Biological RNN seminar

Puria Radmard - pr450cam.ac.uk

December 2023

1 Introduction and Plan

Artificial recurrent neural networks (RNNs) are a very popular method of mechanistically modelling biological neural networks. In neuroscience, trained RNNs have had a rocky history to get to where they are now. The entire concept of an artificial neuron originated as a model of a biological neuron. An early such example is Rosenblatt's perceptron (Rosenblatt 1958). Later 'deep neural networks' are just nested versions of the perceptron, hence the original name 'multilayered perceptron' (MLP; Haykin 1994) However, for a long time there wasn't enough computational power (or even a good enough algorithm) to train these deep networks. Computational neuroscientists had to design handtuned RNNs based on biophysical rules (Mongillo et al. 2008), or functional ease (Wimmer et al. 2014), and continue to do so (Bouchacourt et al. 2019).

Meanwhile, in the machine learning community, GPU acceleration allowed training of huge networks to take off. Many of these architectures departed from biological realism. For example, the convolutional neural network (CNN) was designed for its ability to process images, but was not based on the eye. Similarly, transformers (Vaswani et al. 2017) which form the basis of GPT models taking the world by storm, were based on their ability to process sequences efficiently, departing from the biologically realistic RNNs for the same tasks. In recent years, neuroscientists have compared CNNs Zhuang et al. 2021 and transformer models (Caucheteux et al. 2022) to human and animal brains, and found that they can account for a surprising amout of brain activity, but the fact remains that their designs are not biological.

The ability to train deep networks means we can also train RNNs on more complex tasks than those which we can handtune networks to Soo et al. 2022. One key example is visual working memory (VWM), where RNNs are typically used to model the pre-frontal cortex (PFC) while it maintains retracted stimulus information over the course of ~1-5 seconds.

In this seminar we will introduce artifical neurons and RNNs from a biological perspective. The history of neural networks above is functionally oriented - new models and architectures are introduced to do a task. We will do the opposite - we will introduce details in our model for the sake of mirroring biology, and only worry about the task at the end. The key building blocks we need for this are:

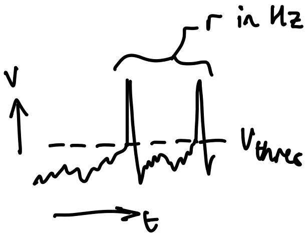
- The single-comparment model: the simplest model of the neuron and its synapses that allows us to perform simulations with many neurons in a network.
- The firing rate model: how the membrane voltage of a single neuron changes over time when it has inputs from other external neurons. We will be covering the rate network model only, and

not leaky integrate-and-fire (LIF) networks that make fewer simplications but are generally harder to program and train on key tasks.

- Introducing recurrence: how we can incorporate these previously external neurons and simulate their behaviour too
- Network training: how we can use backpropagation to train a network on a simple WM task (and how viable is backpropagation in a biological simulation?)

2 Single-compartment models

Neuron histology is intricate. Neurons have complex and diverse shapes, and are their electrochemical properties are hetereogenous both across neuron types and within a single neuron (Figure 1). Even if a neuron was homogeneous in its properties, the attenuation and interaction of electrical signals travelling across the cell body would create a complex electrical topology. To simulate a single neuron, let alone a whole network, we need to simplify our model down to a reasonable level. This simplifcation leaves us with the *single-compartment model* (Dayan et al. 2001).



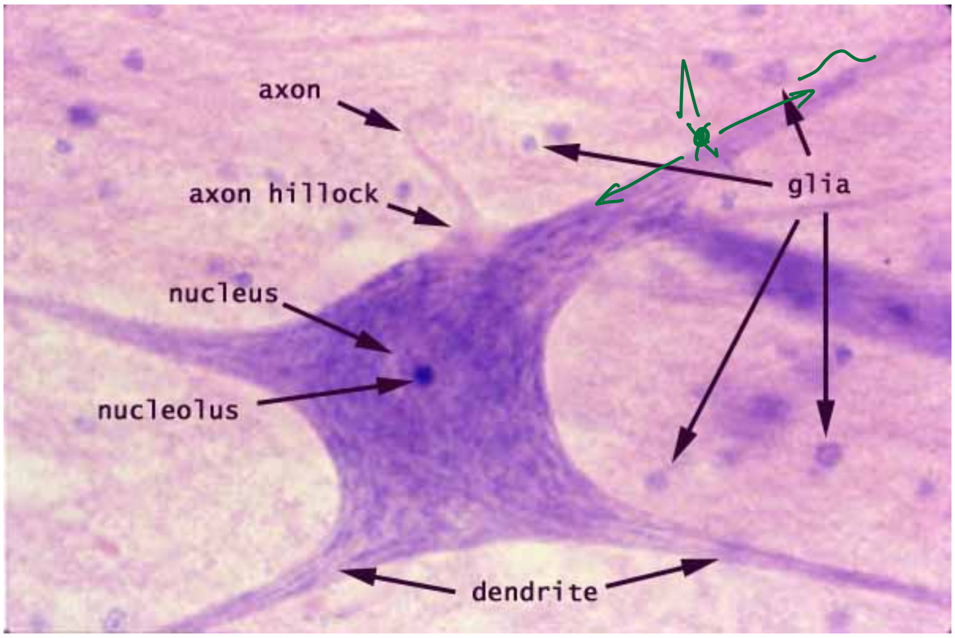
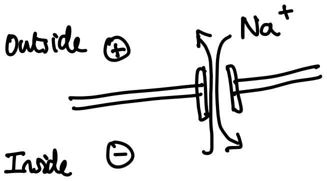


Figure 1: Real neurons are scary!

The key property of this model is that neuron activity is represented by a single membrane voltage. In the full RNN model, where we also track the effect of neural activity on other neurons, we also implicitly assume that action potentials do not attenuate, regardless of which postsynaptic neuron they are delivered to. Neurons effectively become nodes of a graph that do not have spatial dimensions or heterogeneity.

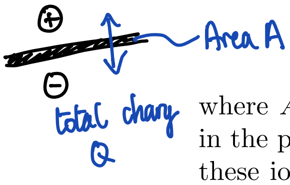


2.1 Electrical theory

We now introduce some basics of neuron membrane modelling. The neuron cell membrane contains ion channels, which selectivly allow different types of ions to flow through. We won't go through the chemical processes that allows this, all we need to know is that there is a gradient of both charge and ion concentration between the cell interior and exterior. These exert opposing electrostatic and diffusive forces on ions, with the *Nerst equations* determining the *reversal potential E* of the ion. This is the potential difference required between the interior and exterior of the cell to hold that ion type at equilibrium, i.e. no flow through the channel due to balancing of these forces. The *resting membrane potential V_{rest}* is the potential difference across the cell membrane, and hovers −70mV (i.e. the inside of the cell is more negatively charged than the exterior) although varies based on cell type and condition This number is the average of the reversal potentials across ion channels on the cell membrane (see below).

Similarly, when the conductance g of one type of ion across the membrane increases, then the membrane voltage V_m moves towards that ion's E value For example, E_{Na^+} and $E_{\mathrm{Ca}^{2+}}$ are both positive, so increased g_{Na^+} and/or $g_{\mathrm{Ca}^{2+}}$ will *depolarise* the cell - make V_m less negative/more positive. Conversely, increasing g_{K^+} will *hyperpolarise* the cell - make it more negative. Synaptic conductances act in a similar way, bringing the postsynaptic membrane potential closer to some E value Synapses with reversal potentials below V_{thres} - the firing threshold potential - are called *inhibitory*, and the opposite is called *excitatory*. *Dale's principle* (Strata et al. 1999) dictates that cells are only presynaptic to one kind of synapse: excitatory or inhibitory.

Again, all of these electrophysical properties are assumed by our single-compartment model to be universal in each cell. We will now rattle through some key cell values that determine their activity and interactions. There is an excess negative charge inside the cell compared to the extracellular matrix, Q. The specific *capacitance* c_m of the cell is given by:



$$Q = \boxed{c_m} A \overset{\text{p.d.}}{V_m}$$

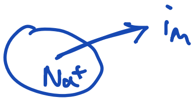
[redacted] (1)

where A is the surface area of the cell. The specific *conductance* g_α for channel of type α is seen in the paragraph above. The *specific membrane current i_m* is the positive current per area through these ion channels due to the flow of ions; it is defined positive-outwards by convention. Because, by definition, specific current through a channel type $i_\alpha = 0$ when $V_m = E_\alpha$, we apply Ohm's law around this point for each channel type:

$$\boxed{i_m} = \sum_{\alpha} g_\alpha (V_m - \overset{\text{rev. pot.}}{\boxed{E_\alpha}})$$

α ← ion type

[redacted] (2)



This assumes that conductance is constant, so only holds for small $\Delta V_\alpha = V_m - E_\alpha$. Models such as the *Hodgkin-Huxley model* deal with the case of *voltage-dependent conductances*

$\tau = RC$

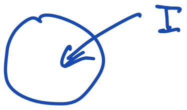
A key electrical property of a cell is its *membrane time constant $\tau_m = c_m/g_m$*, where g_m is the specific conductance averaged over channel type. This is a key ratio we use later. One last thing we should keep in mind is that some external current can also be injected into the cell, which we call I. This is a non-specific quantity and is defined positive-inwards by convention. Overall then, the fundamental equation for single compartment models is the simple *capacitor equation*:

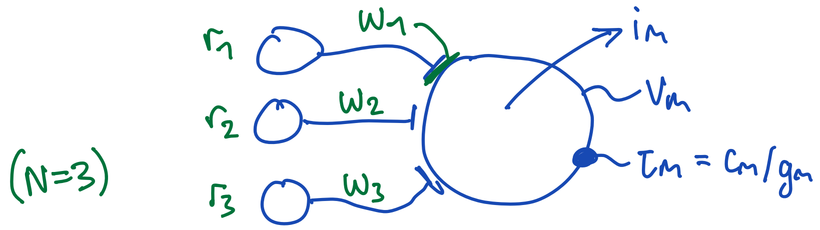
$$\underbrace{\frac{c_m}{g_m}}_{\tau_m} \dot{V}_m = -\underbrace{\frac{i_m}{g_m}}_{V_m} + \frac{\boxed{I}}{A}$$

c.c. →

o.c. →

[redacted] (3)





$$\underline{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix}$$

$$\underline{r}(t) = \begin{bmatrix} r_1(t) \\ \vdots \\ r_N(t) \end{bmatrix}$$

3 Firing rate model

We previously just had the membrane of a neuron with an external current injected into it. Now, we increment model complexity and consider the source of this external current.

We start with a single postsynaptic neuron with N presynaptic neurons feeding into it. The presynaptic neurons are firing at rates $r_1(t), ..., r_N(t)$ at time t, and synapse onto the single shared postsynaptic neuron with strengths $\tilde{w}_1, ..., \tilde{w}_N$. Here, we will slightly deviate from the derivation in the Dayan et al. 2001 book, for the sake of simplicity. Also, the aims of this seminar are to give a hands-on introduction to these networks, so you can always come back and adjust the exact mechanics afterwards. For reasons we do not detail (see the textbook), we take the total input current that this produces as $I = \tilde{\boldsymbol{w}}^\intercal \boldsymbol{r}_u(t)A$, where $\boldsymbol{r}_u(t) = [r_1, ..., r_N]^\intercal \in \mathbb{R}^N_+$ and $\tilde{\boldsymbol{w}} = [\tilde{w}_1, ..., \tilde{w}_N]^\intercal \in \mathbb{R}^N$ are column vectors.

$$\frac{\tilde{w}_i}{g_m} = w_i$$

Let's take a look at the single compartment model equation again. Dividing through by the average specific conductance g_m and assuming Ohm's law roughly holds, we get

$$\boxed{\tau_m \dot{V}_m = -V_m + \underline{w}^T \underline{r}}$$

[redacted]

$$\tau_m \frac{dV_m(t)}{dt} = -V_m(t) + \underline{w}^T \underline{r}(t) \quad (4)$$

where the non-tilde weights absorb the g factor. This is the fundamental rate model equation. Again, you will see different formulations in the textbook. As we build this up to a full recurrent neural network, the different path we took will become clear because our neural network will look different to those in most papers (e.g. Yang et al. 2019). Again, you can come back to the exact form in your own time. Other papers use the form we are considering (e.g. Soo et al. 2022)

3.1 Simulation

We will now look at our first simulation. First, we have to find a way to simulate continuous time dynamics such as that in eq (4) in the first place. We approximate this to discrete-time dynamics using a forward Euler method. This becomes:

$$\tau_m \left(\frac{V_m(t+\delta t) - V_m(t)}{\delta t} \right) = -V_m(t) + \underline{w}^T \underline{r}(t)$$

[redacted] (5)

which we rearranged to:

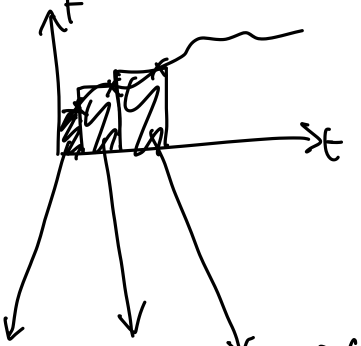
$$V_m(t+\delta t) = V_m(t) + \frac{\delta t}{\tau_m}\left(-V_m(t) + \boldsymbol{w}^\intercal \boldsymbol{r}_u(t)\right) \quad (6)$$

or better:

$$V_m(t+\delta t) = \left(1 - \overset{\alpha}{\cancel{\frac{\delta t}{\tau_m}}}\right) \underline{V_m(t)} + \left(\overset{\alpha}{\cancel{\frac{\delta t}{\tau_m}}}\right) \boldsymbol{w}^\intercal \boldsymbol{r}_u(t) \quad (7)$$

this form makes it clear how the 'leaky integration' aspect of this neuron works: it leaks a portion of its current membrane voltage and integrates in the new presynaptic information it receives. In other words the membrane voltage is a lowpass filtered version of the *input voltage*. The basic feedforward dynamics of eq (7) are coded in `section_4.feed_forward_neuron.PostSynapticNeuron.step_dynamics`.

$$I = \int_{\tau=0}^{t} f(\tau)\, dt$$



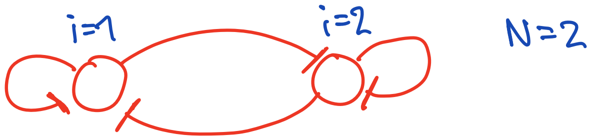
$$I_d = \delta t \left(f(0) + f(\delta t) + f(2\delta t) + \cdots f(t) \right)$$

$$\dot{I}(t) = f(t) \longrightarrow I(t) = \int f(t)$$

$$\frac{I(t+\delta t) - I(t)}{\delta t} = f(t)$$

Discrete time integrator

$$I(t+\delta t) = \underset{(1-\alpha)}{I(t)} + \underset{\alpha}{\delta t}\, f(t)$$



In section_4.main we generate a set of input rate patterns using the following procedure:

$$\begin{aligned} f_i &\sim \mathcal{N}(0,1) \\ \tilde{b}_i &\sim \mathcal{N}(0,1) \\ b_i &= |\tilde{b}_i|\ ; \quad i = 1, ..., N \\ B &= max_i(b_i) \\ r_i(t) &= b_i \cos(2\pi f_i t) + B \end{aligned} \tag{8}$$

and similarly we randomly sample the input weights $w_i \sim \mathcal{N}(0,1)$. Running this script will output a plot of the input voltage and the membrane voltage. Visually, it is clear that the latter is lowpass filtered version of the former.

4 Recurrence

Great, so we have a bunch of presynaptic neurons firing at some pattern, and they synpase onto a single neuron. Now, we want to apply the same treatment to the presynaptic neurons, and simulate their dynamics too. Rather than look at things as a popuation of N presynaptic neurons and 1 postsynaptic neuron, we look at them all them same: 1 of a network of N neurons. The ith neuron in this network will have the same dynamics as before. With a small change in notation, and dropping the explicit dependence on time, we have:

Vm
input weights

$$\tau_m \dot{u}_i = -u_i + \boldsymbol{w}_i^\intercal \boldsymbol{r} \tag{9}$$

Here, u_i is the membrane voltage of the ith neuron in the network, and $\boldsymbol{w}_i = [w_{i1}, ..., w_{iN}]$ is are the strengths of the synapses *onto* neuron i, with w_{ii} being the autapse for this neuron. Annoyingly then , w_{ij} is the weight of the connection from j to i. The remaining term $\boldsymbol{r}$ are the firing rates of the all the neurons in the network. In this model, these rates, in Hz, are set to instataneously track a transformed version of the membrane voltages:

[redacted] $\forall t \quad r_i(t) = f(u_i(t))$ (10)

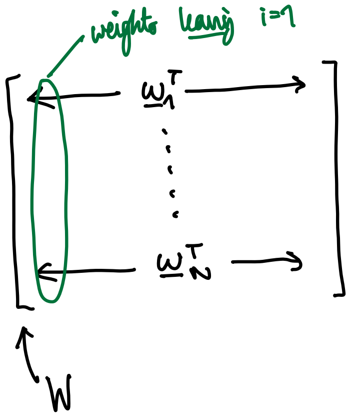
where $f : \mathbb{R} \mapsto \mathbb{R}_+$ is called the *non-linearity function* that transforms real membrane voltages to non-negative firings rates. The choice of non-linearity can completely change the dynamics of the network, and opens up entirely new research directions in cortical dynamics. For simplicity and ease of training networks down the line, we will stick to the ReLU non-linearity: $f(u) = \max(0, u)$.

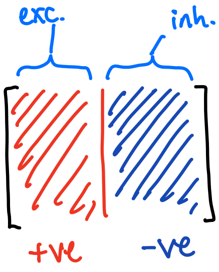
Finally, there is also a noise term $\eta(t)$ typically added. This noise injection may be crucial to the task that we are simulating (Echeveste et al. 2020), or may be required to observe some key network behaviour (Cueva et al. 2021), but is often there as a surrogate for the natural cortical noise in the presence of which any biological network operates. Often, an *Ornstein-Uhlenbeck process*, which smooths a Gaussian white noise process, is used for this noise term. Overall then, the dynamics of our network are:

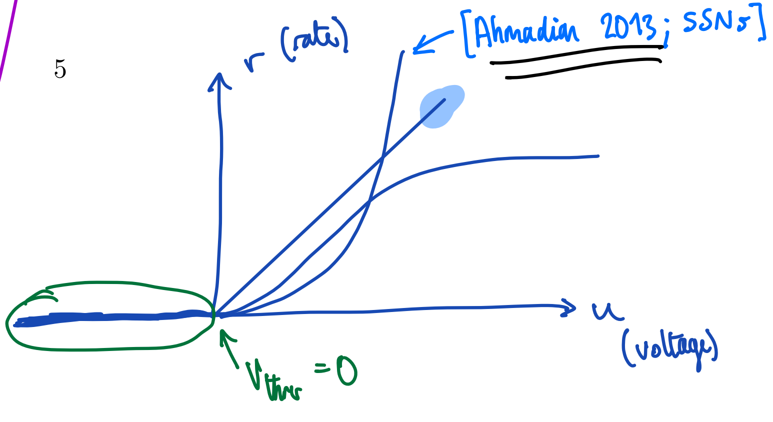
colum
noisy

$$\tau_m \dot{\boldsymbol{u}} = -\boldsymbol{u} + W f(\boldsymbol{u}) + \boldsymbol{\eta} \tag{11}$$

Here, we have stacked the dynamics of individual neurons (eq (10)) as individual rows: $\boldsymbol{u} = [u_1, ..., u_N]^\intercal$ Stare at these equations for a bit to convince yourself that W_{ij} in the *recurrent weight matrix* is the same as w_{ij} we defined above. We will defer defining the Ornstein-Uhlenbeck dynamics until we need to simulate them in discrete time!







$$\underline{u}(t+\delta t) = \left(1-\overset{\frac{\delta t}{\tau_m}}{\alpha}\right)\overset{\in\mathbb{R}^N}{\underline{u}}(t) + \alpha\left[\overset{\in\mathbb{R}^{N\times N}}{W}\,\underset{f(\underline{u}) = \begin{bmatrix} f(u_1)\\ \vdots \\ f(u_N)\end{bmatrix}}{f}(\underline{u}(t)) + \overset{\in\mathbb{R}^N}{\eta}(t)\right]$$

4.1 Simulation

Again, we need to discretise the dynamics of eq (11). Following the same procedure as before, we have:

[redacted] (12)

where the noise process is defined as :

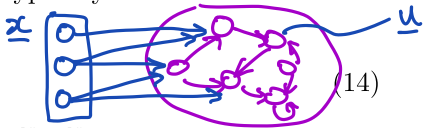
$$\begin{aligned} \text{O-U.P.} \rightarrow \boldsymbol{\eta}(t) &= C\tilde{\boldsymbol{\eta}}(t) \\ \tilde{\boldsymbol{\eta}}(t) &= \varepsilon_1\tilde{\boldsymbol{\eta}}(t-\delta t) + \varepsilon_2\boldsymbol{z}(t) \\ \boldsymbol{z}(t) &\sim \underline{\mathcal{N}(\mathbf{0}, I)} \in \mathbb{R}^N \ \forall\ t \end{aligned} \tag{13}$$

where $\varepsilon_1, \varepsilon_2$ controls the time smoothing of the noise process, and C controls its magnitude. To ensure the noise variance does not explode or vanish, we need $\varepsilon_1^2 + \varepsilon_2^2 = 1$. Can you see why? The script in `section_5.main1` simulates a process for the case where we have just $N = 2$ neurons, then plots the *neural trajectory* of both the membrane voltage and the rate. Running this script many times produces two types of very different behaviour. Why is it that when we run the simulation the neural trajectory sometimes stays near the origin, but other times it explodes to huge numbers?

5 Task-optimised RNN

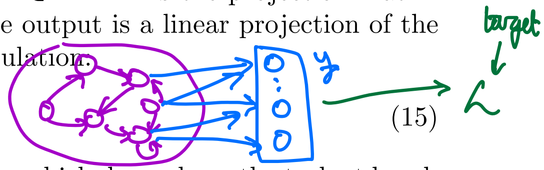
So far, we've only been able to simulate the *autonomous dynamics* in eq (11). This name is given because once the simulation is started, the dynsmics are only driven by recurrence and the noise term. If we want to train a network to perform a task, or even have a network that can perform tasks where the network behaviour depends on some sensory information at all, then we need to have a term in the dynamics that encodes sensory information. This is typically done with an additive term in the dynamics:

$$\tau_m\dot{\boldsymbol{u}} = -\boldsymbol{u} + Wf(\underline{\boldsymbol{u}}) + \boxed{W^{\text{in}}\underline{\boldsymbol{x}}} + \underline{\boldsymbol{\eta}} \tag{14}$$



here, $\boldsymbol{x} = \boldsymbol{x}(t)$ is a time dependent task input, of size N_x. $W^{\text{in}} \in \mathbb{R}^{N_x\times N}$ is the projection matrix from the input space into the network dynamics. Similarly, the output is a linear projection of the network rate. This is meant to mirror a 'readout neurons' population.

$$\text{current} \rightarrow \boldsymbol{y} = \underset{w\ \times}{W^{\text{out}}}\,\underset{r}{f(\boldsymbol{u})} \tag{15}$$

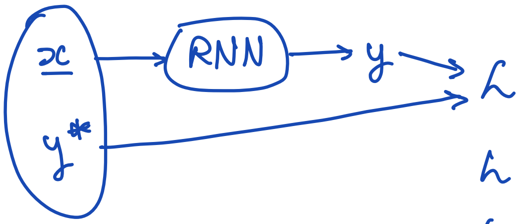


where $W^{\text{in}} \in \mathbb{R}^{N\times N_y}$, with N_y being the size of the output space, which depends on the task at hand.

So far, we haven't defined exactly what the task is. Biological RNNs have been used for simulating tasks involving probabilistic inference (Echeveste et al. 2020), perceptual decision making, sensory integration, etc. In the following subsection we will look at two example working memory task. In general, we use backpropagation through time (BPTT) to train a network on one or multiple tasks. Introducing BPTT is beyond the scope of this course, but we will derive some loss functions for the tasks below.

5.1 Example working memory task

Arguably the most basic visual working memory (VWM) task is the delayed match to sample (DMS) task. In this task, the subject (e.g. monkey) is required to fixate on the center of a display while



$$\mathcal{L} = \underset{\uparrow \text{number of datapoints}}{\frac{1}{N}}\left(y^{(n)} - y^{*(n)}\right)^2$$

$y \approx y^* \rightarrow \mathcal{L}\downarrow$

$y \neq y^* \rightarrow \mathcal{L}\uparrow$

$\mathcal{L}$ high when y is 'bad'

$\mathcal{L}$ low when y is 'good'

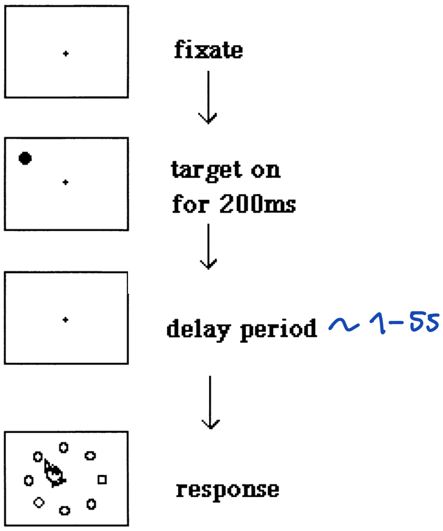


Figure 2: Example DMS task. Adapted from Sohee Park on ResearchGate

some stimulus is presented in their visual periphery (Figure 2). After this, there is a delay period where there is no stimulus except for the fixation signal at the display center. The monkey must maintain fixation until the fixation signal is switched off. At this point, the monkey must saccade to the location of the peripheral stimulus.

A challenge here is to simulate the task elements in a way that maintains the *spatial component* of the task. We can start with a version of the task that does not respect the spatial component of the task, and then move to a better version that does. In both cases there are some key task elements left to be defined:

- Input space and representation
- Output space and representation
- Loss function

In both cases, we assume that the stimulus can appear at N_a possible **a**ngles around the peripheral circle. The angle index for each trial is called $r \in [1, N_a]$, such that $r = j$ means the stimulus appears at $\frac{2\pi}{N_a}j$ radians $= \frac{360}{N_a}j$ degrees around the unit circle. Also in both cases, we have an input size of $N_x = N_a + 1$; the extra one dimension is to represent the fixation cue.

5.1.1 No spatial component

In this version of the task, where we don't care about the spatial component, the locations around the circle are 'independent', so the following input vector:

$$\boldsymbol{x} = [0, ..., 0, 1, 0, ...0, 1]^\intercal \tag{16}$$

is used to represents $r = j$, with only the jth element and the fixation cue active.

~~The output again discards the spatial definition of angles. We can have an output of size $N_y = N_a$, and treat the output as a classification estimation. This means we can use the cross entropy function as our BPTT loss.~~

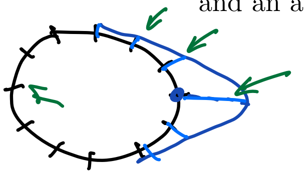
[redacted] (17)

~~Where y_a si the ath element of the network output $\boldsymbol{y}$. If you're not sure why this form of the loss arises, please review the Softmax and Cross Entropy functions.~~

~~It should be clear why there is not spatial component of this task now. The elements of the input and output vectors have no spatil relation. This is due to the one-hot-encoding input we use, and the cross entropy loss that is designed for categorisation. For all we care, the stimuli could be categorical (cats, dogs, apples, pears, ...) instead of points around a circle. In the next version, we making changes to the task elements to ensure that spatial dependence is represented in the task.~~

5.1.2 With spatial component

We can ensure the input units of $\boldsymbol{x}$ have some spatial relation by imposing some non-orthoginality on them. For example, we can do what Yang et al. (2019) did by representing each possible angle r by a *von Mises bump*[1], rather than a one-hot-encoding. In this case, for a stimulus at the rth angle and an active fixation cue:



$$e^{\frac{1}{2}x^2} \longrightarrow e^{\cos x}$$

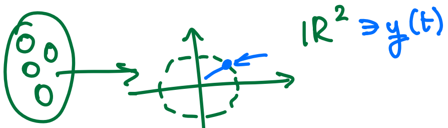
$$x_i = A \exp\left(\kappa \cos\left(\frac{2\pi}{N_a}\underline{(r-i)}\right)\right) \quad \forall\, i = 1, ..., N_a \qquad (18)$$

$x \in \mathbb{R}$ $\quad x \in S^1$

$$x_{N_a+1} = 1 \leftarrow \text{fixatio}$$

Immediately, we have a spatial interpretation of each unit, caused by this continuity between possible stimulus values. We set scalar A and concentration κ as required, and they are often fixed.

Equivalently, we need a way for the output space and loss function to respect the geometry of the saccade task. There are many ways to do this, and instead of following Yang et al. (2019)'s example again, we will use the cosine loss of Stroud et al. 2021. I prefer this method because we use an output space of $N_y = 2$ dimensions, meant to represent the two dimensional space that the monkey is actually saccading in. As such, we devise a loss function that treats the target angle $\frac{2\pi}{N_a}r$ as a location in the same output space. The basic form is the two dimensional squared error:

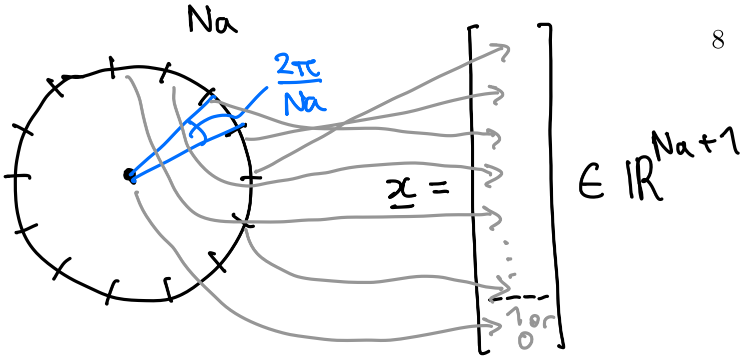


$$\mathcal{L}(\boldsymbol{y}, r) = \left(y_1 - \cos\left(\frac{2\pi}{N_a}r\right)\right)^2 + \left(y_2 - \sin\left(\frac{2\pi}{N_a}r\right)\right)^2 \qquad (19)$$

Here, we have basically embedded the target onto the unit circle, as shown in Figure 3, left. Immediately, we see a spatial interpretation of the loss function, rather than a simple categorisation of possible angles. When we do some reparameterisation of this target embedding, we see another interpretation of this loss, which we call the cosine loss:

$$\mathcal{L}(\boldsymbol{y}, r) = -\lambda \cos\left(\underline{\theta - \frac{2\pi}{N_a}r}\right) \qquad (20)$$

[1] This von Mises curve is the equivalent of the Gaussian bell curve on a circle, and its form is $\exp(\cos(x))$ rather than $\exp(x^2)$



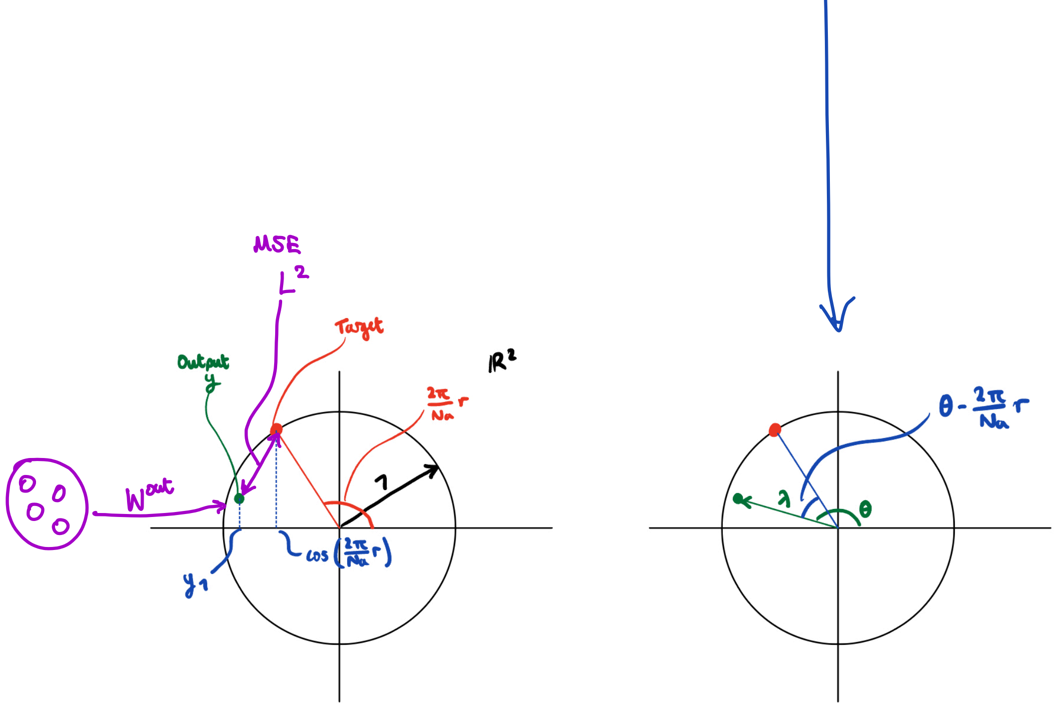


Figure 3: Two ways of showing the spatial loss function between the target on the unit circle and the network output $\boldsymbol{y}$

where (λ, θ) are the polar coordinate expression elements of $\boldsymbol{y}$ This is shown in Figure 3, right. We are essentially penalising the wrong angle of saccade, and penalising it more depending on how much further than the unit circle the network saccades.

5.2 Simulation

Again, the first step is to discretise the network dynamics. The addition of the input term is fairly trivial:

$$\boldsymbol{u}(t+\delta t) = \left(1 - \overset{\alpha}{\cancel{\frac{\delta t}{\tau_m}}}\right)\boldsymbol{u}(t) + \left(\overset{\alpha}{\cancel{\frac{\delta t}{\tau_m}}}\right)\left(Wf(\boldsymbol{u}(t)) + W^{\text{in}}\boldsymbol{x}(t) + \boldsymbol{\eta}(t)\right) \tag{21}$$

and similarly, we have

$$\mathbb{R}^2 \ni \underline{\underline{\boldsymbol{y}(t) = W^{\text{out}} f(\boldsymbol{u}(t))}} \tag{22}$$

at each timestep.

Then, we have to decide what we're plugging into the network as the input $\boldsymbol{x}(t)$. There are four task phases for each trial:

Task phase	**Description**	**Number of active input units (non-spatial)**	**Number of active input units (spatial)**
Pre-stimulus	Waiting for activity to settle before we show the stimulus	1	1
Stimulus	Stimulus is presented	1 + 1	$N_a + 1$
Delay	Working memory delay	1	1
Response	Network must respond with a saccade	0	0

The two tasks are implemented in `section_6.main_non_spatial` and `section_6.main_spatial` respectively. Notice the difference in the way we call `rnn.step_dynamics` for the response phase, versus all the previous phases. Why is that? Running these will generate summaries of the training process, including a loss curve over the course of many trials. Try playing around with A, κ, N, and the durations of each task phase. What do you see change in the learning ability of the network?

Extension exercise 1: how can we implement a network that adheres to Dale's principle (Strata et al. 1999)? Make sure you do it in a way that allows PyTorch to freely perform gradient-based optimisation on the network.

Extension exercise 2: does the network activity differ for the two tasks? How did the changes in the task set-up cause this?

Extension exercise 3: our loss functions are averaged over the timesteps of the response period (see `section_6.utils.loss_function_non_spatial` and `section_6.utils.loss_function_spatial`). This means that the network has to maintain a 'good' output during the response phase, but is not penalised for doing the same during the delay period. For the second version of the task, where the output mirrors an actual eye fixation in a 2D plane, can you come up with a way to penalise breaking of fixation before the fixation signal is turned off?

References

Bouchacourt, Flora, and Timothy J. Buschman. 2019. “A Flexible Model of Working Memory.” *Neuron* 103, no. 1 (July): 147–160.e8.

Caucheteux, Charlotte, Alexandre Gramfort, and Jean-Rémi King. 2022. “Deep language algorithms predict semantic comprehension from brain activity.” *Scientific Reports* 12, no. 1 (September).

Cueva, Christopher J., Adel Ardalan, Misha Tsodyks, and Ning Qian. 2021. *Recurrent neural network models for working memory of continuous variables: activity manifolds, connectivity patterns, and dynamic codes.*

Dayan, P., and L.F. Abbott. 2001. *Theoretical Neuroscience.* Cambridge, MA: MIT Press.

Echeveste, Rodrigo, Laurence Aitchison, Guillaume Hennequin, and Máté Lengyel. 2020. “Cortical-like dynamics in recurrent circuits optimized for sampling-based probabilistic inference.” *Nature Neuroscience* 23, no. 9 (August): 1138–1149.

Haykin, Simon. 1994. *Neural networks: a comprehensive foundation.* Prentice Hall PTR.

Mongillo, Gianluigi, Omri Barak, and Misha Tsodyks. 2008. “Synaptic Theory of Working Memory.” *Science* 319, no. 5869 (March): 1543–1546.

Rosenblatt, F. 1958. “The perceptron: A probabilistic model for information storage and organization in the brain.” *Psychological Review* 65 (6): 386–408.

Soo, Wayne, and Mate Lengyel. 2022. “Training stochastic stabilized supralinear networks by dynamics-neutral growth.” In *Advances in Neural Information Processing Systems,* edited by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, 35:29278–29291. Curran Associates, Inc.

Strata, Piergiorgio, and Robin Harvey. 1999. “Dale’s principle.” *Brain Research Bulletin* 50, no. 5–6 (November): 349–350.

Stroud, Jake P., Kei Watanabe, Takafumi Suzuki, Mark G. Stokes, and Máté Lengyel. 2021. “Optimal information loading into working memory in prefrontal cortex explains dynamic coding” (November).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. “Attention is All you Need.” In *Advances in Neural Information Processing Systems,* edited by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, vol. 30. Curran Associates, Inc.

Wimmer, Klaus, Duane Q Nykamp, Christos Constantinidis, and Albert Compte. 2014. “Bump attractor dynamics in prefrontal cortex explains behavioral precision in spatial working memory.” *Nature Neuroscience* 17, no. 3 (February): 431–439.

Yang, Guangyu Robert, Madhura R. Joglekar, H. Francis Song, William T. Newsome, and Xiao-Jing Wang. 2019. “Task representations in neural networks trained to perform many cognitive tasks.” *Nature Neuroscience* 22, no. 2 (January): 297–306.

Zhuang, Chengxu, Siming Yan, Aran Nayebi, Martin Schrimpf, Michael C. Frank, James J. DiCarlo, and Daniel L. K. Yamins. 2021. “Unsupervised neural network models of the ventral visual stream.” *Proceedings of the National Academy of Sciences* 118, no. 3 (January).