cons

- Directly applying losses to all layers faces the same challenges as greedy local learning: immediate gains are prioritized over more complex objectives

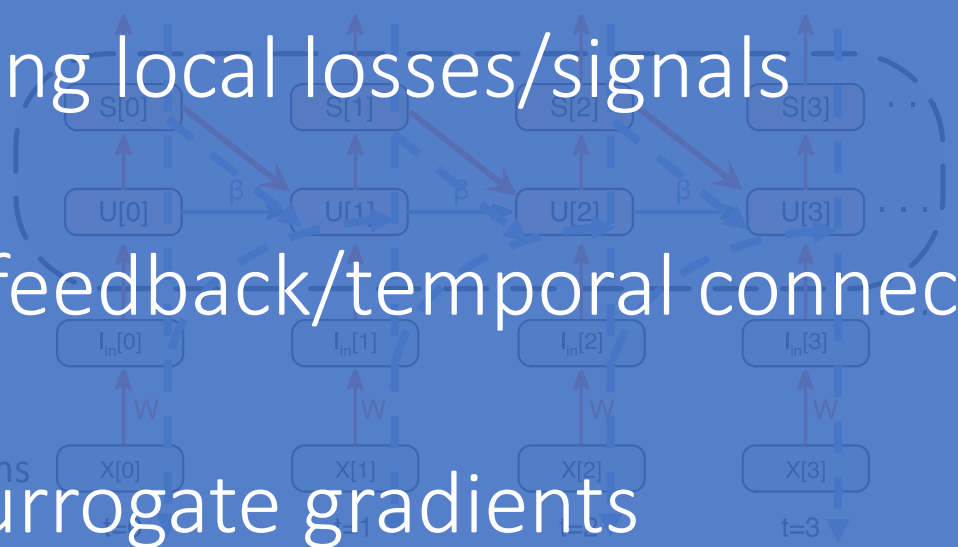
The trend is to approximate RTRL by:

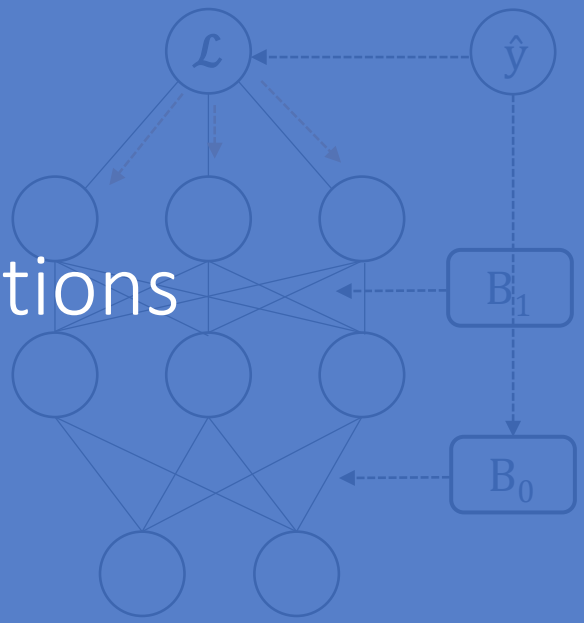
RTRL variants: OSTTP (Ortner & Pes, et al., 2023)

Applying local losses/signals

Approximating feedback/temporal connections

Surrogate gradients



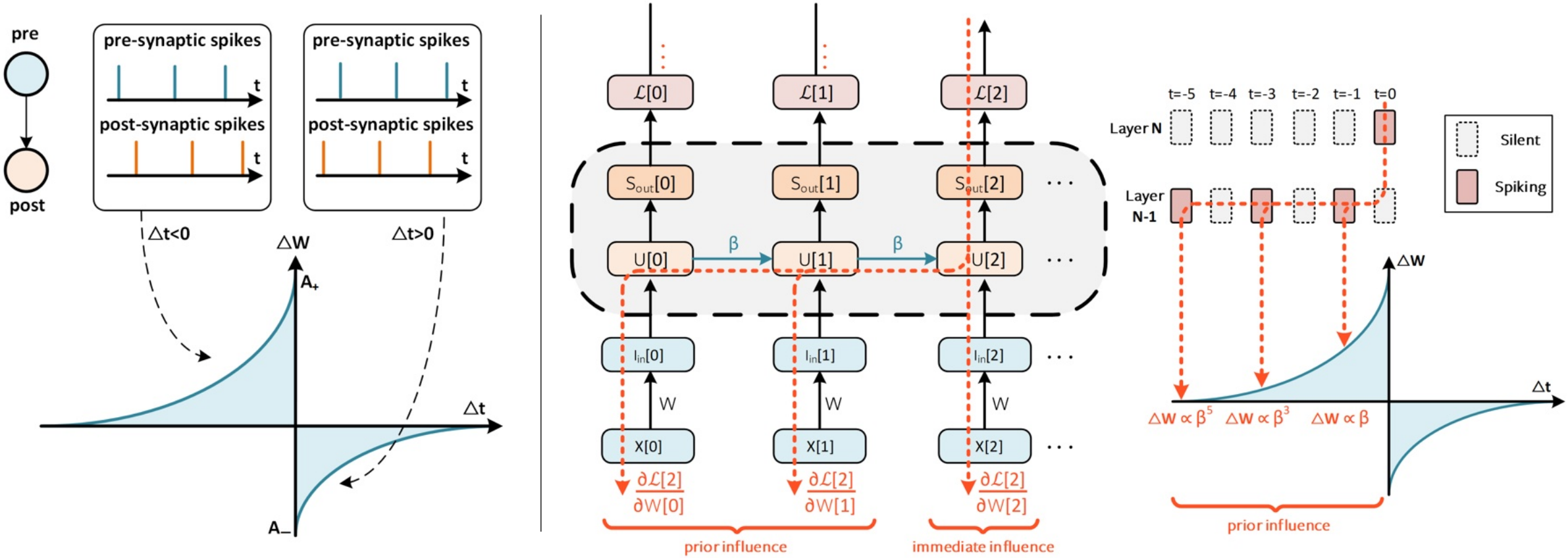


Scaling these approaches without blowing up complexity & maintaining good performance is hard.

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning", arXiv 2021.

Ortner, Pes, Gentinetta, Frenkel, Pantazi, et al., "Online Spatiotemporal Learning with Target Projection", IEEE AICAS, 2023.

The Link Between STDP & Backprop



Low-level neuronal dynamics influence high-level system behavior

J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons from Deep Learning" Proc. IEEE, 2023.

snnTorch

Gradient-based Learning with Spiking Neural Networks

Python package for gradient-based optimization of SNNs

real-time online learning

seamless integration with PyTorch

CUDA + IPU accelerated

neuromorphic HW compatible

github.com/jeshraghian/snntorch

Tutorial	Title	Colab Link
Tutorial 1	Spike Encoding with snnTorch	Open in Colab
Tutorial 2	The Leaky Integrate and Fire Neuron	Open in Colab
Tutorial 3	A Feedforward Spiking Neural Network	Open in Colab
Tutorial 4	2nd Order Spiking Neuron Models (Optional)	Open in Colab
Tutorial 5	Training Spiking Neural Networks with snnTorch	Open in Colab
Tutorial 6	Surrogate Gradient Descent in a Convolutional SNN	Open in Colab
Tutorial 7	Neuromorphic Datasets with Tonic + snnTorch	Open in Colab

Advanced Tutorials	Colab Link
Population Coding	Open in Colab
Regression: Part I - Membrane Potential Learning with LIF Neurons	Open in Colab
Regression: Part II - Regression-based Classification with Recurrent LIF Neurons	Open in Colab
Accelerating snnTorch on IPUs	–

Python package for gradient-based optimization of SNNs

real-time online learning

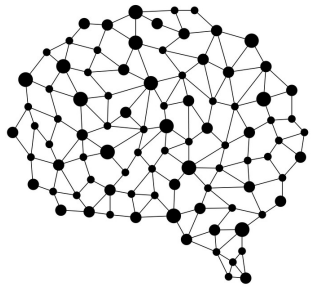
seamless integration with PyTorch

CUDA + IPU accelerated

neuromorphic HW compatible

github.com/jeshraghian/snntorch

How much should we plagiarize the brain?



High-level: Interconnections and systems

Low-level: Neurons and Ion Channels

The Brain is a Predictive Machine

Predictive coding tells us that the brain encodes the world around us into a "predictive" belief system.

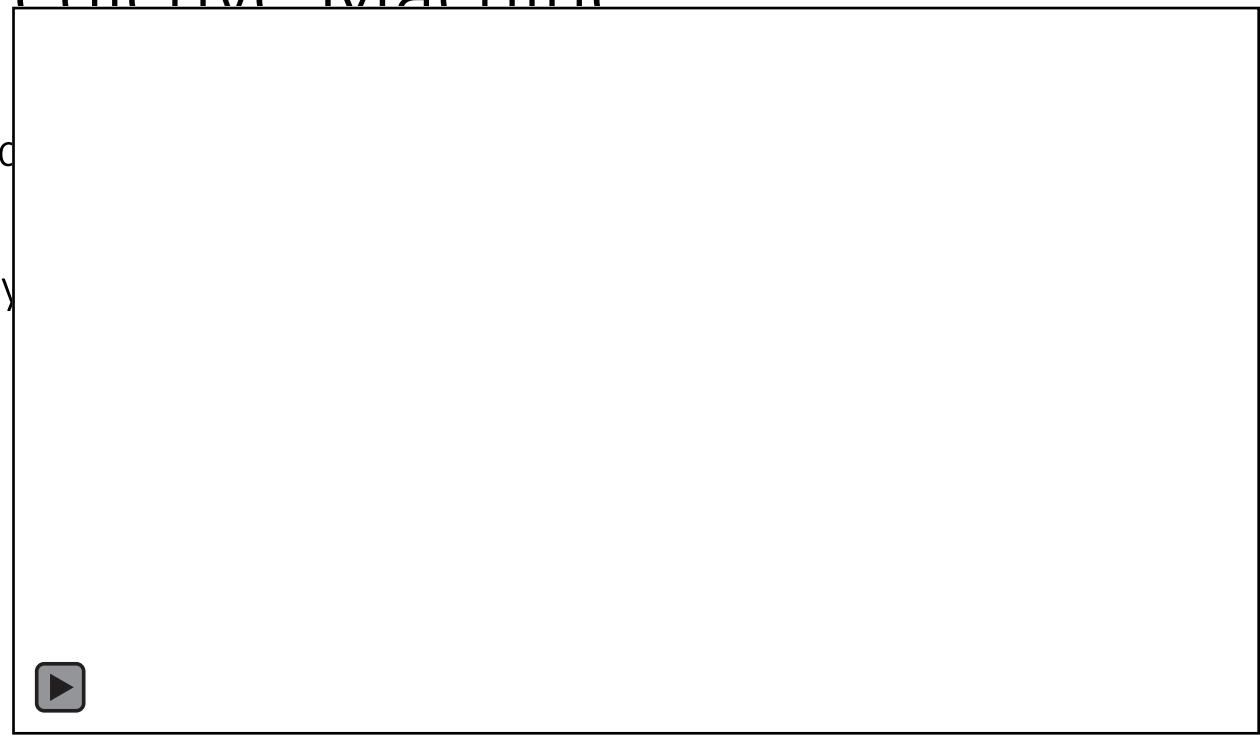
- If the prediction is correct, the don't waste energy processing the input.
- If the prediction is wrong, then learn to do better

Knowledge Distillation Through Time

- To predict the future, use the present.
- Aiming for 100% success is wasteful. Aim to converge to detection performance instead

Benefits:

- Contextual adaptation
- Annotation free learning



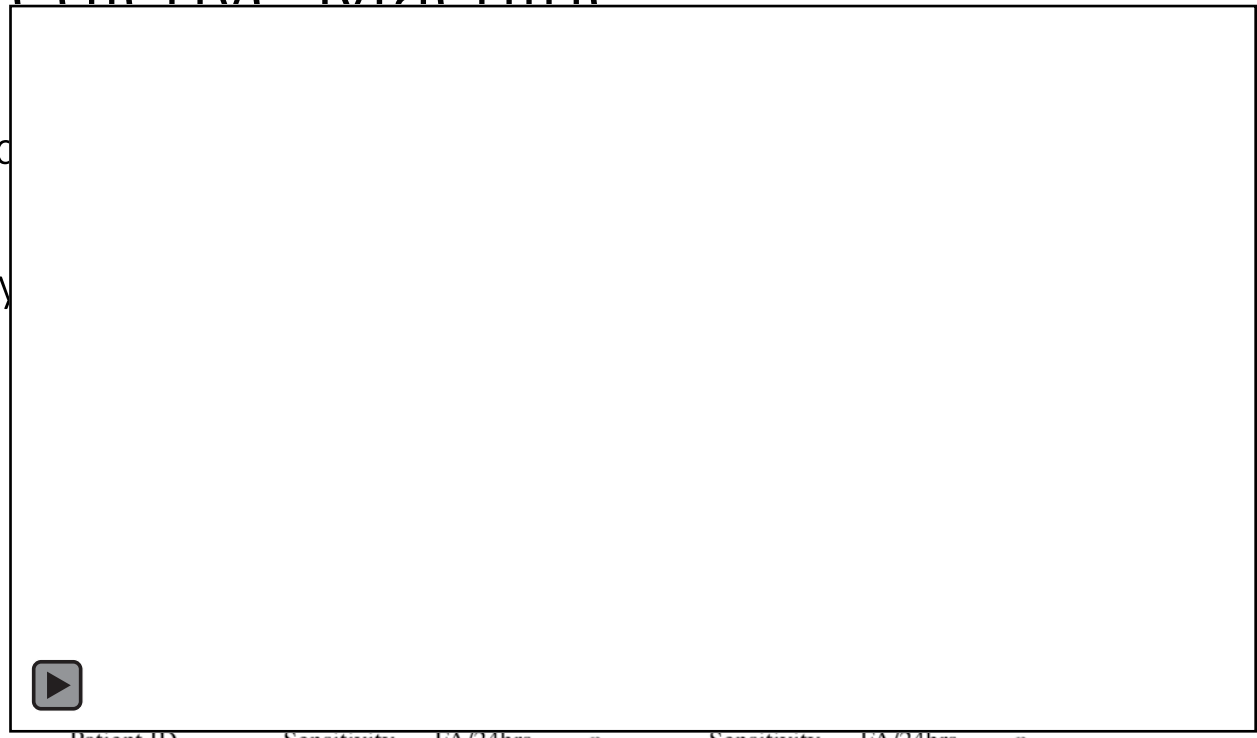
Gunasekaran & Eshraghian, ICLR, 2024.

The Brain is a Predictive Machine

Predictive coding tells us that the brain encodes the world around us into a "predictive" belief system.

- If the prediction is correct, the don't waste energy processing the input.
- If the prediction is wrong, then learn to do better

Knowledge Distillation Through Time



Results Summary

- p-values are quite large due to the small seizure count p/patient
- Sensitivity and FA/24hrs either improve or remain constant for every case

Patient ID	Sensitivity	FA/24hrs	p	Sensitivity	FA/24hrs	p
Focal Pat–NSG						
1	33.33%	5.36	0.488	**66.67%**	**4.74**	0.21
2	50.00%	14.88	0.625	**75.00%**	**12.87**	0.57
3	100.00%	2.72	0.110	**100.00%**	**2.04**	0.10
4	67.67%	4.38	0.077	67.67%	**3.93**	0.18
5	42.86%	4.92	0.131	**57.14%**	**4.02**	0.24
6	33.33%	8.14	0.558	33.33%	**6.87**	0.31
7	71.43%	15.03	0.180	71.43%	**12.88**	0.63
8	50.00%	2.56	0.190	50.00%	**1.54**	0.116
9	50.00%	7.97	0.482	50.00%	**4.58**	0.20
10	100.00%	16.64	0.125	**100.00%**	**13.87**	0.61
Total	55.26%	8.21	0.36	**63.16%**	**6.60**	0.30

Gunasekaran & Eshraghian, ICLR, 2024.

Tutorial Outline

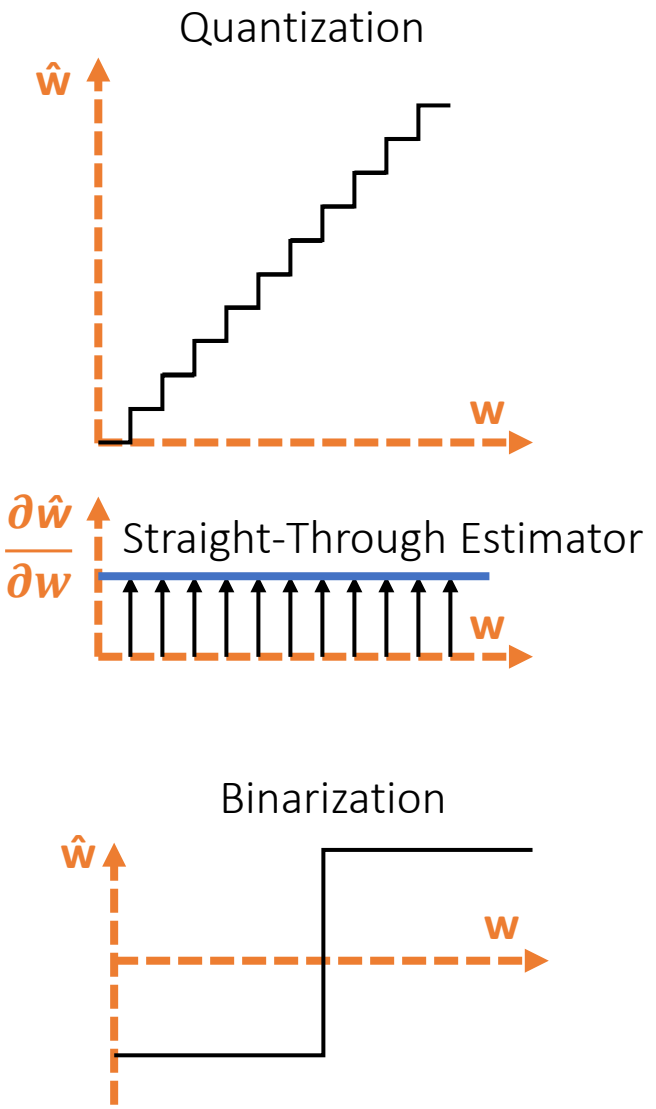
1. **Spiking Neurons**
2. **How to Train Your Spiking Neural Net**
   a. Local Learning
   b. Shadow Training
   c. Spike Time Learning
   d. Backprop Through Time and Surrogate Gradients
   e. Spike Time Learning
   f. Real-Time Recurrent Learning
3. **Low Precision Training**
4. **Perspectives: How can we do better?**

Low-Precision Training

- Full precision is computationally more expensive than fixed precision/binarized operations
    a. Post-Training Quantization
    b. Quantization-Aware Training

Quantization-Aware Training Algorithm

1. Initialize full precision weights
2. Quantize during the forward-pass to obtain a "quantization-aware" loss
3. Replace non-differentiability with a "straight-through estimator" (Hinton, 2011)
4. Apply weight update to full precision weight
5. Repeat steps 2-5 till GPUs are melting.

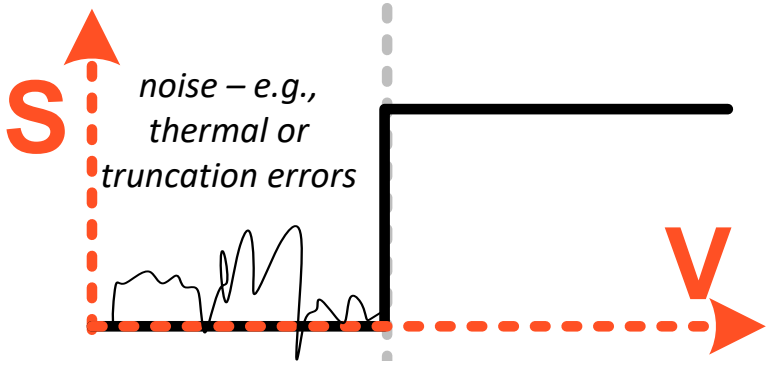


J.K. Eshraghian et al., "Memristor based Binarized Spiking Neural Networks" IEEE Nanotech. Mag. 2022

Low-Precision Training

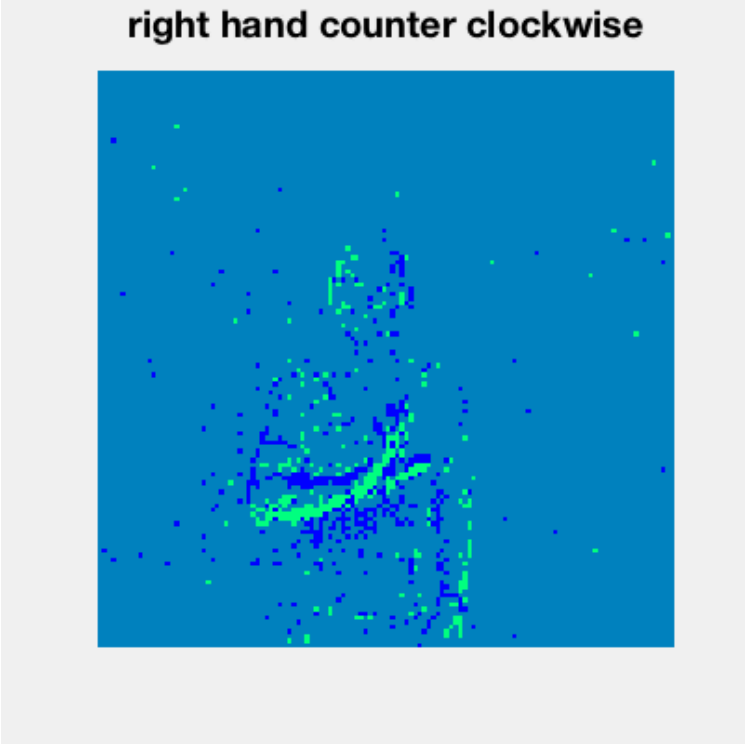
SNNs are remarkably tolerant to extreme quantization

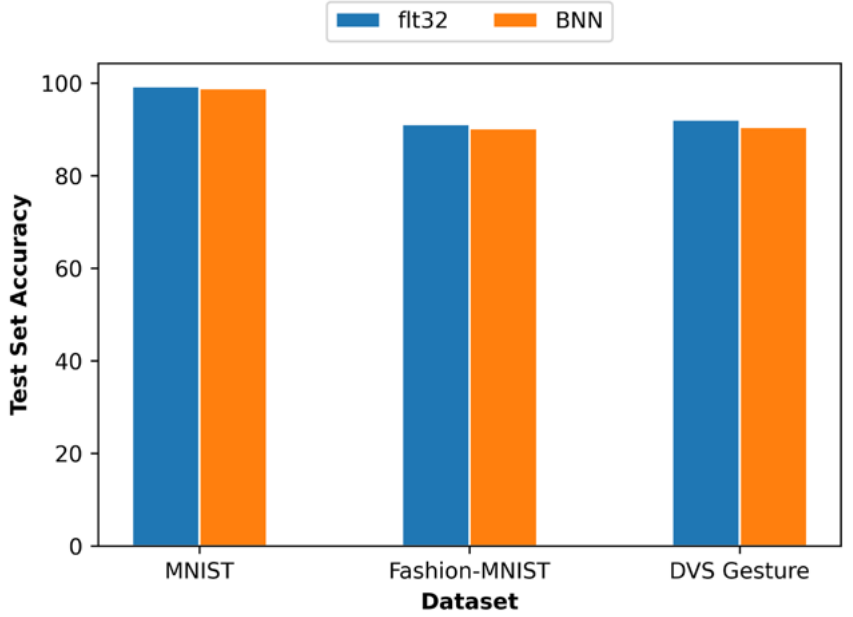
- Thresholding introduces challenges in gradient-based optimization, but it also promotes error resilience
- **Noise can effectively be absorbed into the subthreshold dynamics of spiking neurons**



Accuracy Degradation:

- MNIST – 0.3%
- FMNIST – 0.8%
- DVS Gesture – 1.05%

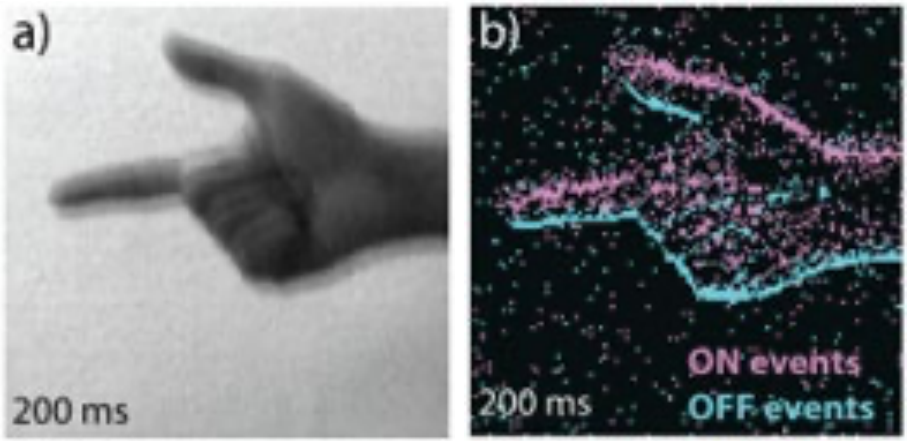


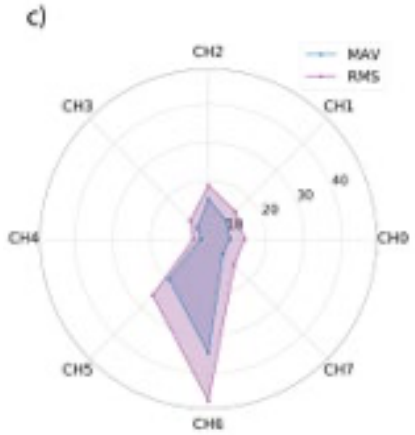


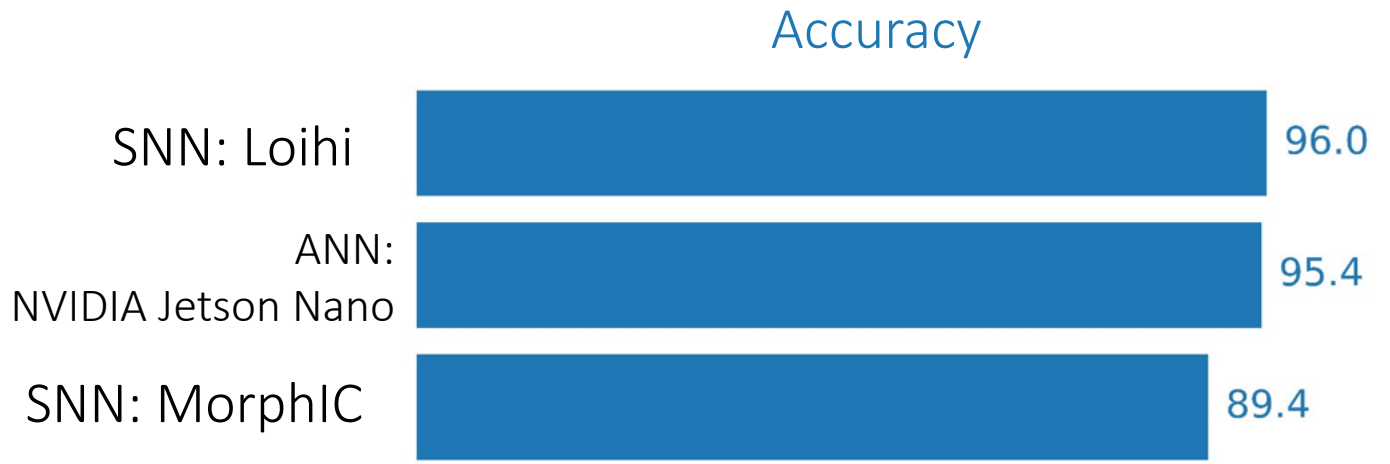
Binarized Spiking Neural Networks

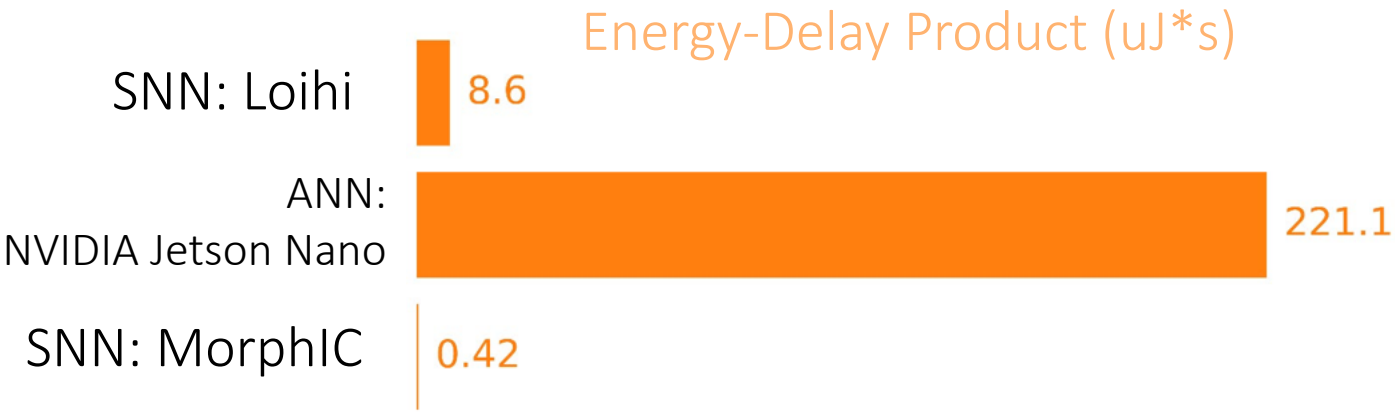
J.K. Eshraghian et al., "Memristor based Binarized Spiking Neural Networks" IEEE Nanotech. Mag. 2022

SNN Evaluation – Multimodal Data









M. Rahimi-Azghadi, J.K. Eshraghian et al., IEEE Trans. Biomedical CAS, Dec. 2020

SpikeGPT

The largest SNN trained via backprop & the first to perform language generation

- 3 variants: 45M, 120M, 260M Parameter models

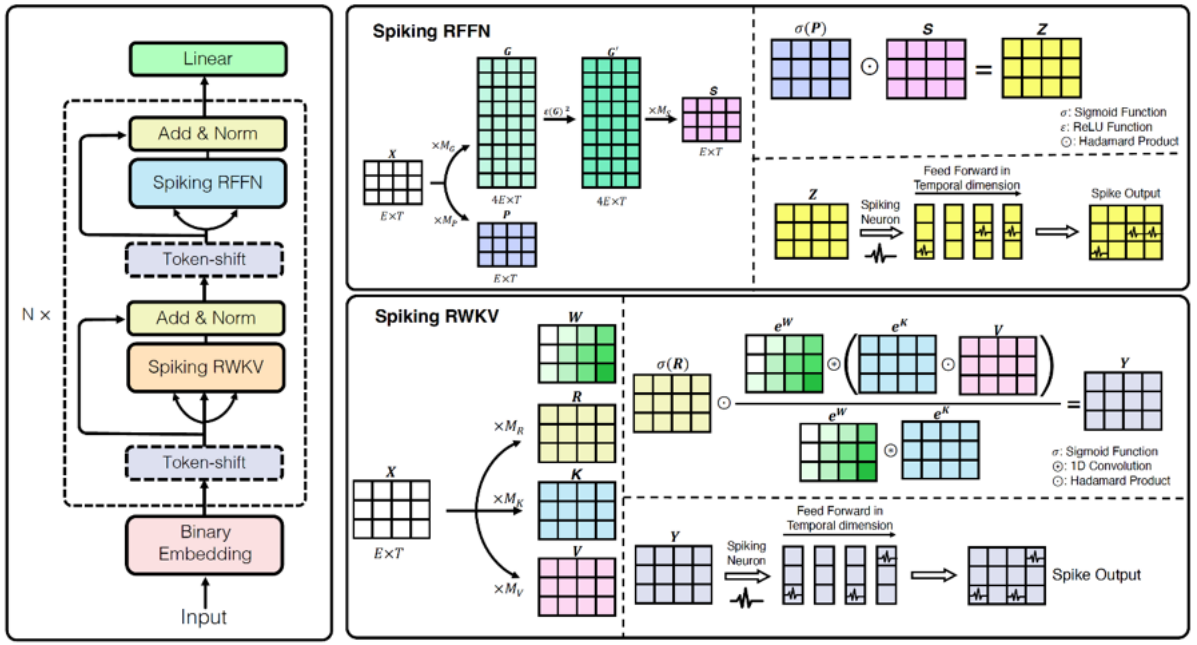
Performance samples

SpikeGPT: Generative Pre-trained Language Model with Spiking Neural Networks

Rui-Jie Zhu
Department of Electrical and Computer Engineering
University of California, Santa Cruz
ridger@live.cn

Qihang Zhao
Kuaishou Technology Co. Ltd
zhaoqihang@kuaishou.com

Jason K. Eshraghian*
Department of Electrical and Computer Engineering
University of California, Santa Cruz
jeshragh@ucsc.edu



Zhu, Zhao, Eshraghian, arXiv, 2023.

SpikeGPT

The largest SNN trained via backprop & the first to perform language generation

- 3 variants

Performanc

SpikeGPT: Generative Pre-trained Language Model with Spiking Neural Networks

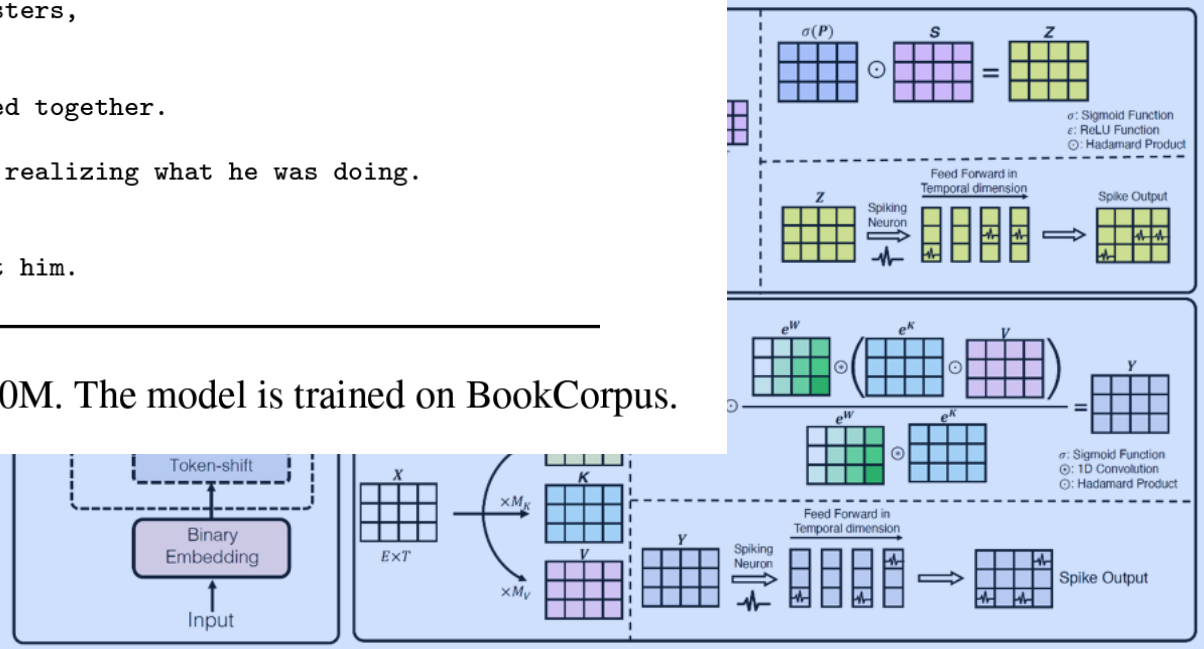
Rui-Jie Zhu
Department of Electrical and Computer Engineering

Qihang Zhao
Kuaishou Technology Co. Ltd
zhaoqihang@kuaishou.com

```
Context →                                                    <start>

---- Generated Answer 1 ----
”so what do you think of me?  ”
I asked.
he was standing behind me, still looking at the posters,
and I suddenly knew what he was asking.
“ what do you think of me?  ”
I looked up at him , the corners of his lips pressed together.
”i don’t know.”
he looked at me and i was staring at him, not even realizing what he was doing.
all I could do was watch him,
look at him, listen to his voice,
and then I would fall back to the stage and look at him.
```

Figure 3: Example of text generated by SpikeGPT 120M. The model is trained on BookCorpus.



Zhu, Zhao, Eshraghian, arXiv, 2023.

SpikeGPT

The largest SNN trained via backprop & the first to perform language generation

- 3 variants

Performanc

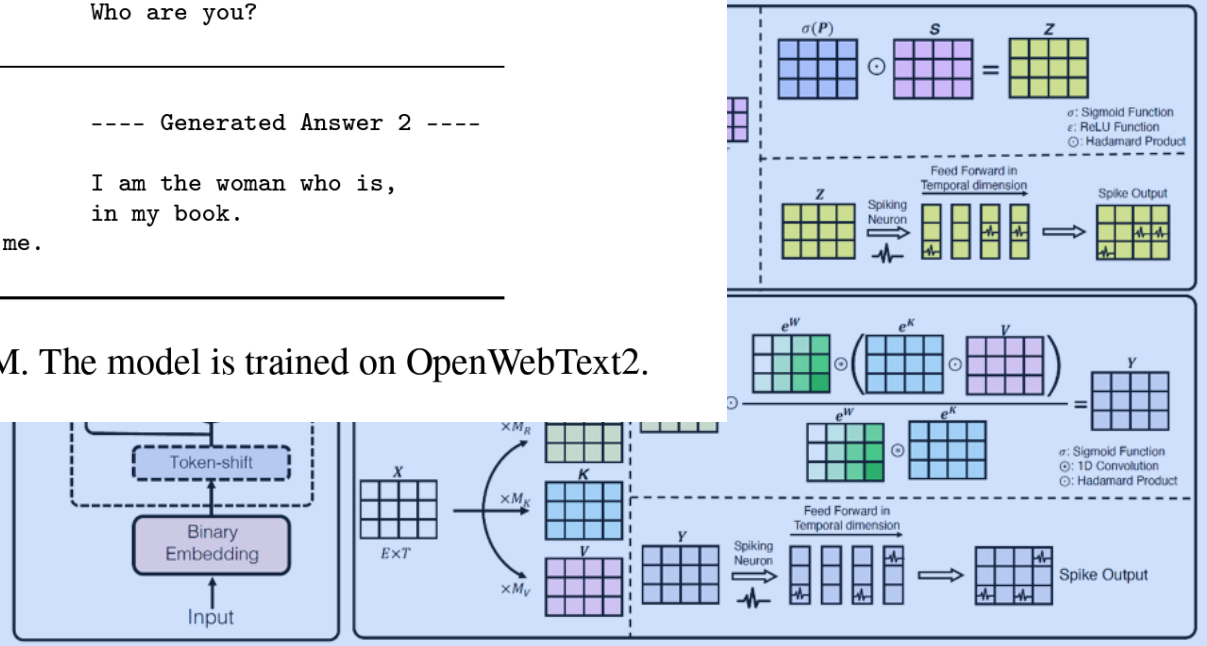
SpikeGPT: Generative Pre-trained Language Model with Spiking Neural Networks

Rui-Jie Zhu
Department of Electrical and Computer Engineering

Qihang Zhao
Kuaishou Technology Co. Ltd
zhaoqihang@kuaishou.com

Context →	Who are you? I am the AI. Who are you? I am Jack. Who are you? I am a woman. Who are you?
---- Generated Answer 1 ---- Amy. The amazing thing is, you can tell the stories about what happened to me.	---- Generated Answer 2 ---- I am the woman who is, in my book.

Figure 2: Example of text generated by SpikeGPT 260M. The model is trained on OpenWebText2.



Zhu, Zhao, Eshraghian, arXiv, 2023.

SpikeGPT

The largest SNN trained via backprop & the first to perform language generation

- 3 variants: 45M, 120M, 260M Parameter models

Performance samples

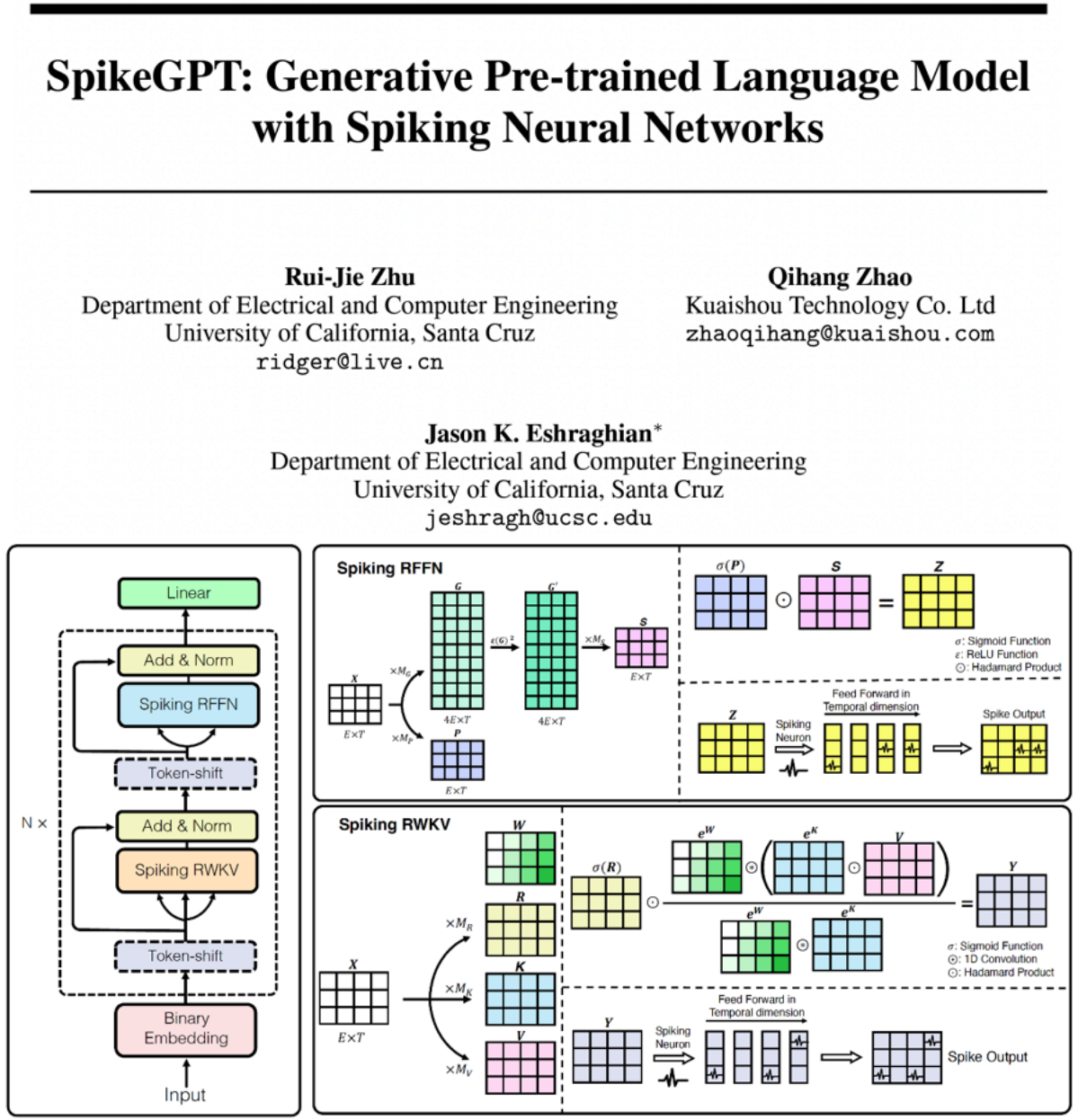
How did we do it?

The model is a dynamical system:

- The weights change over time
- Data is fed sequentially, much like how humans perceive the world

Modern language models are not dynamic. They require **all** input data to be available in parallel.

- 30x less operations than a transformer of equal size (N=12 blocks)



Zhu, Zhao, Eshraghian, arXiv, 2023.

Tutorial Outline

1. **Spiking Neurons**
2. **How to Train Your Spiking Neural Net**
   a. Local Learning
   b. Shadow Training
   c. Spike Time Learning
   d. Backprop Through Time and Surrogate Gradients
   e. Spike Time Learning
   f. Real-Time Recurrent Learning
3. **Low Precision Training**
4. **Perspectives: How can we do better?**

Tutorial Outline

1. **Spiking Neurons**
2. **How to Train Your Spiking Neural Net**
   a. Local Learning
   b. Shadow Training
   c. Spike Time Learning
   d. Backprop Through Time and Surrogate Gradients
   e. Spike Time Learning
   f. Real-Time Recurrent Learning
3. **Low Precision Training**
4. **Perspectives: How can we do better?**

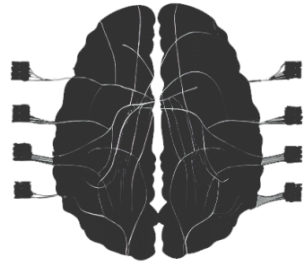
- Sparsity is more important than binarization
- Dynamics are critical
- Backprop is useful, but expensive. Can error signals be compressed?

Tutorial	Title	Colab Link
Tutorial 1	Spike Encoding with snnTorch	Open in Colab
Tutorial 2	The Leaky Integrate and Fire Neuron	Open in Colab
Tutorial 3	A Feedforward Spiking Neural Network	Open in Colab
Tutorial 4	2nd Order Spiking Neuron Models (Optional)	Open in Colab
Tutorial 5	Training Spiking Neural Networks with snnTorch	Open in Colab
Tutorial 6	Surrogate Gradient Descent in a Convolutional SNN	Open in Colab
Tutorial 7	Neuromorphic Datasets with Tonic + snnTorch	Open in Colab

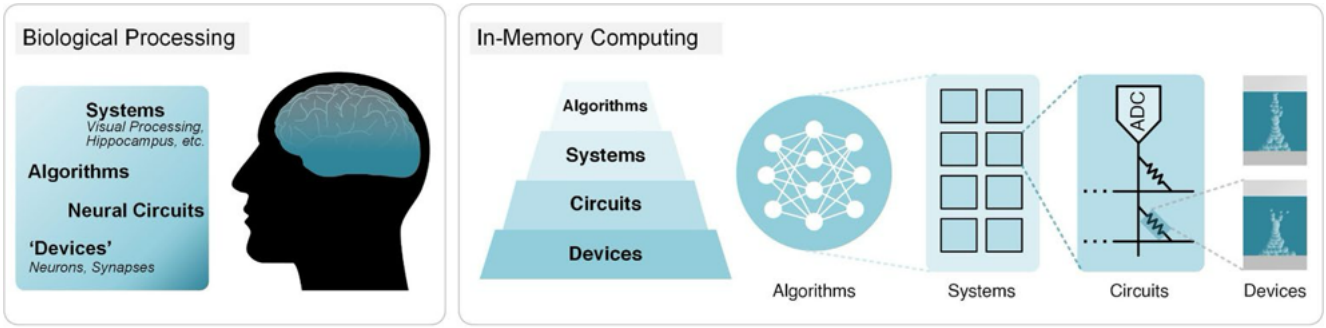
Advanced Tutorials	Colab Link
Population Coding	Open in Colab
Regression: Part I - Membrane Potential Learning with LIF Neurons	Open in Colab
Regression: Part II - Regression-based Classification with Recurrent LIF Neurons	Open in Colab
Accelerating snnTorch on IPUs	—

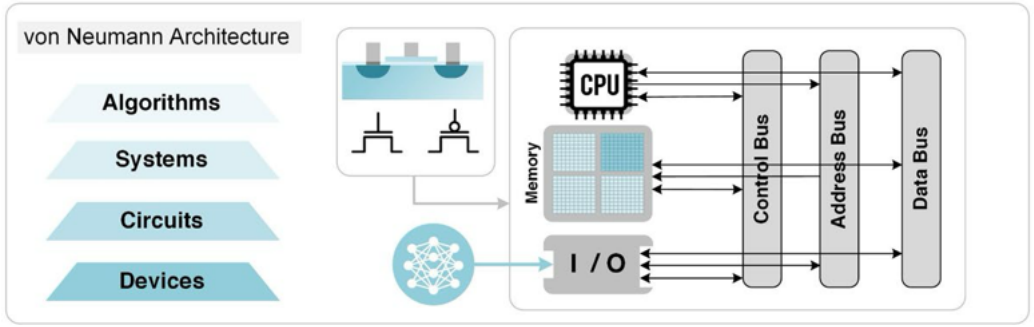
github.com/jeshraghian/snntorch

UCSC Neuromorphic Computing Group



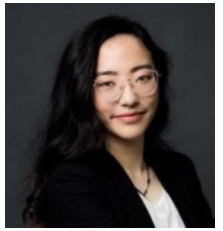
Lab Logo Generated by Stable Diffusion





Ph.D. Students

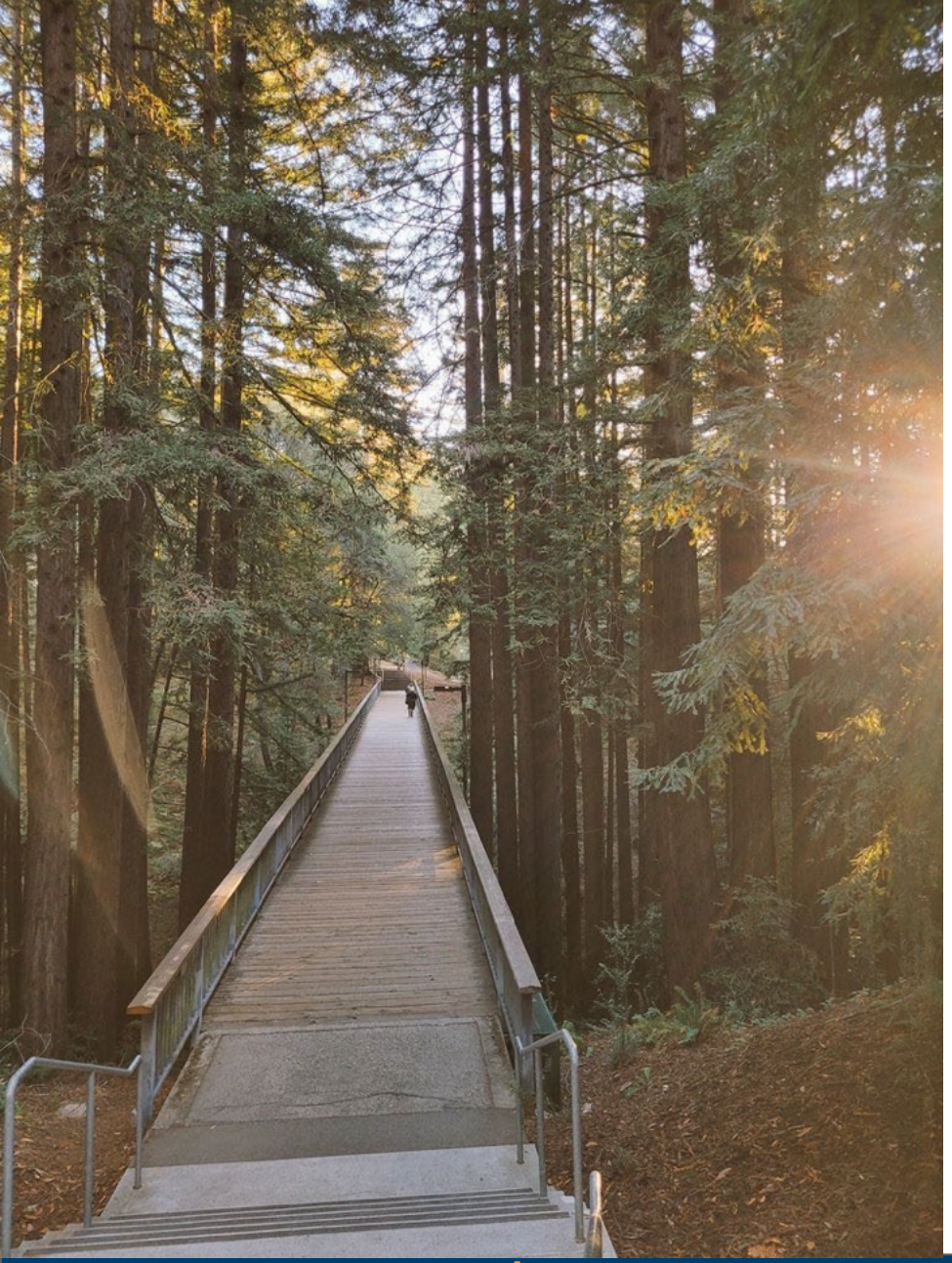
Binh Nguyen
Ruijie Zhu
Juan Lu
Coen Arrow
Assel Kembay



Undergraduate Students

Farhad Modaresi
Sreyes Venkatesh
Skye Gunasekaran
Hannah Cohen-Sandler

Ruhai Lin
Dylan Louie
Sahil Konjarla



Training Brain-Inspired Spiking Neural Networks Using Lessons from Deep Learning

Jason K. Eshraghian
Assistant Professor, ECE, UC Santa Cruz

18 March 2024