

NICE, 21 March 2026, Atlanta

Training SNNs with exact gradients: Progress and Challenges

Thomas Nowotny

School of Engineering and Informatics

University of Sussex

Event-based (Spiking) Neural Networks (SNNs) and gradient descent

Typical SNN: leaky integrate-and-fire (LIF) neurons and current-based (CUBA) synapses:

1. Dynamics:
$$\frac{dV}{dt} = -V + I$$

$$\frac{dI}{dt} = -I$$

2. Spike condition: $V_i - V_{\text{thresh}} = 0$

3. Jumps:
$$V_i^+ = V_{\text{reset}}$$
$$I_j^+ = I_j^- + w_{ji}$$

But also:

Adaptive LIF

Resonate and fire

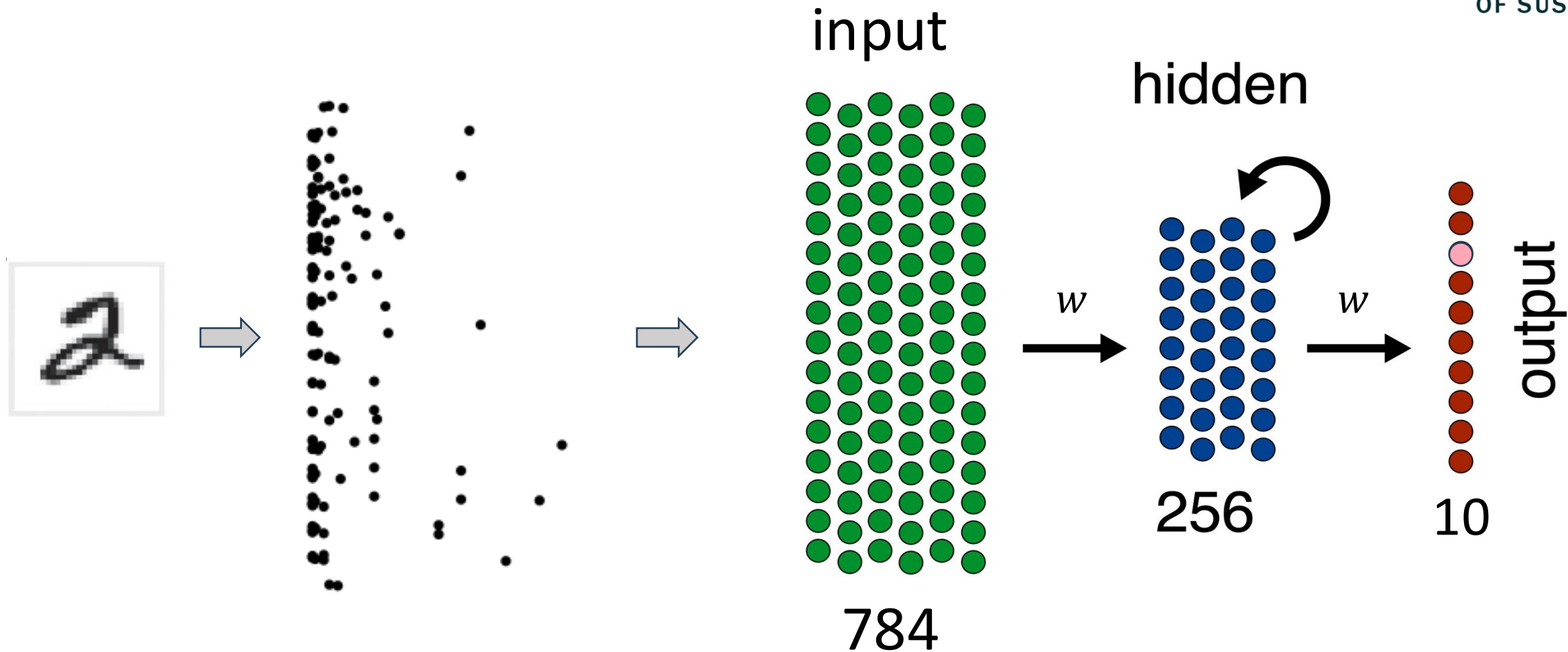
Quadratic LIF

Izhikevich

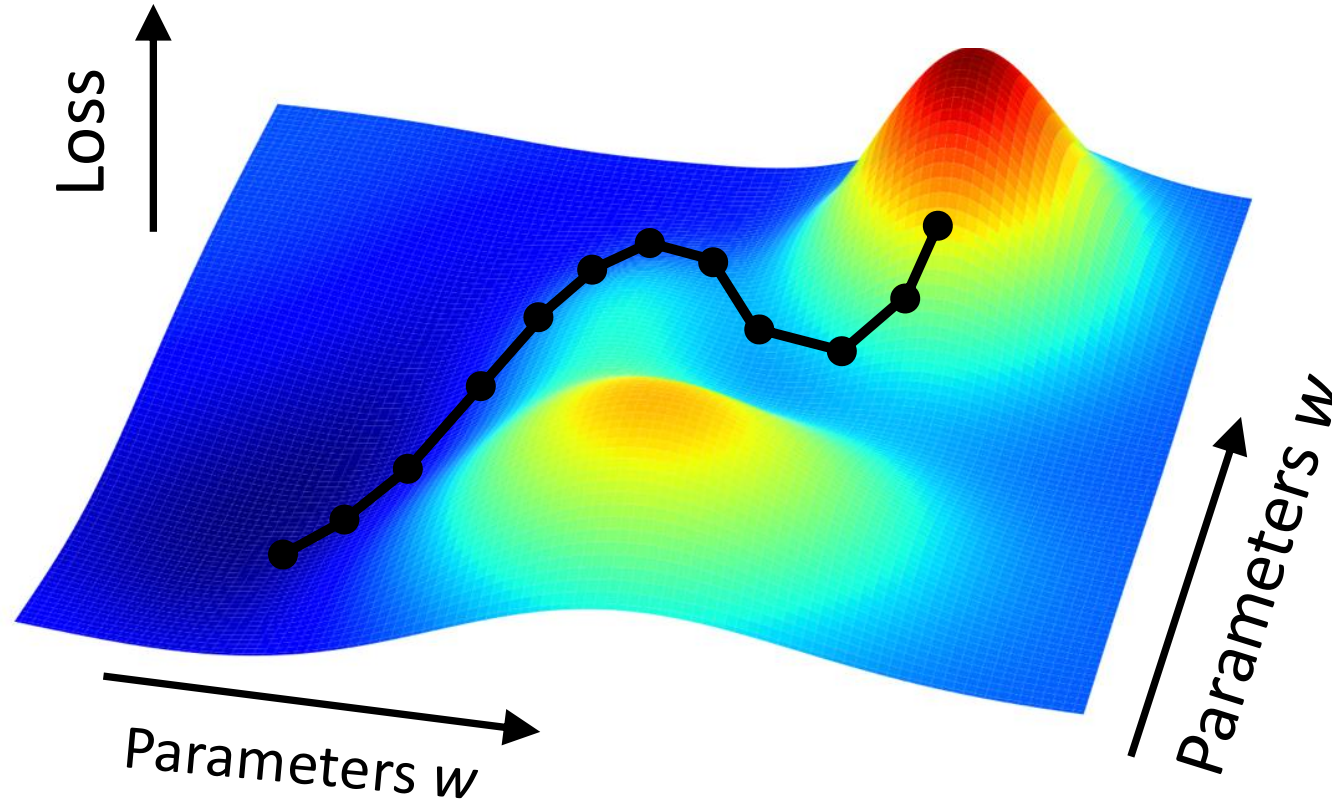
Hodgkin-Huxley

...

Classification (Regression similar)



Training: Gradient descent



Gradient:

$$\frac{d\mathcal{L}}{dw}$$

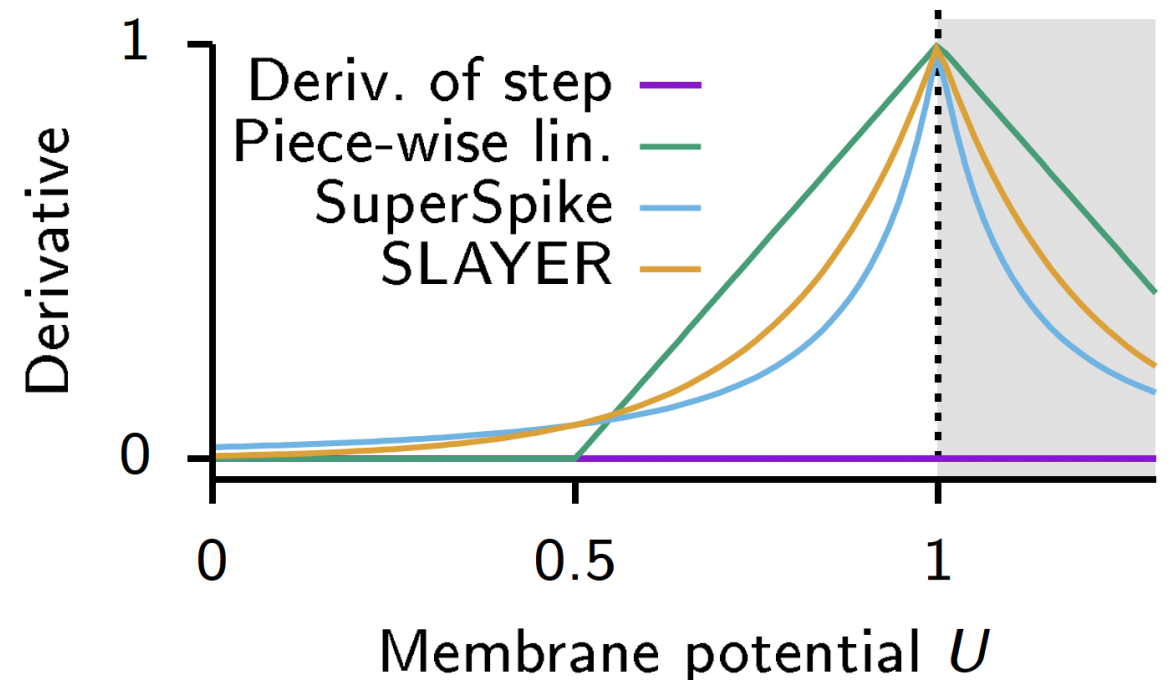
Weight update:

$$w^+ = w^- - \eta \frac{d\mathcal{L}}{dw}$$

Gradient descent in SNN

“Discretize then optimize” with surrogate gradients:

- Use a numerical integration method to turn SNN into a discrete time RNN
- Use back-propagation through time (BPTT) to calculate $\frac{d\mathcal{L}}{dw}$
- When encountering σ' for the “activation function” θ (Heaviside), use “surrogate gradients”



“Optimize then discretize”:

- The **Eventprop algorithm** allows calculating exact gradients for losses

$$\mathcal{L} = l_p(t^{\text{post}}) + \int_0^T l_V(V(t), t) dt$$

- It was derived by Wunderlich and Pehle from earlier work in optimal control theory (and physics) for LIF neurons and exponential CUBA synapses

Free dynamics	Transition condition	Jumps at transition
Forward: $\tau_{\text{mem}} \frac{d}{dt} V = -V + I$ $\tau_{\text{syn}} \frac{d}{dt} I = -I$	$(V)_n - \vartheta = 0,$ $(\dot{V})_n \neq 0$	$(V^+)_n = 0$ $I^+ = I^- + W e_n$
Backward: $\tau_{\text{mem}} \lambda'_V = -\lambda_V - \frac{\partial l_V}{\partial V}$ $\tau_{\text{syn}} \lambda'_I = -\lambda_I + \lambda_V$	$t - t_k = 0$	$(\lambda_V^-)_{n(k)} = (\lambda_V^+)_{n(k)} + \frac{1}{\tau_{\text{mem}} (\dot{V}^-)_{n(k)}} \left[\vartheta (\lambda_V^+)_{n(k)} + (W^T (\lambda_V^+ - \lambda_I^+))_{n(k)} + \frac{\partial l_p}{\partial t_k} + l_V^- - l_V^+ \right]$
	Gradient	$\frac{\partial \mathcal{L}}{\partial w_{ji}} = -\tau_{\text{syn}} \sum_{t_{\text{spike}}(i)} \lambda_{I,i}(t_{\text{spike}}(i))$

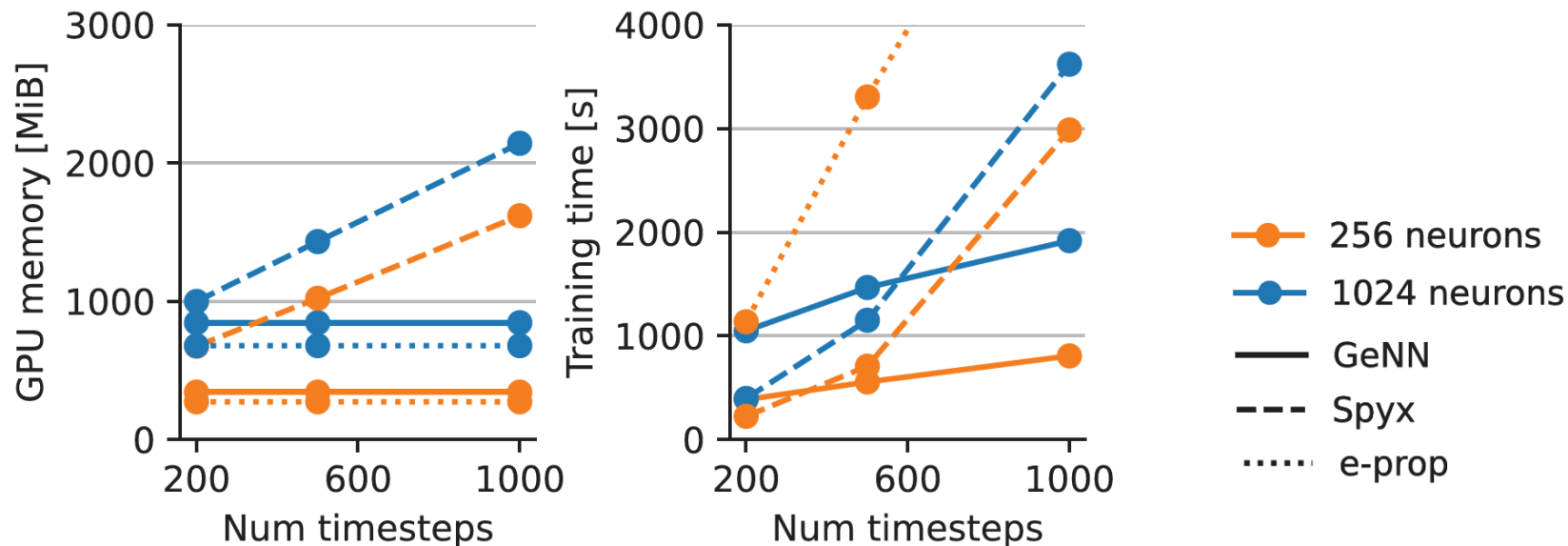
Wunderlich TC, Pehle C. Event-based backpropagation can compute exact gradients for spiking neural networks. Sci Rep. 2021, 11, 12829

Optimize then discretize

- Exact gradient
- Storage $\mathcal{O}(N_{spike})$
- Compute $\mathcal{O}(NT) + \mathcal{O}(N \cdot N_{spike})$
- mlGeNN and Eventprop

Discretize then optimize

- “Surrogate” gradient
- Storage $\mathcal{O}(NT)$
- Compute $\mathcal{O}(N^2T)$
- PyTorch with **AutoDiff**



Spiking Heidelberg Digits (SHD)



- 10420 recordings
- 12 speakers
- Digits 0-9 in English and German
- Converted to 700 spike trains using inner ear model

Cramer, B., Stradmann, Y., Schemmel, J., & Zenke, F. (2020). The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*.

<https://doi.org/10.1109/TNNLS.2020.3044364>

Results

Using mlGeNN (https://github.com/genn-team/ml_genn):

- With extended loss functions $\mathcal{L} = \mathcal{F}\left(\int l_v dt\right)$ we can classify the Spiking Heidelberg Digits (SHD) and Spiking Speech Commands (SSC) to high accuracy [1]
- We can calculate gradients wrt neuron parameters (e.g. membrane time constant) [1]
- We can train networks with delays & calculate gradients wrt delays [2]
- We can deploy trained networks on Loihi with minimal accuracy loss [3], and also with delays [4]

[1] Nowotny, Turner, Knight (2025). *Neurom. Comput. & Eng.* 5(1), 014001.

[2] Mészáros, Knight, Nowotny (2025) *arXiv preprint arXiv:2501.07331*.

[3] Shoesmith, Knight, Mészáros, Timcheck, Nowotny (2025) NICE 2025, Heidelberg

[4] Mészáros, Knight, Timcheck, Nowotny (2025), *submitted to ICONS*

Beyond LIF

- One foundation of deep learning is auto-diff in PyTorch
- If we want to use the adjoint method for other models, this involves long mathematical derivations
- Can we have an auto-adjoint method in mlGeNN?

Spiking Neural Networks (SNNs) and gradient descent

Eventprop == LIF & exp CUBA:

1. Dynamics: $\frac{dV}{dt} = -V + I$

$$\frac{dI}{dt} = -I$$

2. Spike condition: $V_n - V_{\text{thresh}} = 0$

3. Jumps: $V_n^+ = V_{\text{reset}}$

$$I_m^+ = I_m^- + w_{mn}$$

General adjoint method for SNN:

1. Dynamics: $\frac{dx}{dt} = F(x, p, t)$

2. Spike condition: $g(x_n, p) = 0$

3. Jumps: $x_n^+ = f(x_n^-, p)$

$$x_m^+ = x_m^- + h(x_n^-, p_{mn})$$

General adjoint method for SNN

Adjoint dynamics
relates to dynamics F

$$\lambda_r^i = \frac{\partial F_s^i}{\partial x_r^i} \lambda_s^i - \frac{\partial l}{\partial x_r^i}$$

loss injection
Jumps gated by derivative
of spike condition g

$$\lambda_r^{n-} = \lambda_s^{n+} \frac{\partial f_s}{\partial x_r^{n-}} + \lambda_s^{m+} \frac{\partial h_s^{mn}}{\partial x_r^{n-}} + \frac{\frac{\partial g}{\partial x_r^{n-}}}{\frac{\partial g}{\partial x_s^{n-}} \dot{x}_s^{n-}} \left[\frac{\partial l_p}{\partial t_k} + l_k^- - l_k^+ - \lambda_s^{n+} \left(\frac{\partial f_s}{\partial x_q^{n-}} \dot{x}_q^{n-} - \dot{x}_s^{n+} \right) - \lambda_s^{m+} \left(\dot{x}_s^{m-} - \dot{x}_s^{m+} + \frac{\partial h_s^{mn}}{\partial x_q^{n-}} \dot{x}_q^{n-} \right) \right].$$

Transport of loss caused
by forward jump

$$\lambda_r^{m+} = \lambda_r^{m-}$$

$$\frac{\partial \mathcal{L}}{\partial w_{ji}} = - \sum_{\text{spikes in } i} \lambda_s^{j+} \frac{\partial h_s^{ji}}{\partial w_{ji}}$$

Event-based update
of gradient

Results

- We implemented the method in mlGeNN using sympy
- Users can provide a forward model like so:

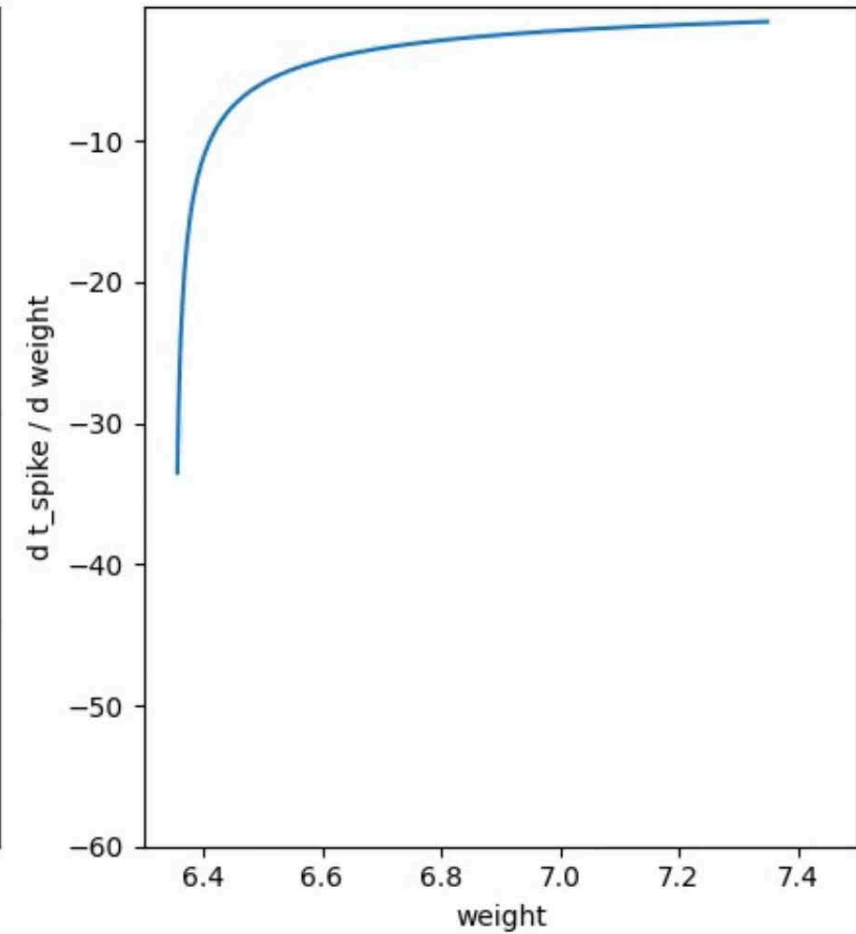
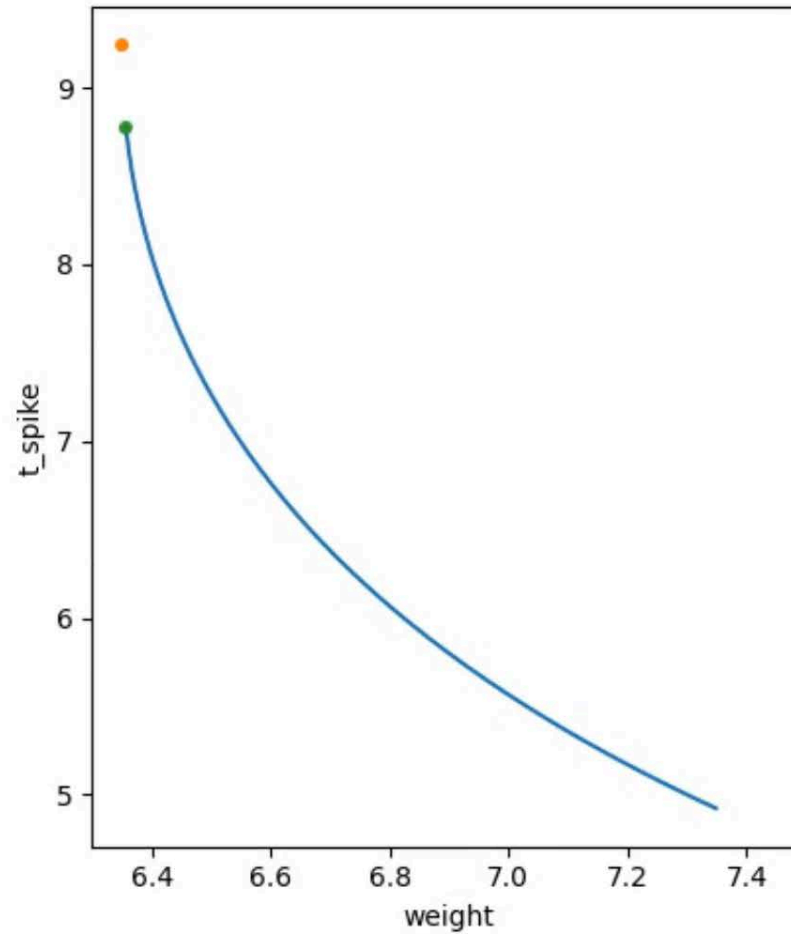
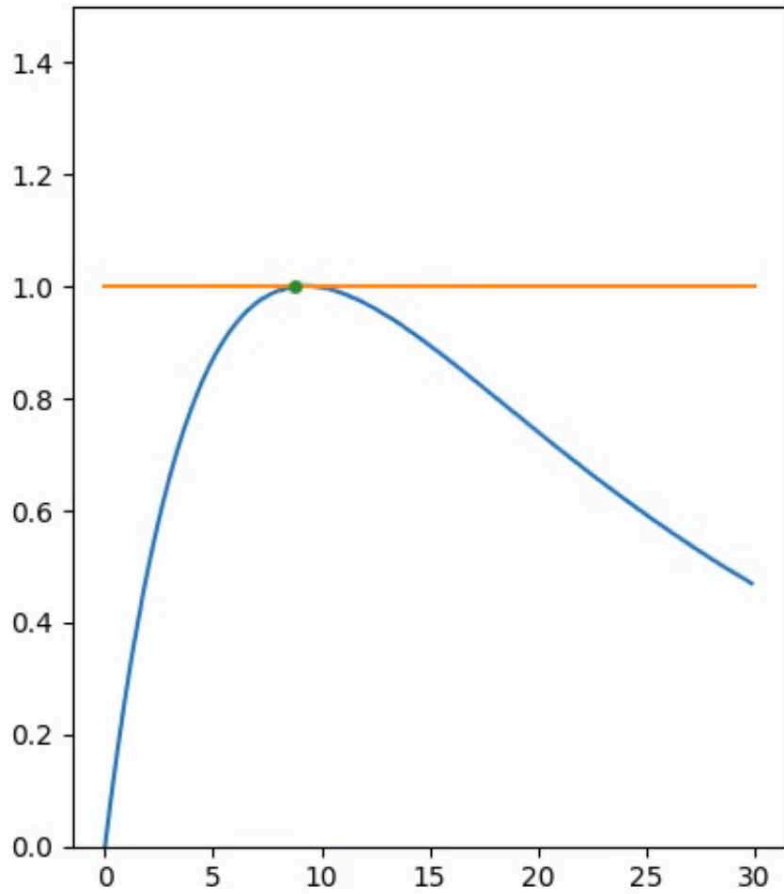
```
lif_neuron = UserNeuron(vars={"v": ("(-v + Isyn) / tau_mem", "0.0")},  
                        threshold="v - 1.0",  
                        output_var_name="v",  
                        param_vals={"tau_mem": 20.0},  
                        var_vals={"v": 0.0})
```

```
exp_synapse = UserSynapse(vars={"i": ("-i / tau", "i + weight")},  
                          inject_current="i",  
                          param_vals={"tau": 5.0},  
                          var_vals={"i": 0.0})
```

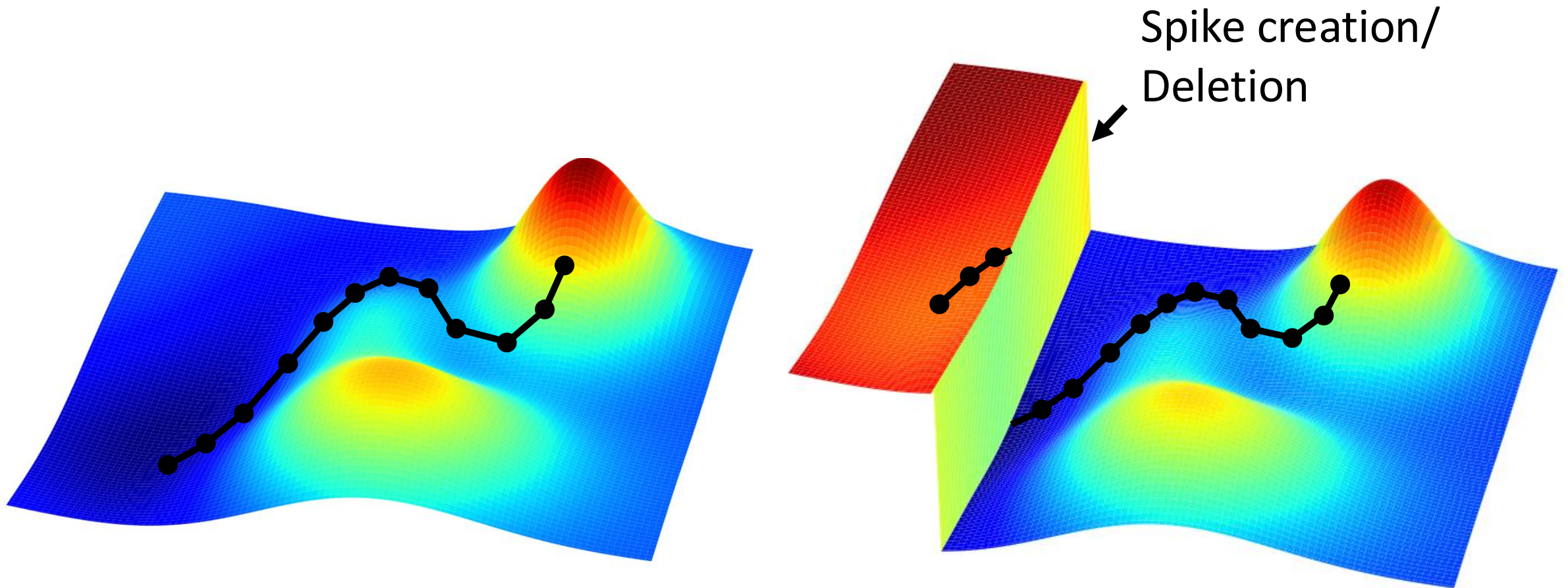
Challenges

- Disruptive spike deletion and creation:

LIF with CUBA synapse



Disruptive spikes creation/deletion



Challenges

- Disruptive spike deletion and creation
- Silent neurons

Once a neuron ceases firing, it is dead permanently:

$\lambda_r^{m-} = \lambda_r^{m+}$: non-spiking neurons have no λ jumps

$\lambda_r^j = \frac{\partial F_s^j}{\partial x_r^j} \lambda_s^j - \frac{\partial l}{\partial x_r^j}$: The adjoint variables start at $\lambda_s^j(T) = 0$, so all

λ_r^j stay 0 unless there is direct loss injection

$\frac{\partial \mathcal{L}}{\partial w_{ji}} = - \sum \lambda_s^{j+} \frac{\partial h_s^{ji}}{\partial w_{ji}}$: No gradient for w_{ji} if $\lambda_s^j = 0$

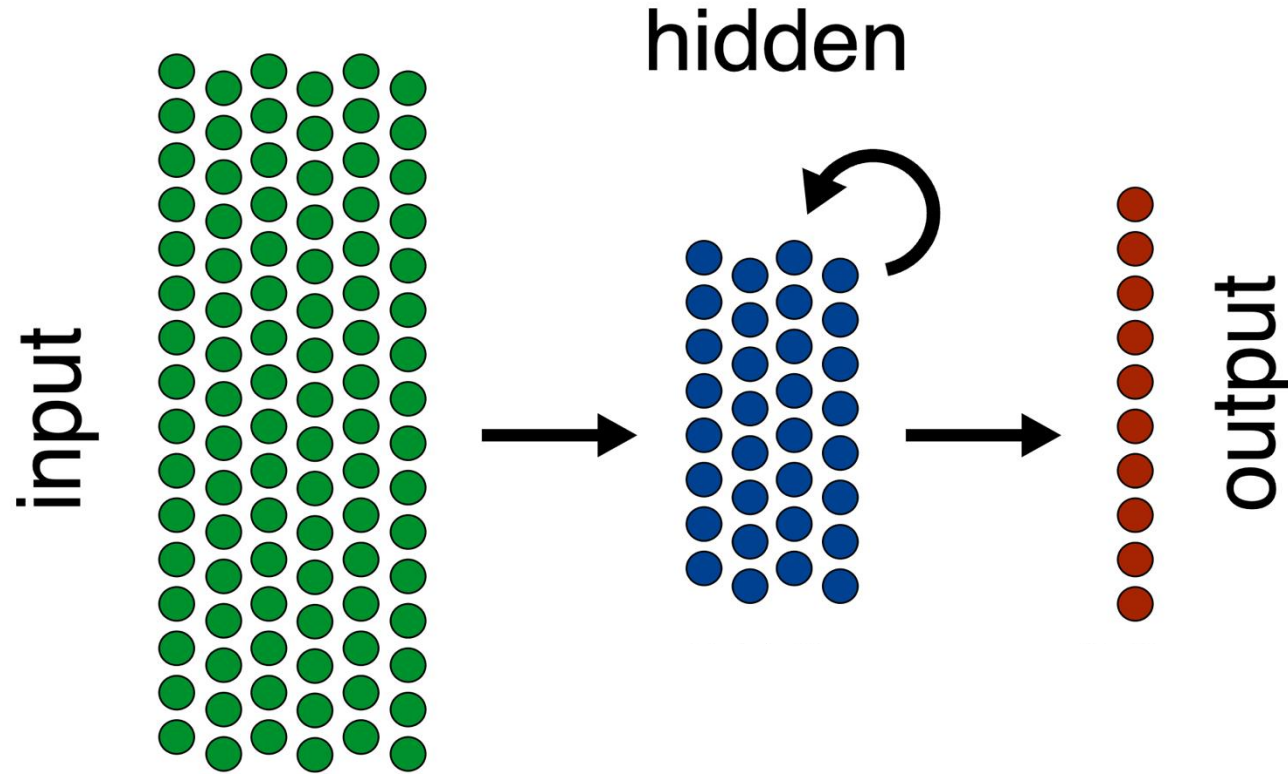
Challenges

- Disruptive spike deletion and creation
- Silent neurons
- Regularisation

Challenges

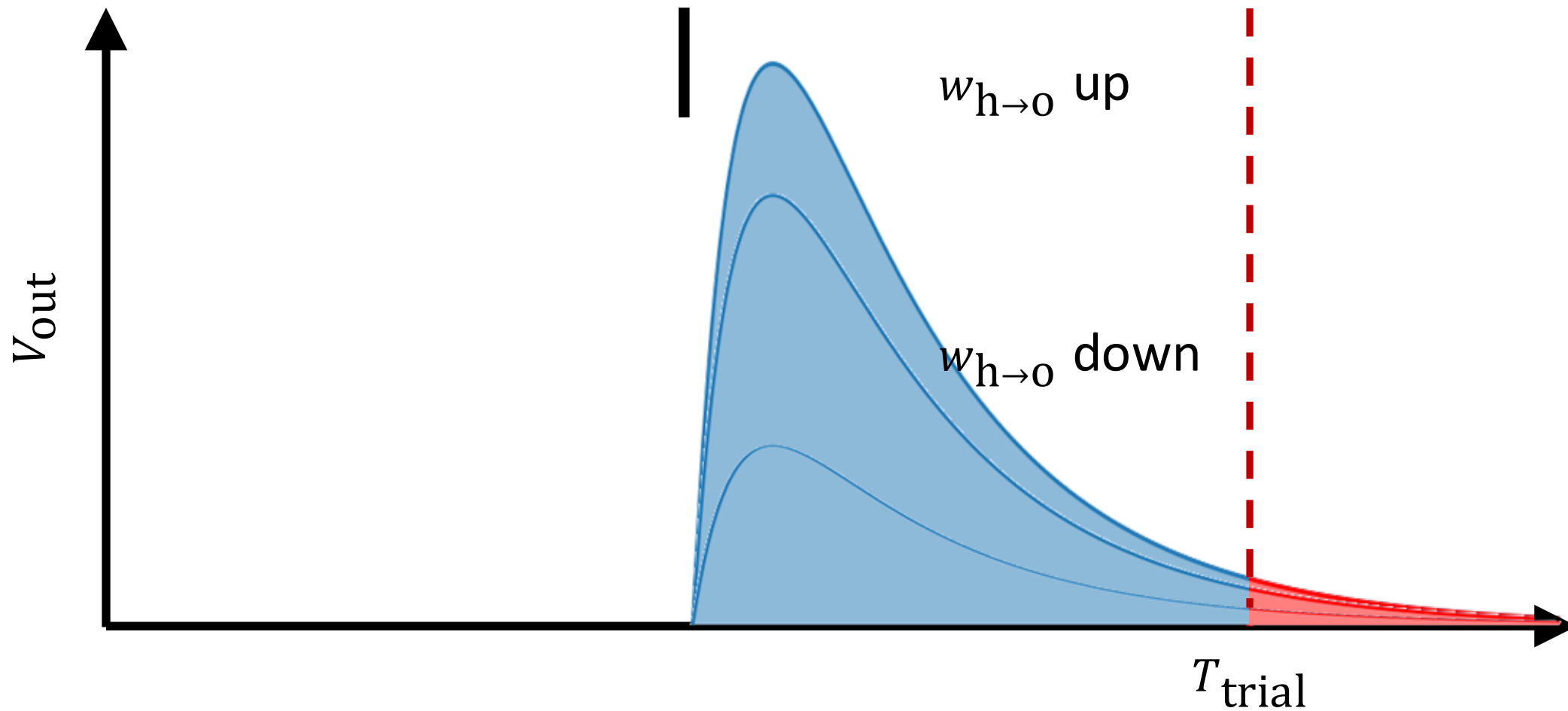
- Disruptive spike deletion and creation
- Silent neurons
- Regularisation
- Gradient flow

Gradient flow

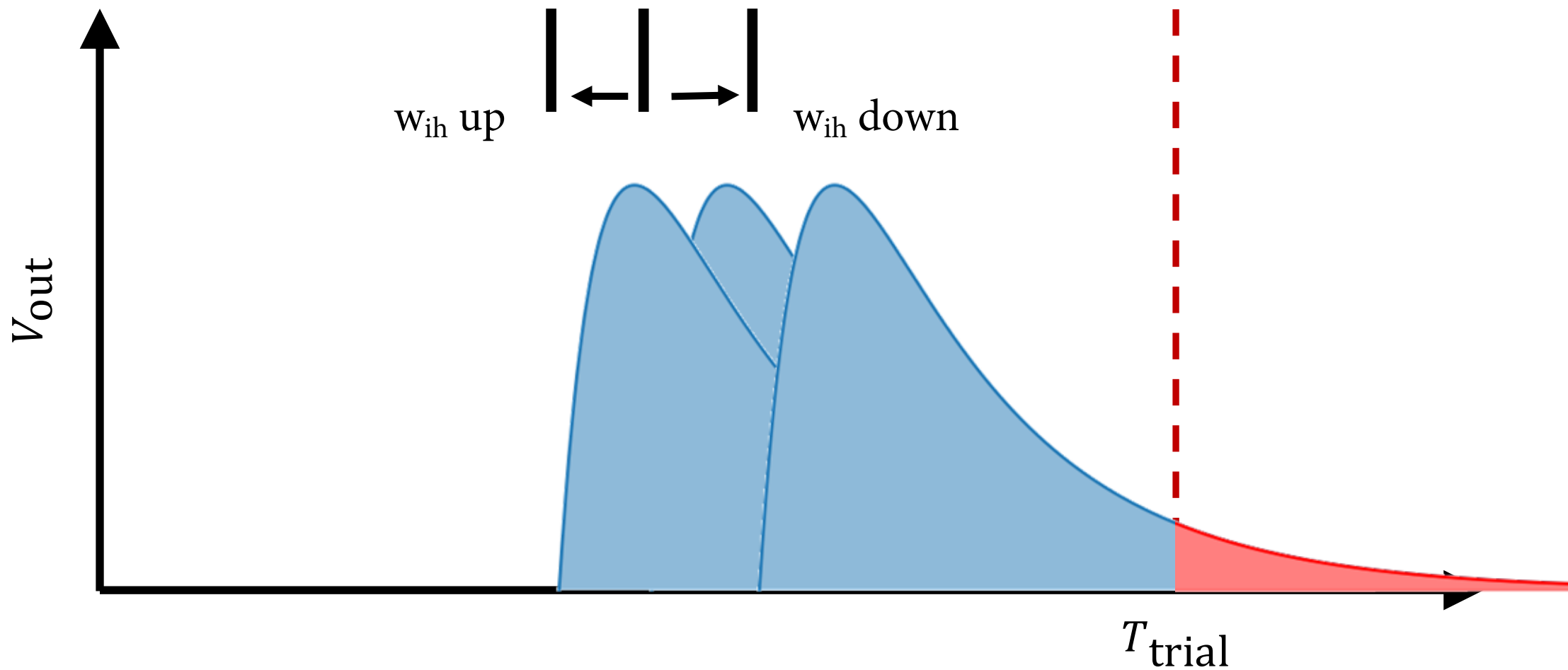


Gradient flow

Sussex AI



Gradient flow



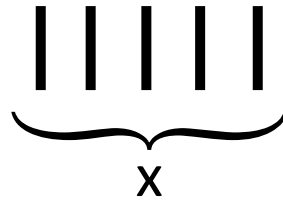
Challenges

- Disruptive spike deletion and creation
- Silent neurons
- Regularisation
- Gradient flow
- **Encoding and readout**

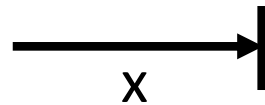
Encoding and readout

Encoding

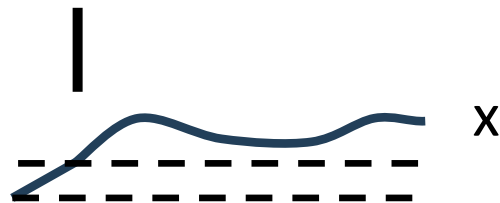
- Rate



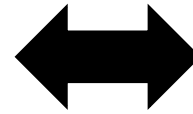
- Time to first spike



- $\sigma - \delta$

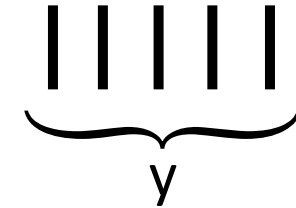


- Input current



Readout

- Rate



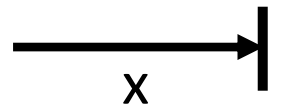
- V_{out}



- $\int V_{out} dt$



- Time to first spike



Acknowledgements



Co-authors:

- Jamie Knight
- Balázs Mészáros
- Jonathan Timcheck
- Tom Shoesmith



The Leverhulme Trust



GeNN developers:

- Esin Yavuz
- James Turner
- Anton Komissarov



mlGeNN: https://github.com/genn-team/ml_genn