Slides and code available at this link, feel free to open on your laptop
https://github.com/neural-reckoning/cambridge-fens-chen-summer-school-2026

Learning with spikes

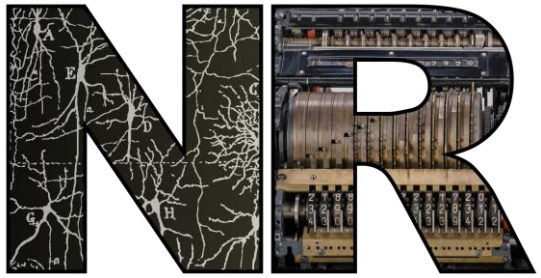
FENS-Chen Institute summer school lecture

Dan Goodman

Imperial College London

Department of Electrical and Electronic Engineering

IMPERIAL



neural-reckoning.org

Plan for today

1. Why we should care about spiking neural networks (SNNs)
2. Short introduction to SNNs and how to train them
3. Live demo of training an SNN using Colab
4. While you try out the Colab code yourself, some of our SNN research
5. The computational role of neural heterogeneity
6. The computational role of neuromodulation
7. Finally, Q&A and practical session, answering your questions while you try using the Colab notebook
8. Finish with a competition: who managed to train their network to the best level of accuracy during the lecture?

Why spikes?

Neurons in the brain spike, so why am I even asking this question?

Trend is for non-spiking artificial neural network models

Assumption is that only spike rates matter

Still a controversial topic but: evidence doesn't support this

- Most of the energy used by the brain is about producing or recovering from spikes
- Information theoretic analyses show spike times at least as important as rates
- Many examples where spike timing is shown to be functionally important

My intuition is that **the temporal dynamics of neurons** are very important

ANN models miss this out

But my reason today:

I can add biophysical mechanisms to spiking models and see how they influence computations. This lets me discover **the possible computational role of a mechanism**.

Simplest model of spiking neuron: integrate-and-fire

Membrane potential v

Incoming synapses with weights w_i

On presynaptic spike index i:

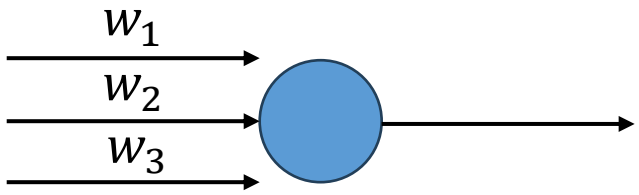
$v \leftarrow v + w_i$

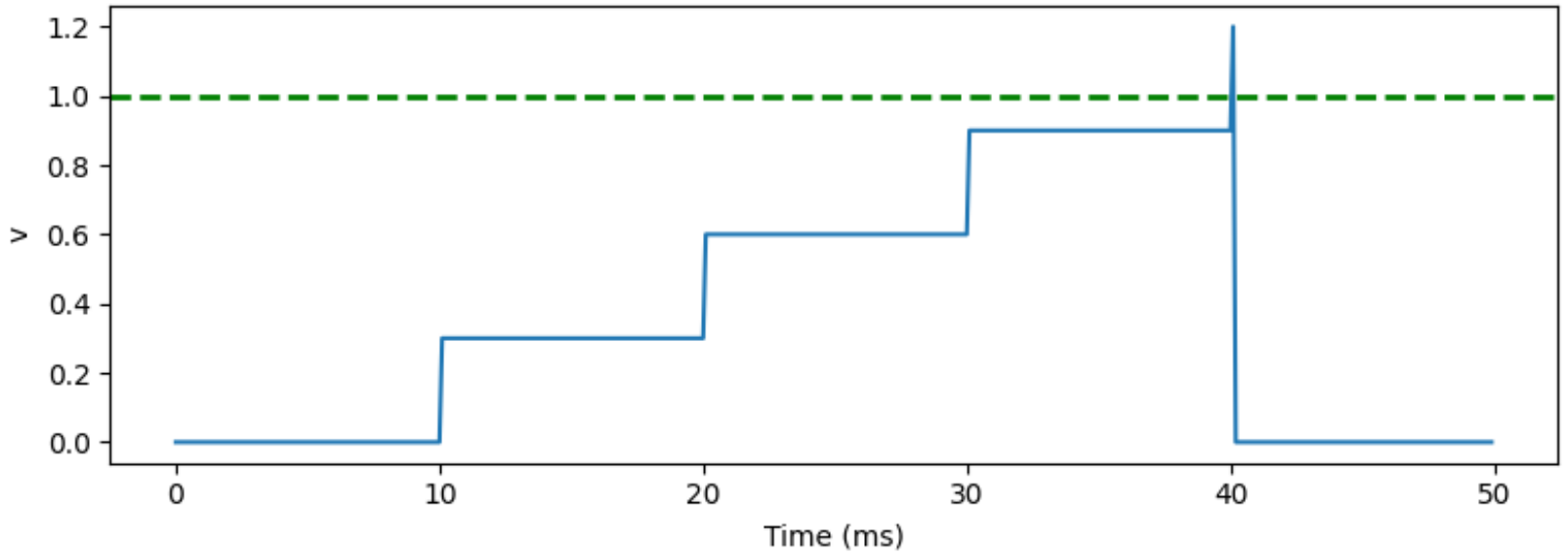
Threshold condition, fire a spike if:

$v > 1$

Reset event, after a spike set:

$v \leftarrow 0$

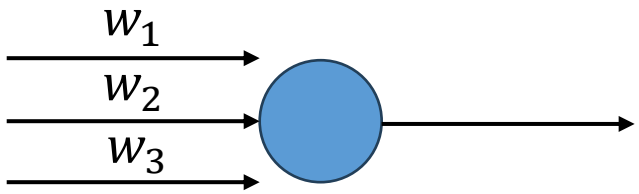




Leaky integrate and fire (LIF) neuron

Continuous evolution over time:

$$\tau \frac{\mathrm{d}v}{\mathrm{d}t} = -v$$ Solution is $v(t) = v(0)\exp(-t/\tau)$



On presynaptic spike index i:

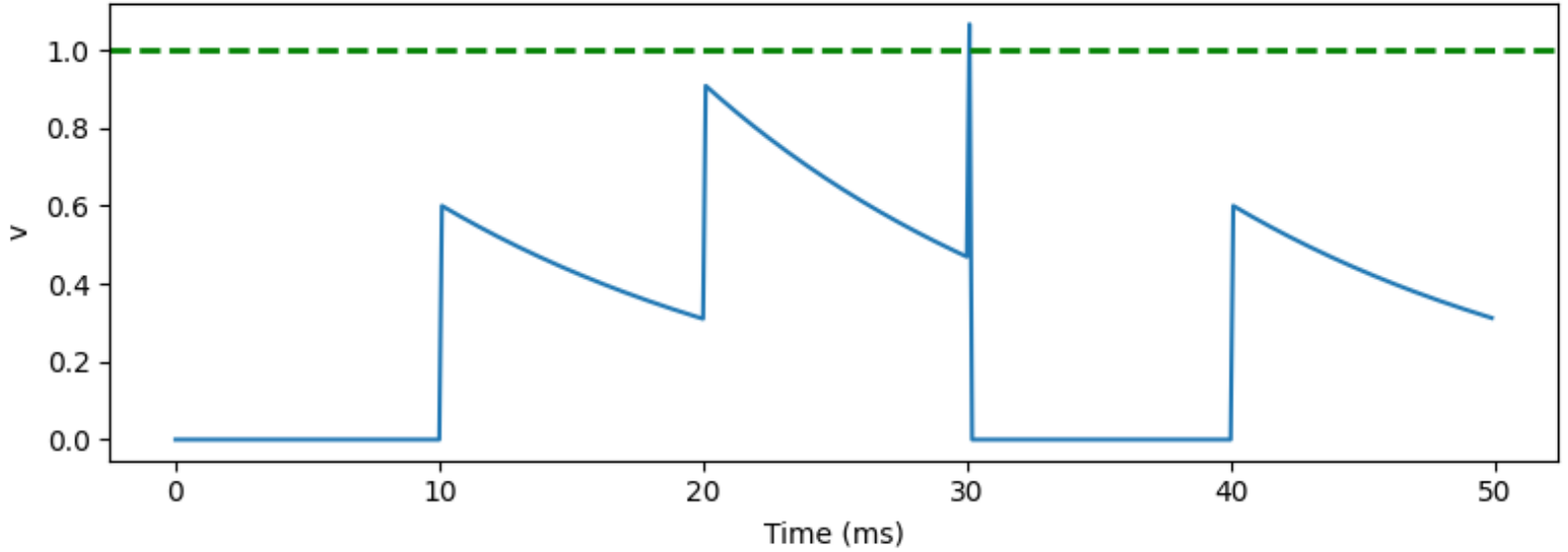
$$v \leftarrow v + w_i$$

Threshold condition, fire a spike if:

$$v > 1$$

Reset event, after a spike set:

$$v \leftarrow 0$$



Training SNNs: What we want to do

Gradient descent for artificial neural networks

Write neural network as $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$

- Parameters $\boldsymbol{\theta}$
- Input $\boldsymbol{x}$ leading to output $\boldsymbol{y}$

Want to minimise loss:

- Loss function $\mathcal{L}(\boldsymbol{y}, \boldsymbol{y}^*)$
- Desired output $\boldsymbol{y}^*$
- $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{x}, \boldsymbol{\theta}), \boldsymbol{y}^*)$

Gradient descent:

- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{x}, \boldsymbol{\theta}), \boldsymbol{y}^*)$
- Learning rate γ
- Chain rule to compute ∇ term

For example:

- $\boldsymbol{\theta} = \boldsymbol{w}$ weight matrix
- $f(\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$
- σ = some nonlinear function (e.g. sigmoid)

Computing the gradient:

- Gradient $(\nabla \mathcal{L})_i = \frac{\partial \mathcal{L}}{\partial \theta_i}$
- Vector chain rule:
  - $x \in \mathbb{R}^n \xrightarrow{f} y \in \mathbb{R}^m \xrightarrow{g} z \in \mathbb{R}^p$
  - Jacobian matrix $J^f_{ij} = \frac{\partial f_i}{\partial x_j}$
  - Chain rule is $J^{g \circ f} = J^g J^f$
- Efficient: just matrix/ vector multiplications

So, let's just do this with f being a spiking neural network?

Why we can't do it

Rewrite LIF equations:

(Temporarily ignore synaptic connections.)

Threshold

$$S(t) = H(v(t) - 1)$$

Update solution at time $t + \delta t$ in terms of values at time t:

$$v(t + \delta t) = e^{-\delta t/\tau} v(t)(1 - S(t))$$

To compute $\nabla \mathcal{L}$ we use chain rule

Everything differentiable except H

$$\frac{\partial H}{\partial x} = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$$

Gradients are zero almost everywhere

Gradient descent doesn't work for SNNs

Recap:

Leaky integrate-and-fire neuron

Continuous evolution over time:

$$\tau \frac{\mathrm{d}v}{\mathrm{d}t} = -v$$

On presynaptic spike index i:

$$v \leftarrow v + w_i$$

Threshold condition, fire a spike if:

$$v > 1$$

Reset event, after a spike set:

$$v \leftarrow 0$$

Heaviside function $H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$

Spiking neurons: the surrogate gradient

Threshold

$$S(t) = H(v(t) - 1)$$

Update solution at time $t + \delta t$ in terms of values at time t:

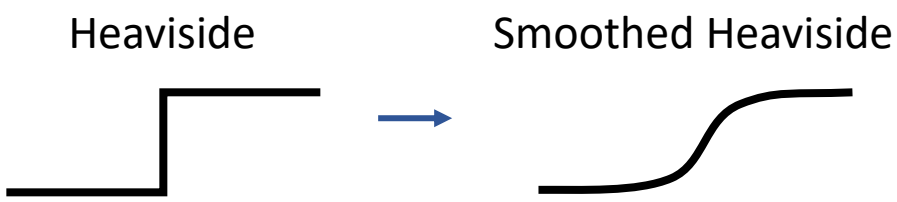
$$v(t + \delta t) = e^{-t/\tau} v(t)(1 - S(t))$$

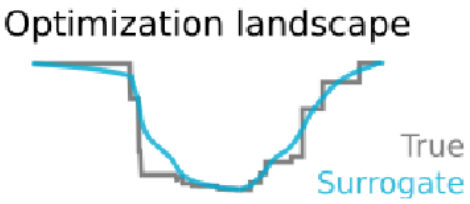
Problem:

$$\frac{\mathrm{d}H}{\mathrm{d}x} = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$$

Solution:

Replace $\frac{\mathrm{d}H}{\mathrm{d}x}$ with $\frac{\mathrm{d}}{\mathrm{d}x} H^{\text{smooth}}$





e.g. use logistic function $H^{\text{smooth}}(x) = \sigma(x) = \frac{1}{1+e^{-\beta x}}$ to get

$$\frac{\mathrm{d}}{\mathrm{d}x} H^{smooth} = \beta\sigma(x)(1 - \sigma(x))$$

Recap:

Leaky integrate-and-fire neuron

Continuous evolution over time:

$$\tau \frac{\mathrm{d}v}{\mathrm{d}t} = -v$$

On presynaptic spike index i:

$$v \leftarrow v + w_i$$

Threshold condition, fire a spike if:

$$v > 1$$

Reset event, after a spike set:

$$v \leftarrow 0$$

Heaviside function $H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$

Robust to choice of function!

Zenke and Vogels (2021)

Implementation with PyTorch

Fortunately, PyTorch (and other libraries) make this easy:

```
class SurrogateHeaviside(torch.autograd.Function):
    @staticmethod
    def forward(ctx, input):
        ctx.save_for_backward(input)
        return torch.heaviside(input, 0)
    @staticmethod
    def backward(ctx, grad_output):
        input, = ctx.saved_tensors
        beta = 5
        s = torch.sigmoid(beta*input)
        grad = grad_output*beta*s*(1-s)
        return grad

surrogate_heaviside  = SurrogateHeaviside.apply
```

Derive from PyTorch to make the magic happen

Forward pass: just return Heaviside function

Backward pass:
Return gradient of sigmoid

Now just use this instead of Heaviside function

Demo time!

Try it out yourself in your browser with Colab

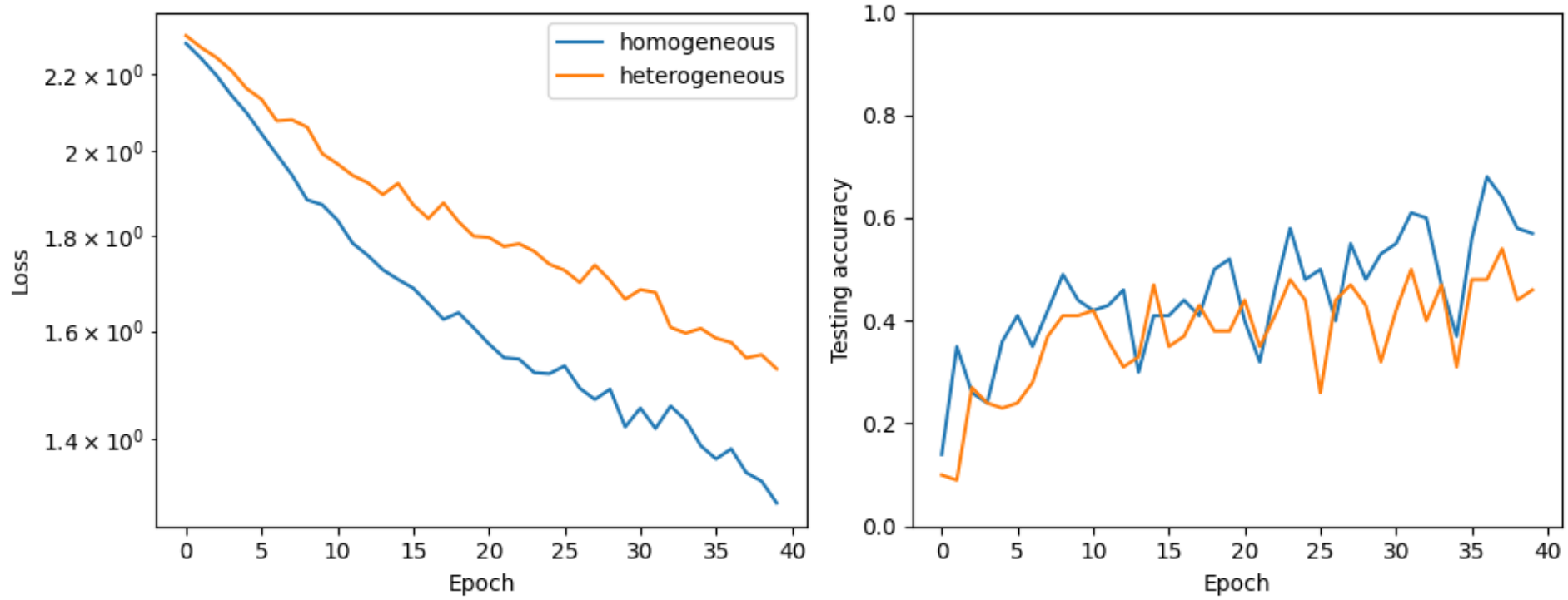
https://github.com/neural-reckoning/cambridge-fens-chen-summer-school-2026

And here's what I got with a longer training run

40 epochs with 40 batches of size 32 per epoch (51,200 simulations), learning rate 0.001

Took about an hour when I ran it on Colab, still hadn't converged!

Was expecting the heterogeneous network to do better, but it did worse – ah well



Some of our research

Feel free to keep tinkering with the code during this next bit

You can always read the papers later 😉 TLDR:

Heterogeneity

- For tasks with rich temporal dynamics, making time constants heterogeneous improves performance
- Code change is very simple: write
  `tau = nn.Parameter(torch.empty(n))` instead of
  `tau = torch.empty(n)`

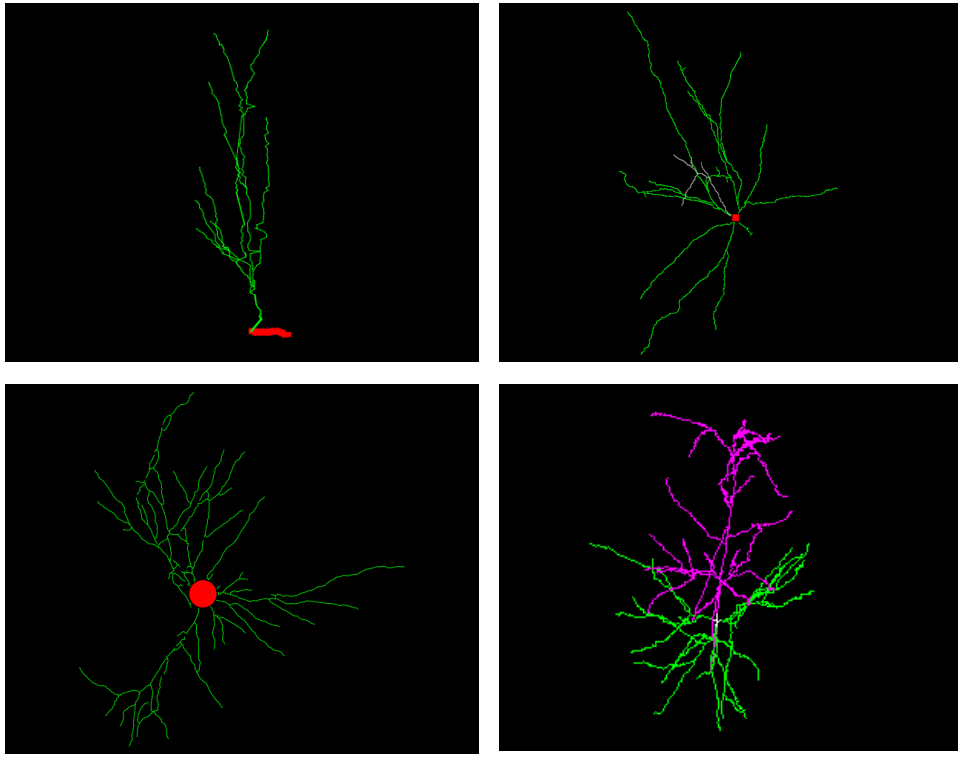
Neuromodulation

- Takes this one step further, let the network control its own parameters.
- This is as simple as letting a spike change any parameter instead of just a membrane potential (but there's more you can do too)

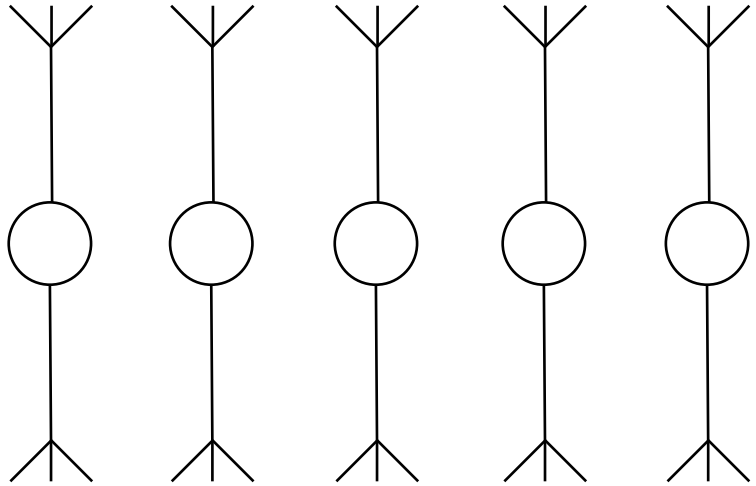
Why is the brain so heterogeneous?

What the brain is like

Some random neurons from neuromorpho.org



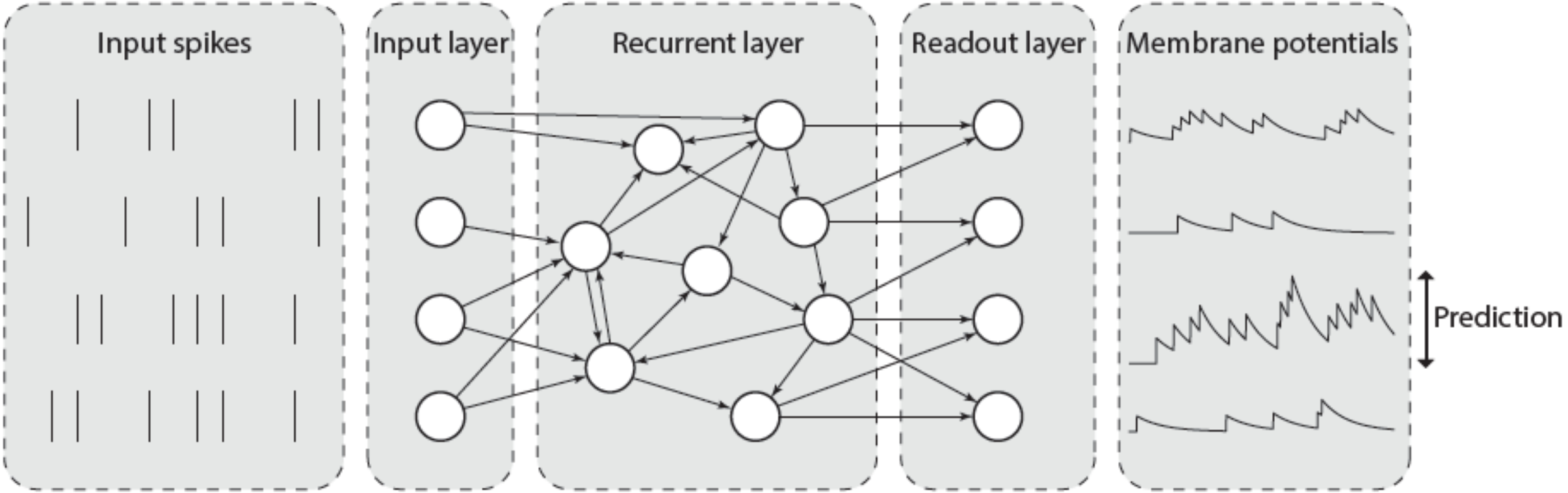
What our models are like



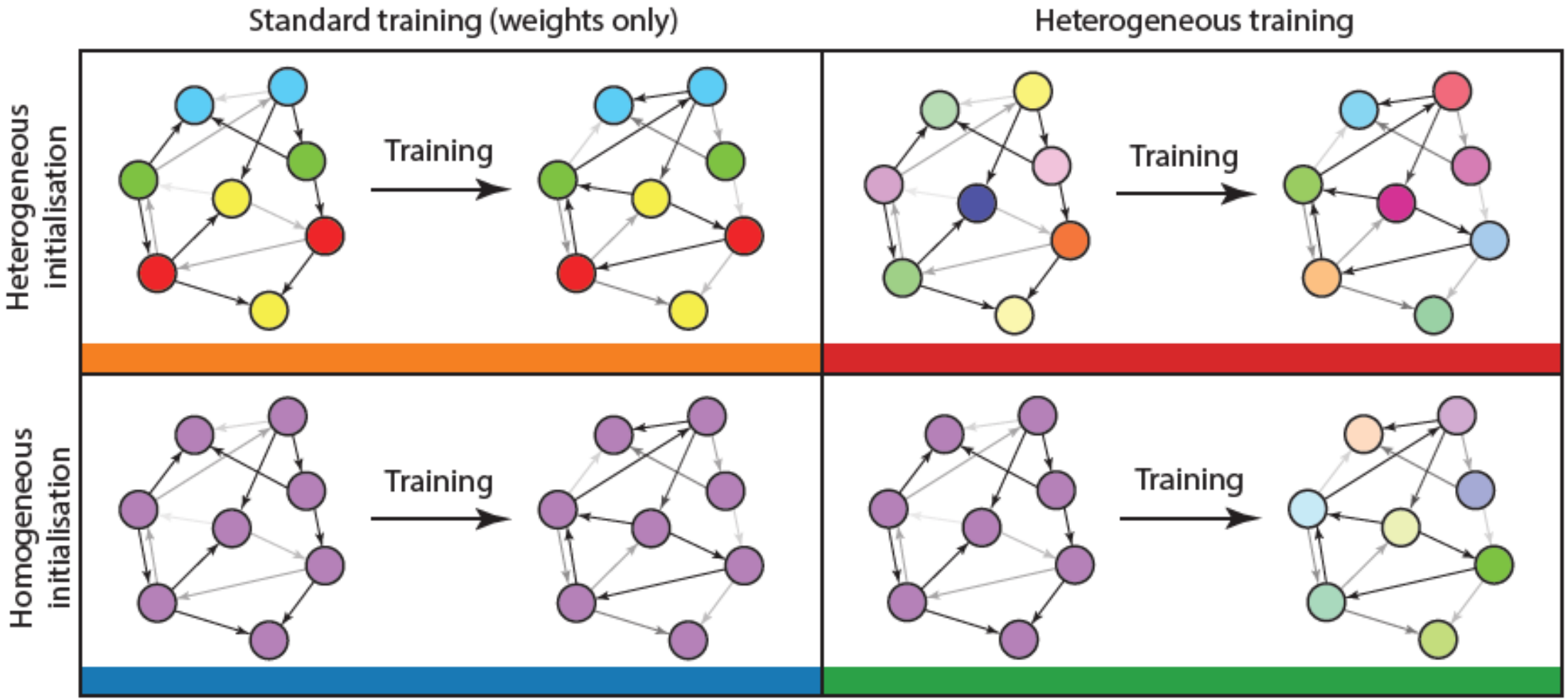
This study

Replace single value of τ with different for each neuron
Let the network learn τ as well as weights
Compare performance on different tasks

Methods: model architecture



Methods: training



Methods: tasks

Visual

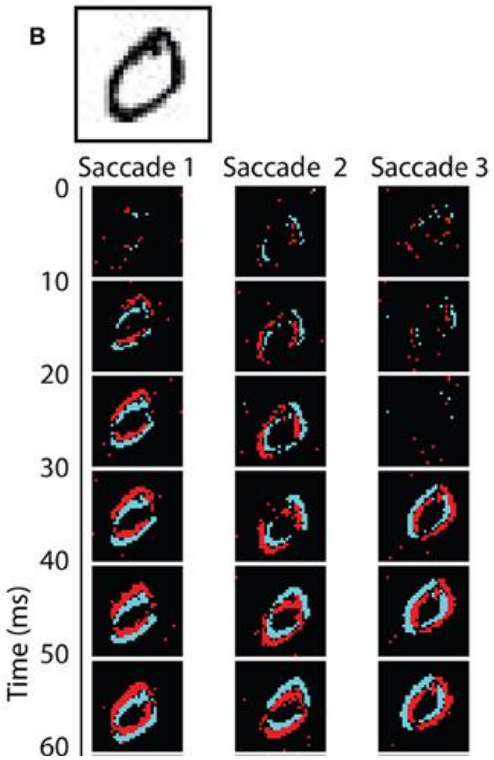
Auditory

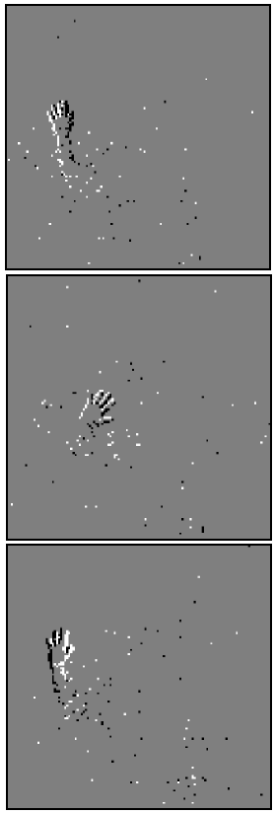
N-MNIST

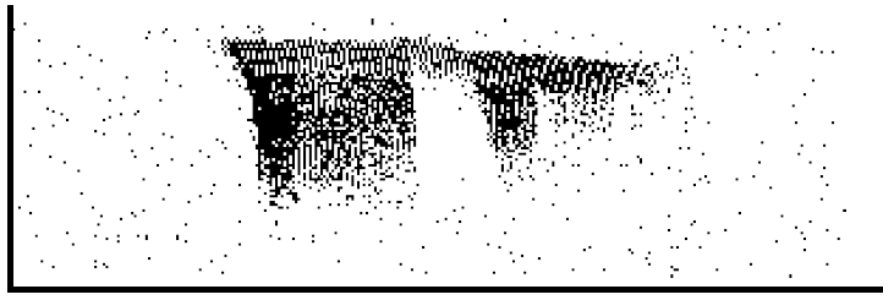
F-MNIST

DVS128

SHD and SSC



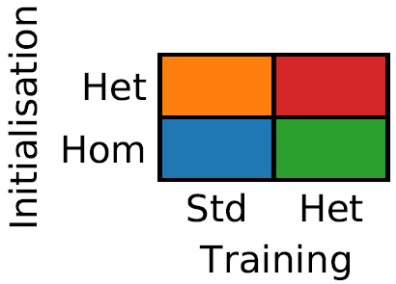


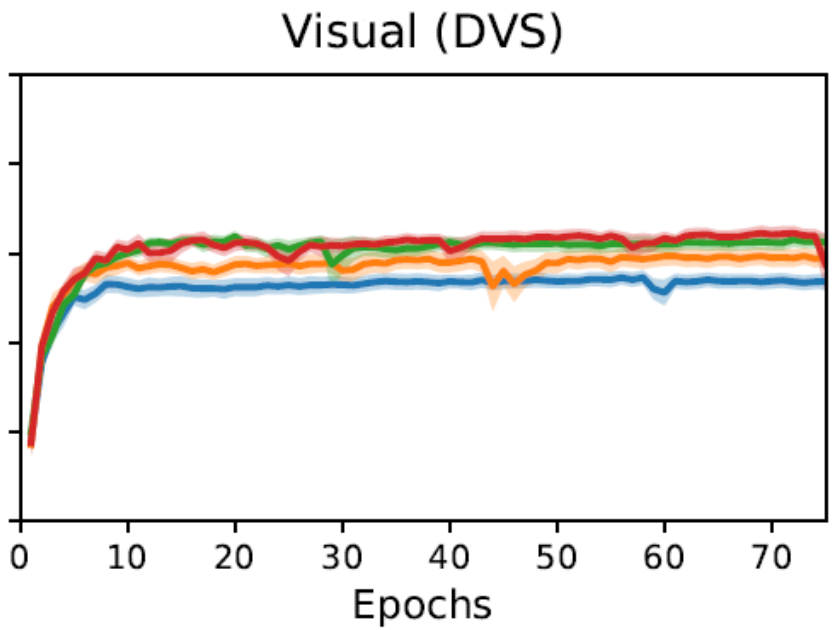


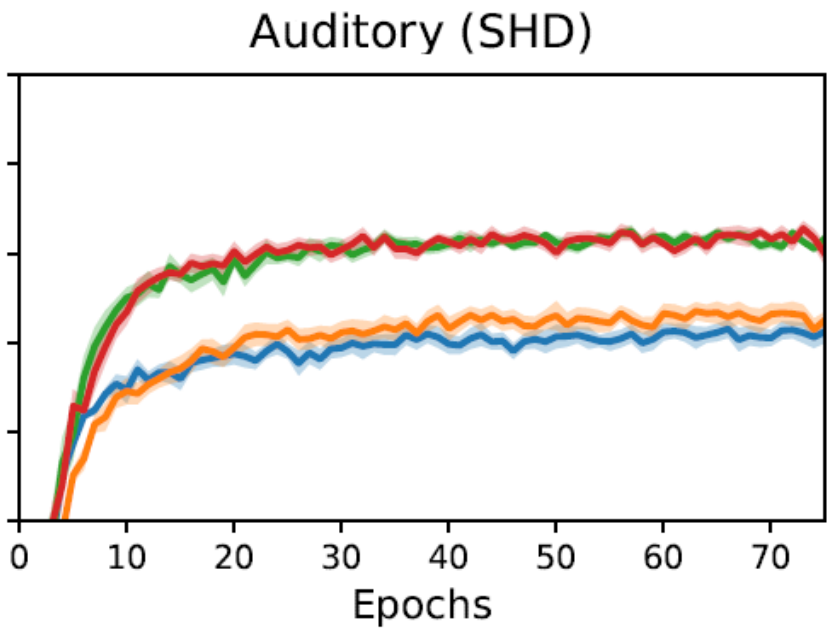
Increasing temporal structure

Results: performance

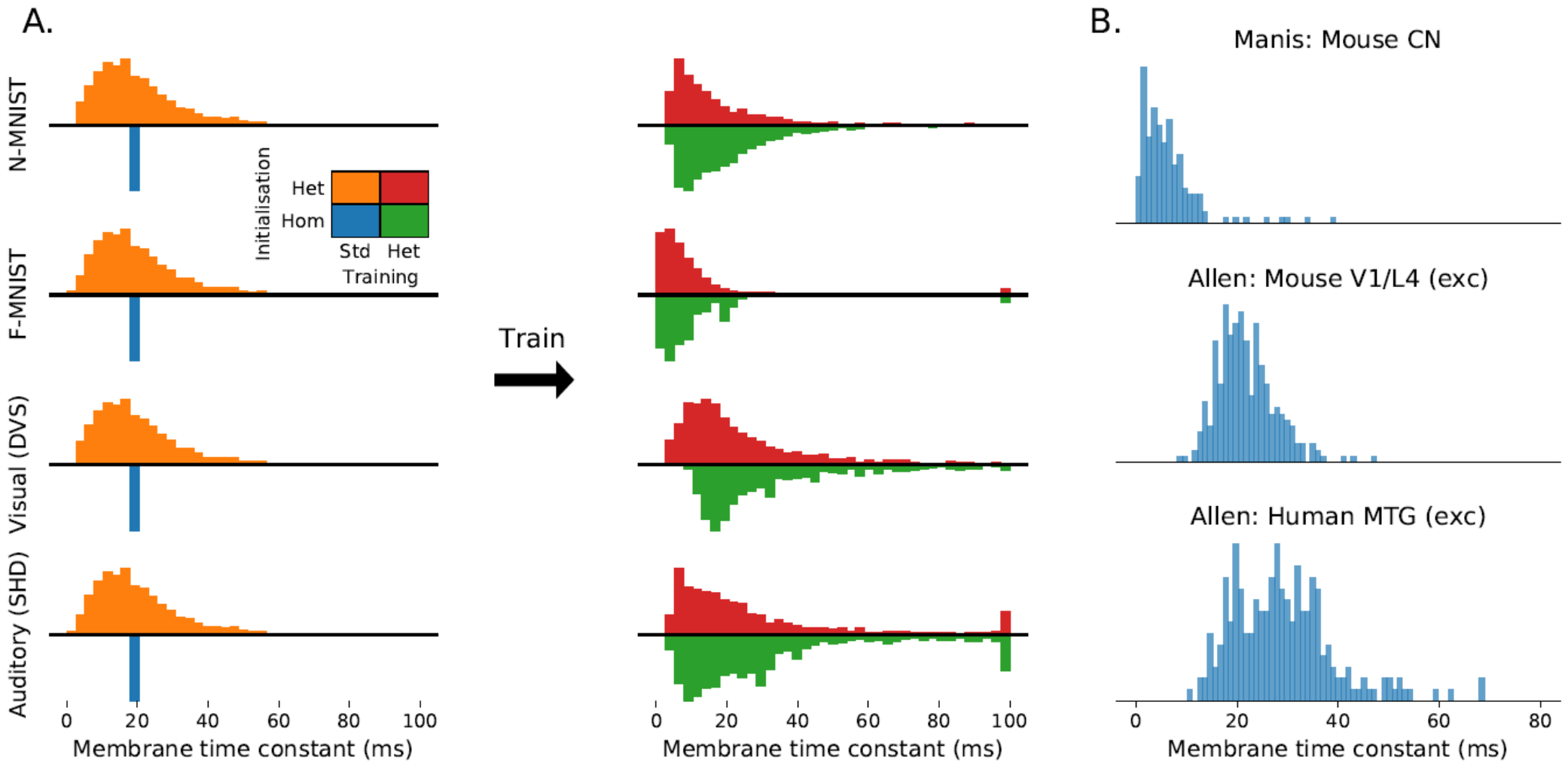
Initialisation	Training	N-MNIST	F-MNIST	DVS128	SHD	SSC
Homog.	Standard	97.4±0.0	80.1±7.4	76.9±0.8	71.7±1.0	49.7±0.4
Heterog.	Standard	97.5±0.0	87.9±0.1	79.5±1.0	73.7±1.1	53.7±0.7
Homog	Heterog.	96.6±0.2	79.7±7.4	81.2±0.8	82.7±0.8	56.6±0.7
Heterog.	Heterog.	97.3±0.1	87.5±0.1	82.1±0.8	81.7±0.8	60.1±0.7
	Chance level	10.0	10.0	10.0	5.0	2.9







Results: distribution of time constants



Discussion / conclusions

Heterogeneity is metabolically / energetically efficient

- Big performance gains with no increase in energy costs

Open questions

- Does the brain optimise for this?
- Do other forms of heterogeneity matter as much? (See next paper)
- Can we find a rigorous theoretical explanation? (Still working on this!)

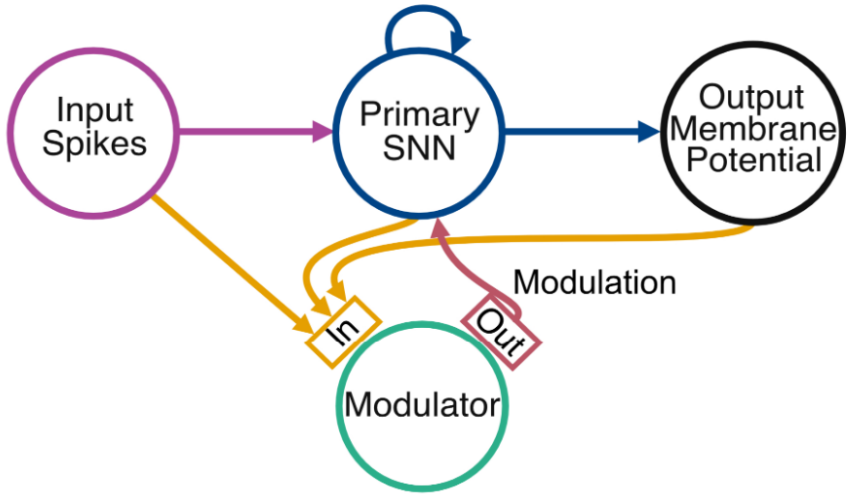
Neuromodulation

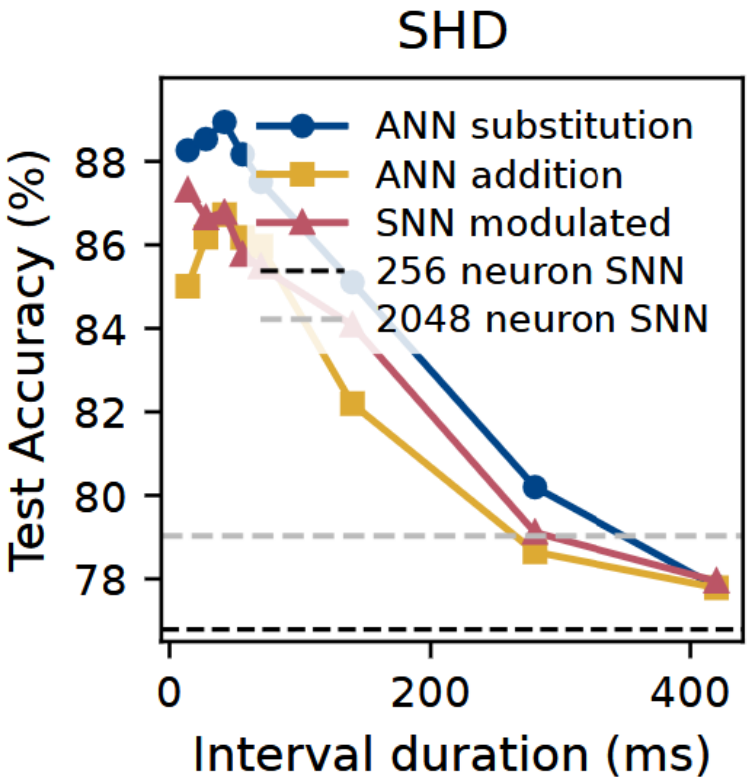
Hot off the press: preprint on this came out on Monday!

All code etc. available on GitHub (including simple tutorial)

Basic question can be framed in one of two ways, pick your favourite:

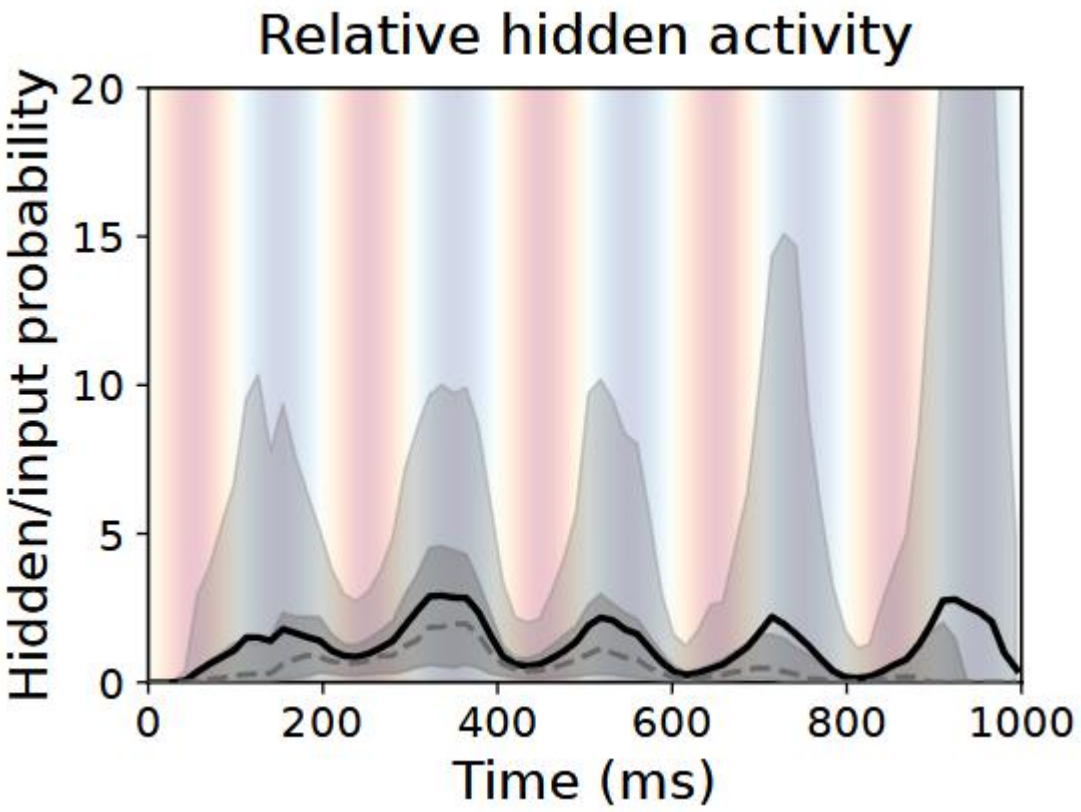
1. If making neurons heterogeneous in space is good, why not in time too?
2. We know the brain has neuromodulators, how could they help computationally?

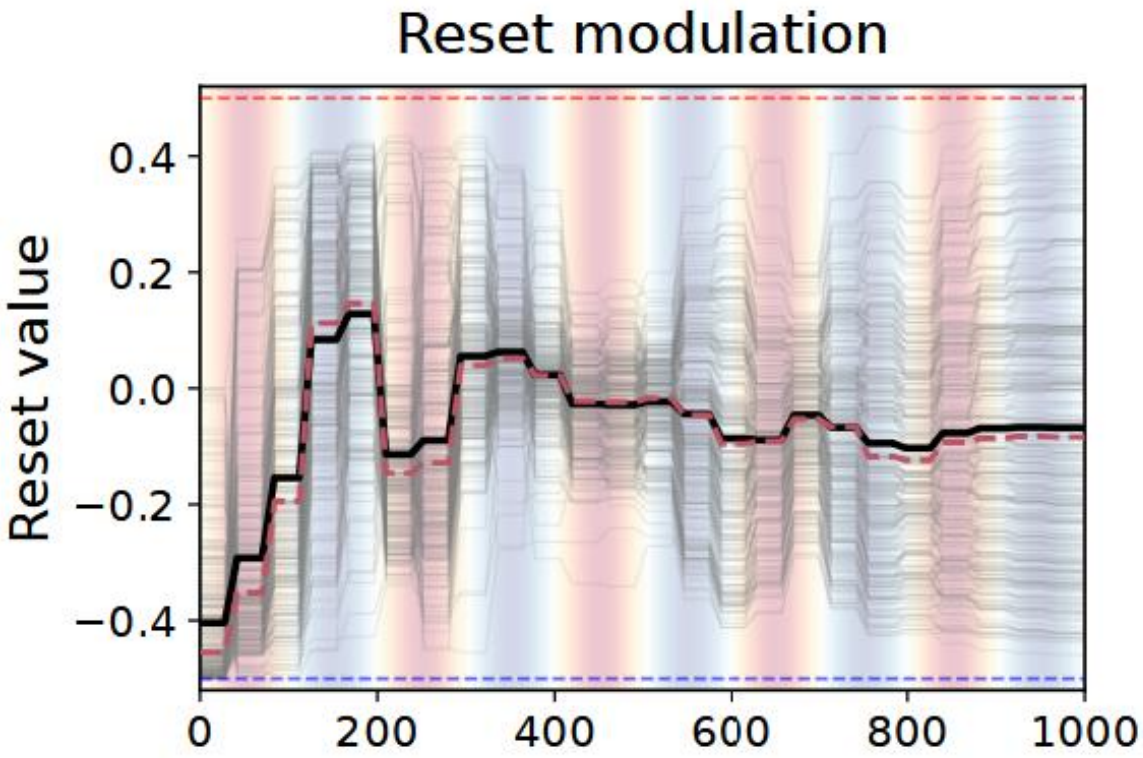




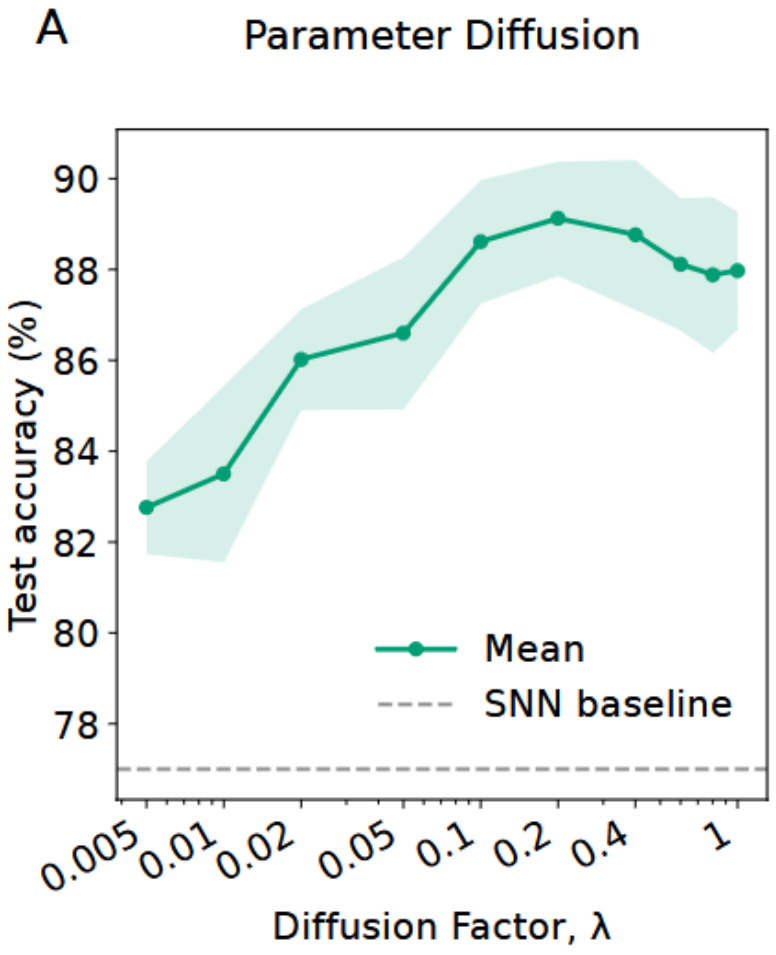
Learns dynamic gain control in fluctuating noise

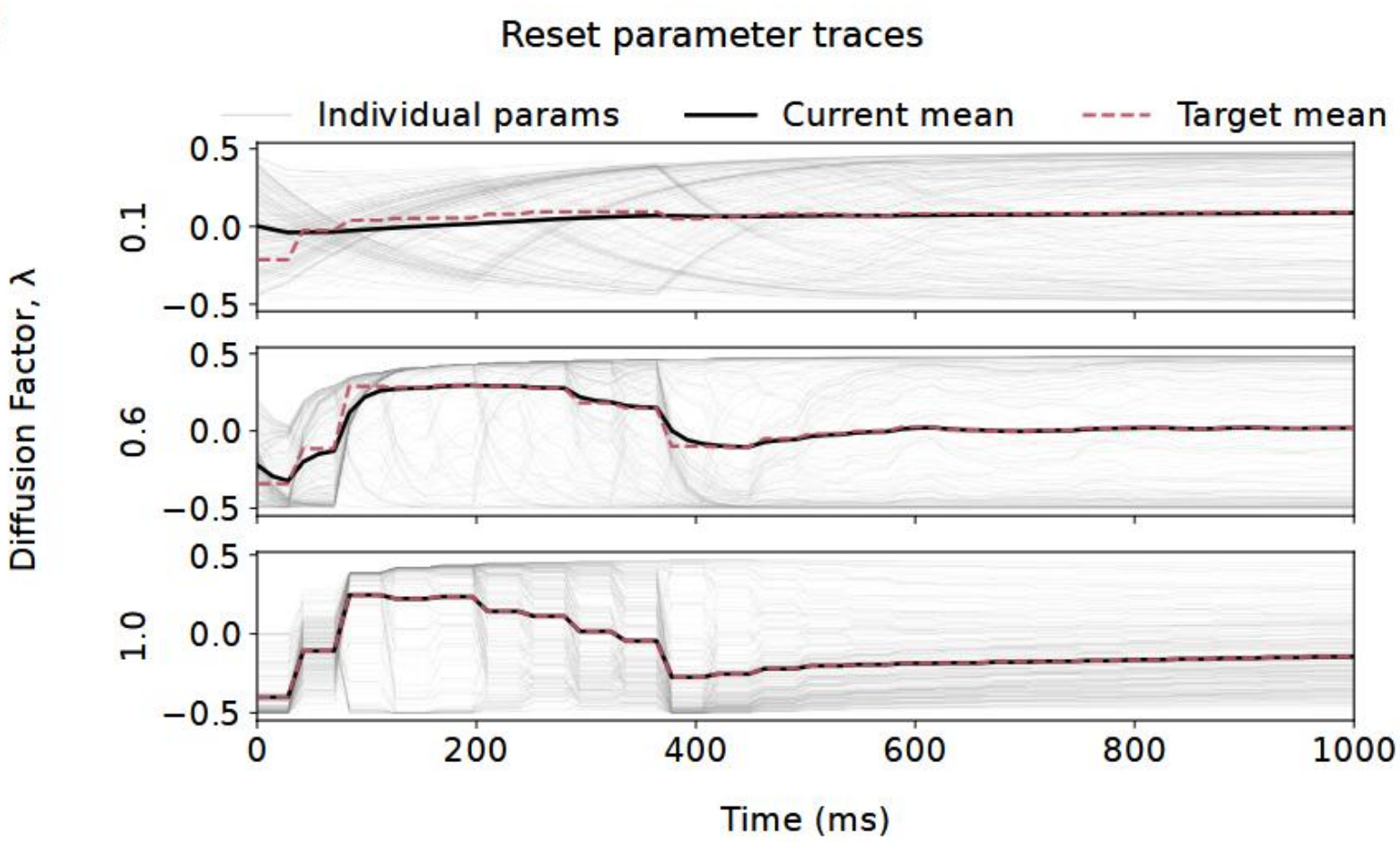
Task: learn spoken digits in a fluctuating background noise



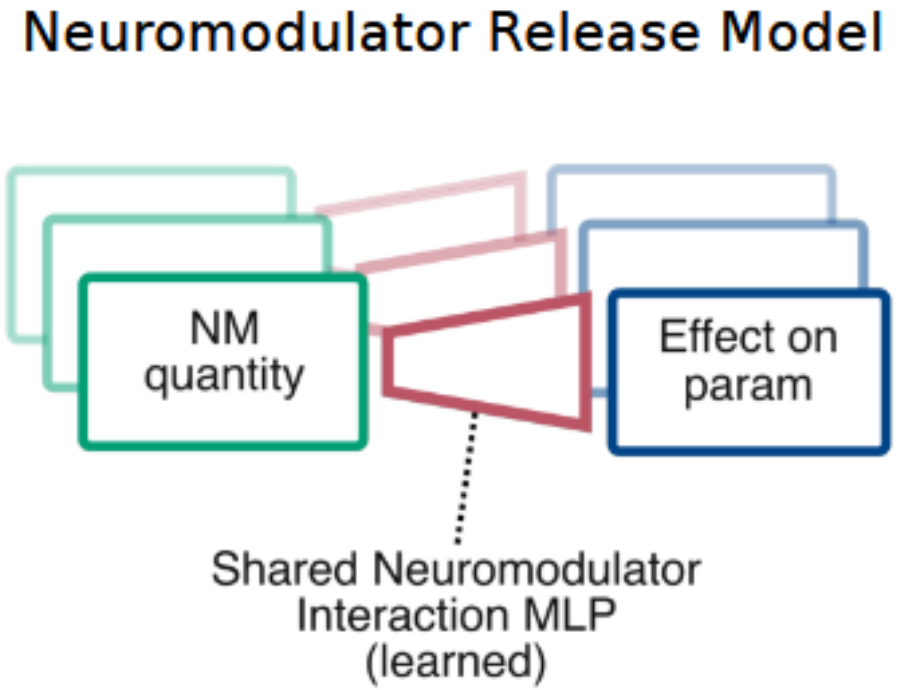


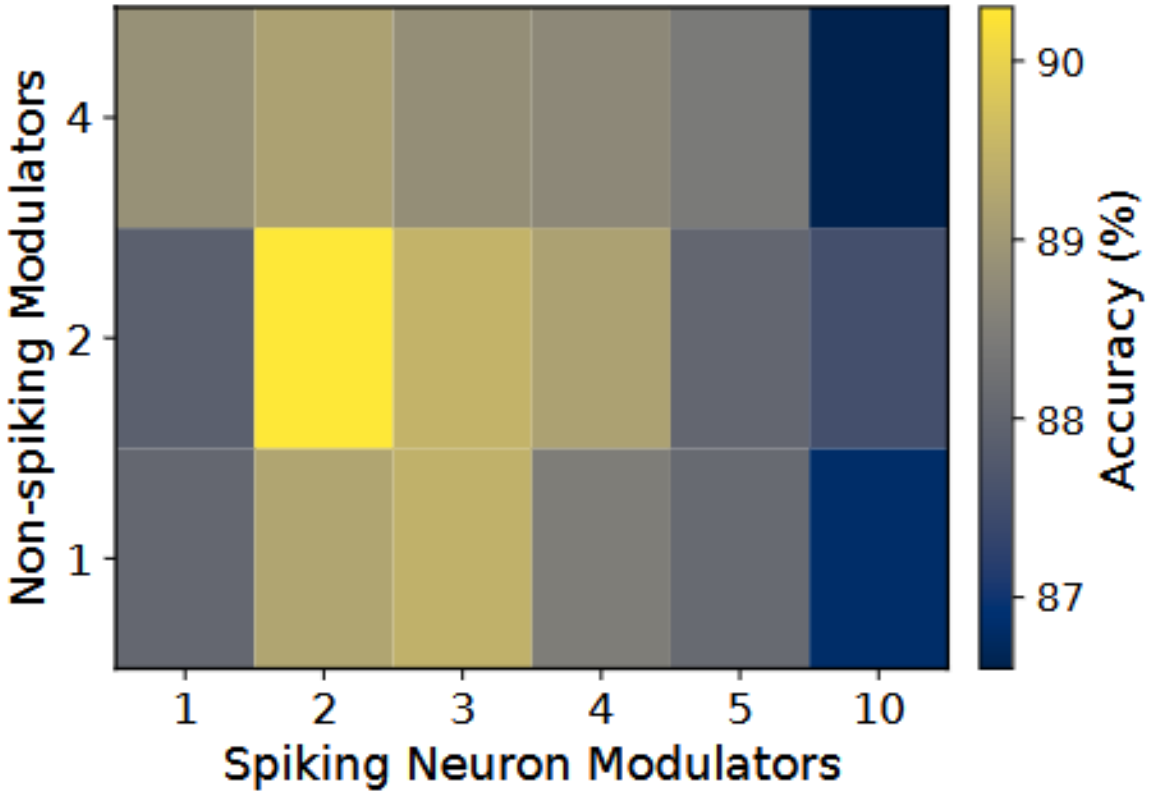
Temporal diffusion can help



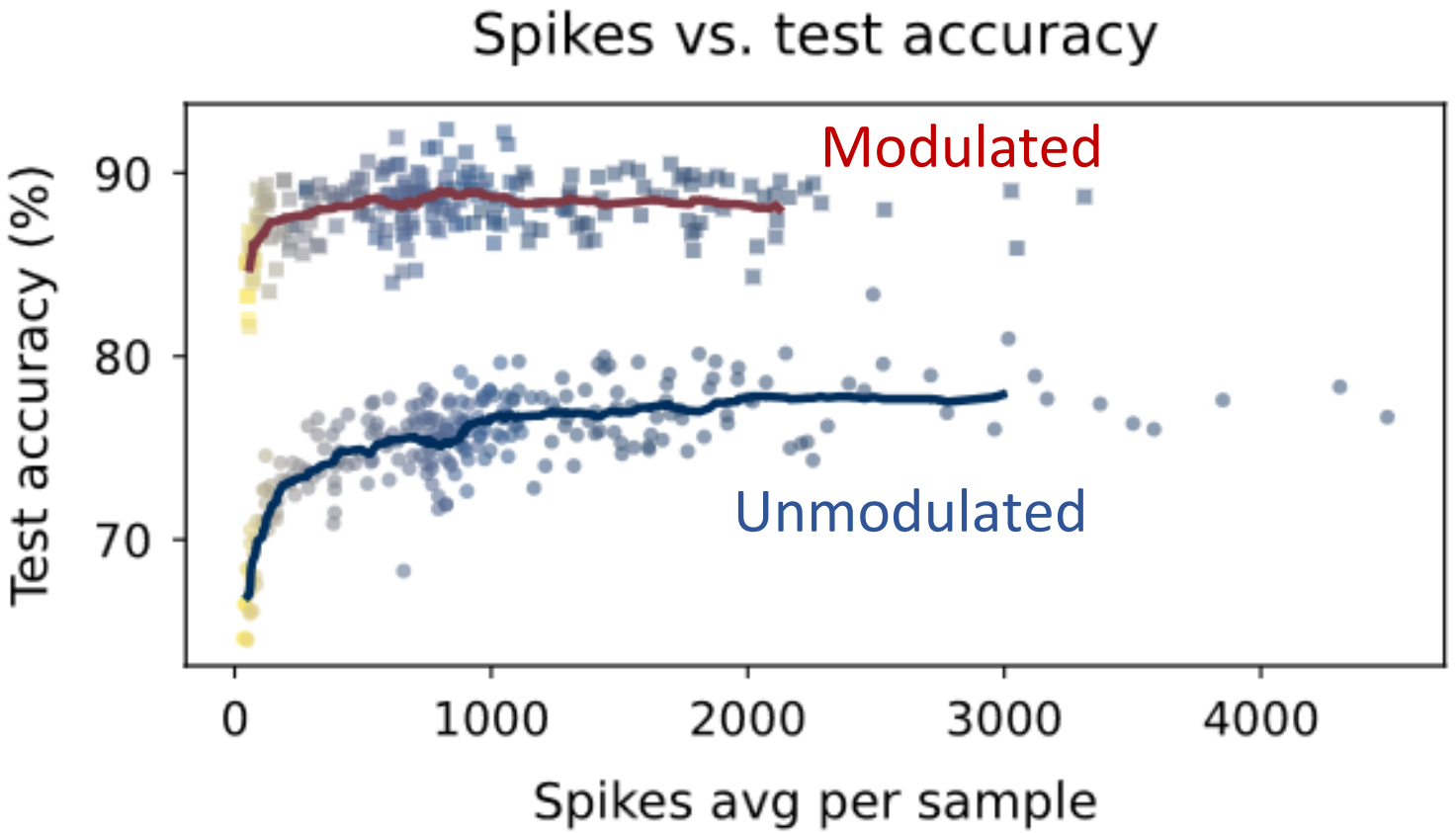


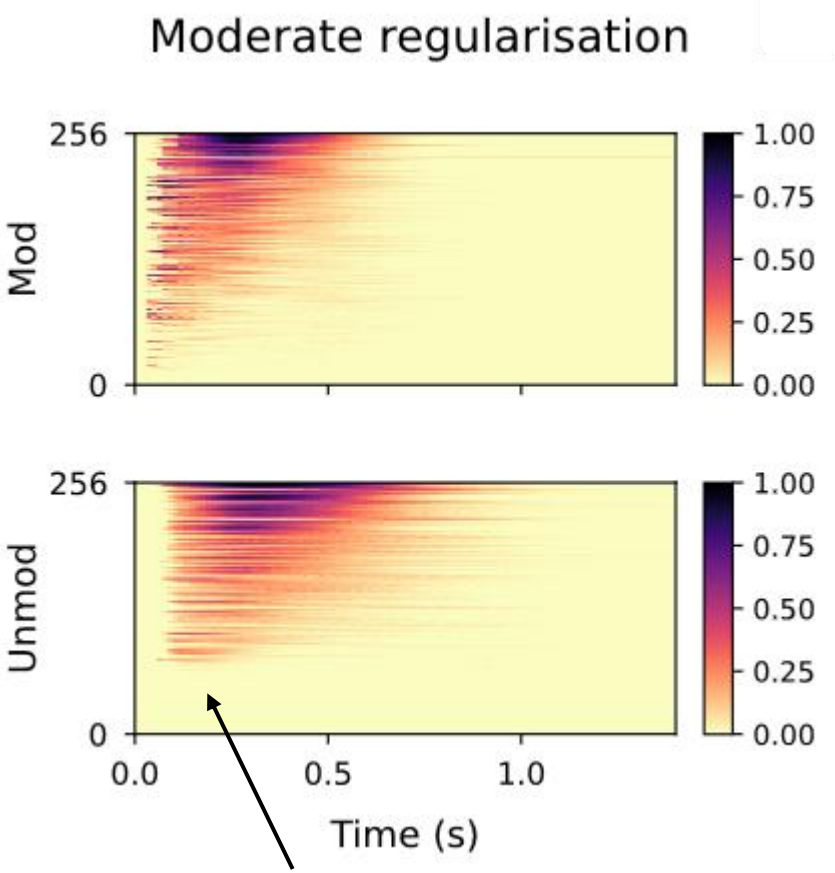
A small number of neuromodulator types is best





Can use many fewer spikes (lower energy cost)





Wasted neurons (never active)

Conclusions, Q&A, Competition

https://github.com/neural-reckoning/cambridge-fens-chen-summer-school-2026

Take home messages

SNNs are the way the brain works

We can test mechanism-function hypotheses using them

Heterogeneity in space and time (with neuromodulators) improve energy efficiency

It's now very easy to write code like this!

Competition and Q&A:

Can you beat my accuracy at the task?

You can change the neuron params, learning params, architecture, model, etc.

Baseline: 90s training to get to about 35% accuracy, 30m to get to about 60% (20 neurons)

I'll be around to help until the end of the session

Thank you!
And also…

Nicolas Perez
Heterogeneity
Now @ DeepMind

Abdal AlKilany
Neuromodulation

Full course available free at https://neuro4ml.github.io