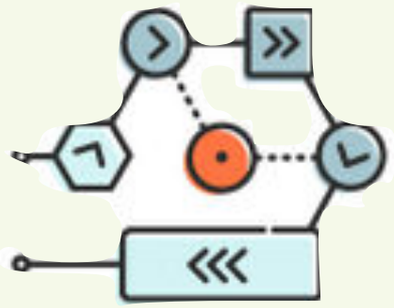
DEEP LEARNING



ALGORITHM

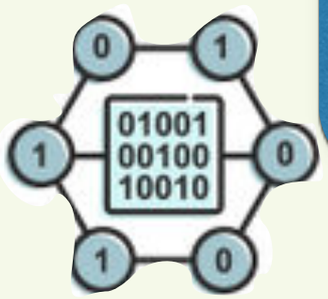
LEARNING

AI

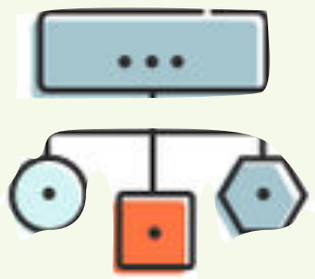
COURSE
"APLICAÇÕES DE INTELIGÊNCIA ARTIFICIAL"
University of Aveiro

Mestrado em Eng. Computational

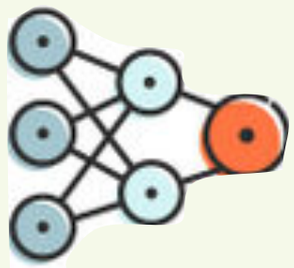
IMPROVES



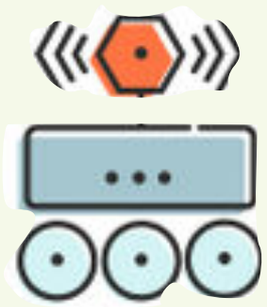
DATA MINING



CLASSIFICATION



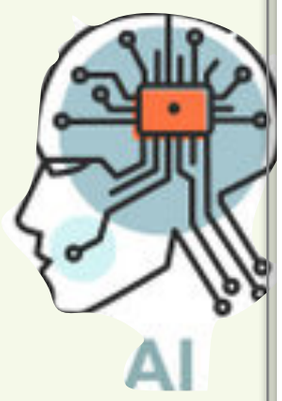
NEURAL NETWORKS

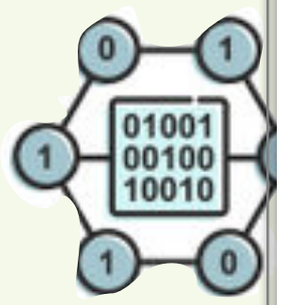


AUTONOMUS

ANALYZE

EXTRA INFO - this label will appear in slides with complementary information



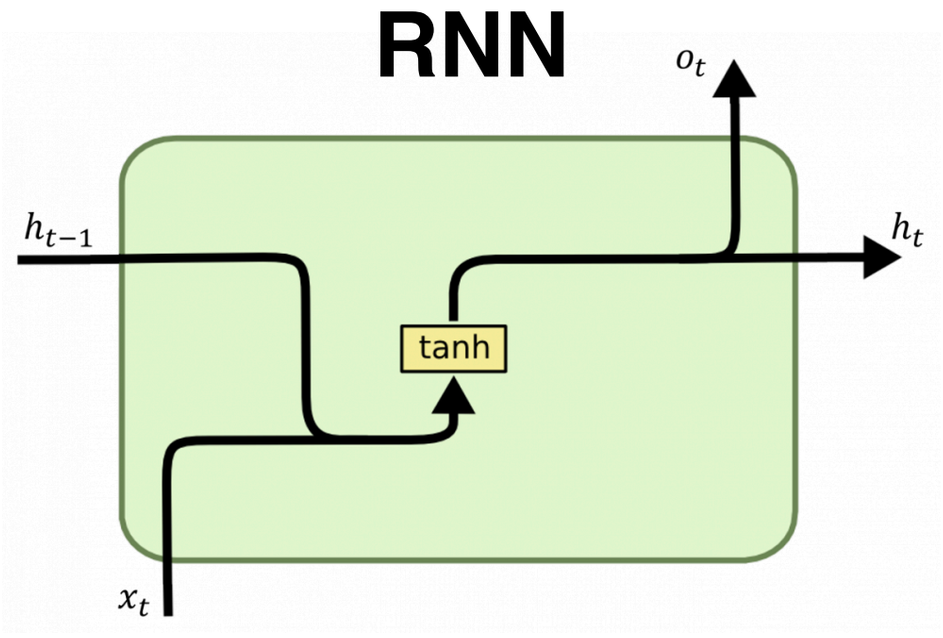


SUMMARY: Recurrent Neural Networks

[T] RNN model, relevant variants and types, examples, the mathematical theory;

[P] Exploring a complete RNN code example.

EXTRA INFO - this label will appear in slides with complementary information

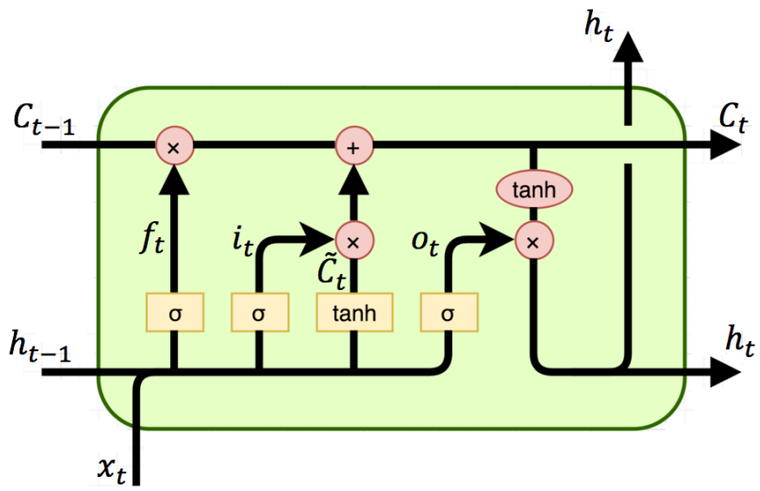


Feed-Forward

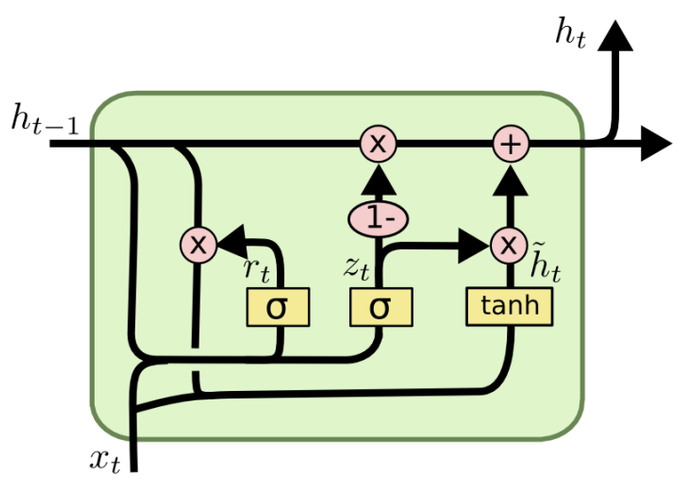
$$h_t = \sigma_h(i_t) = \sigma_h(U_h x_t + V_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(a_t) = \sigma_y(W_y h_t + b_h)$$

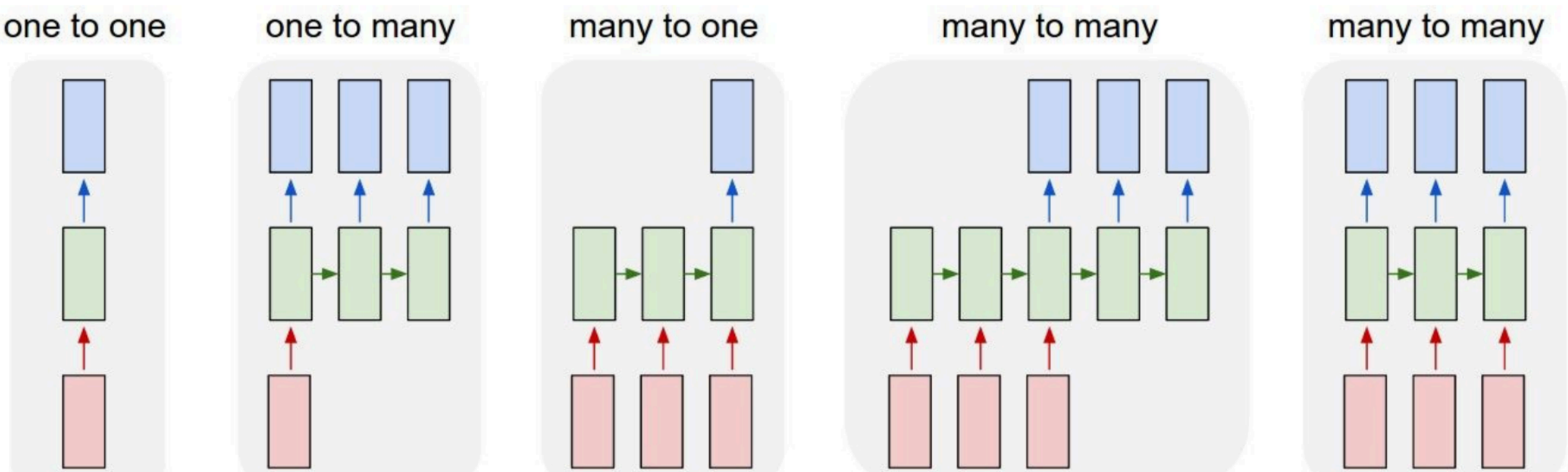
LSTM – LONG-SHORT TERM MEMORY



GRU – GATED RECURRENT UNIT



Recurrent Neural Networks: Process Sequences



e.g. **Image Captioning**
image -> sequence of words

e.g. **Sentiment Classification**
sequence of words -> sentiment

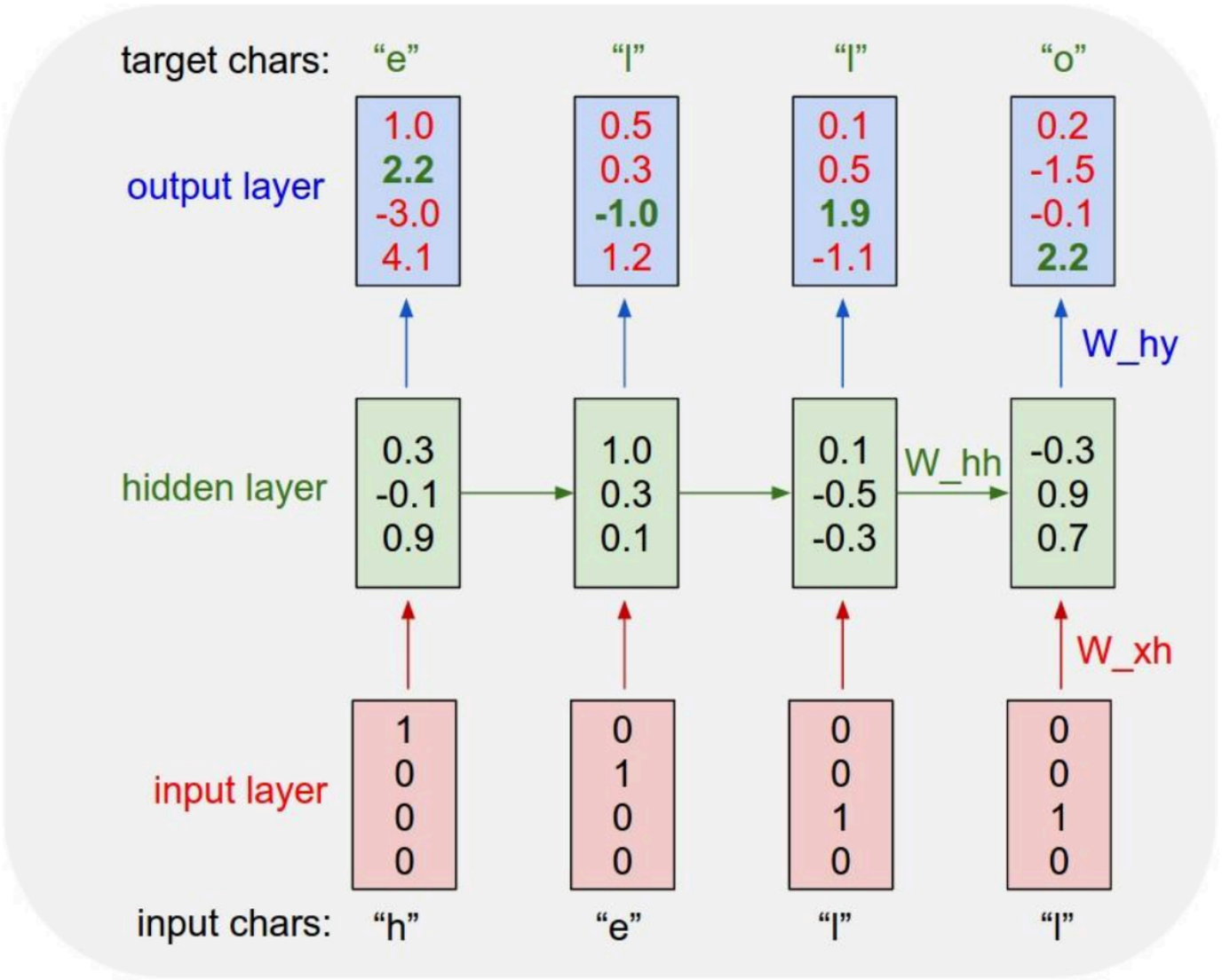
e.g. **Machine Translation**
seq of words -> seq of words

e.g. **Video classification on frame level**

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

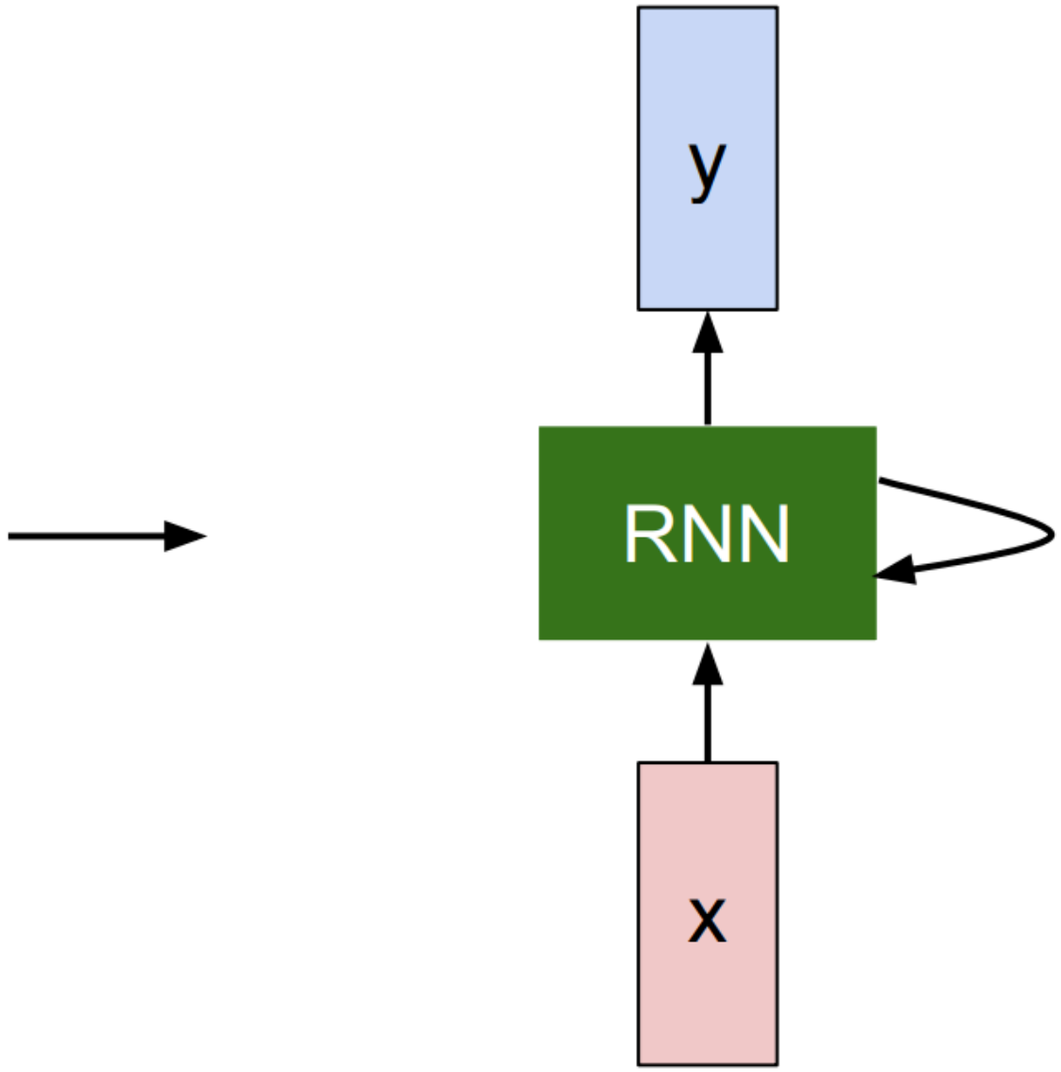


THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
    Pity the world, or else this glutton be,
    To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
    This were to be new made when thou art old,
    And see thy blood warm when thou feel'st it cold.



at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Feed-forward neural networks

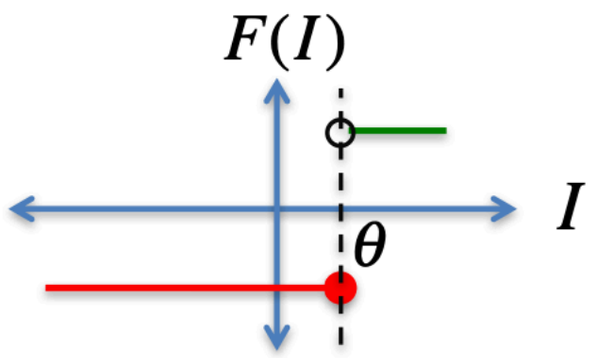
(revision)

- Synaptic input is the firing rate of the input neuron times a synaptic weight w.

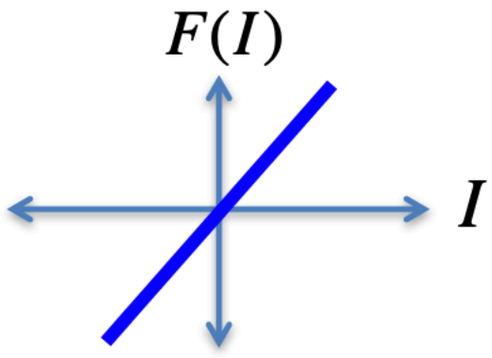
$$I_s = wu$$

- The output firing rate is some non-linear function of the synaptic input.

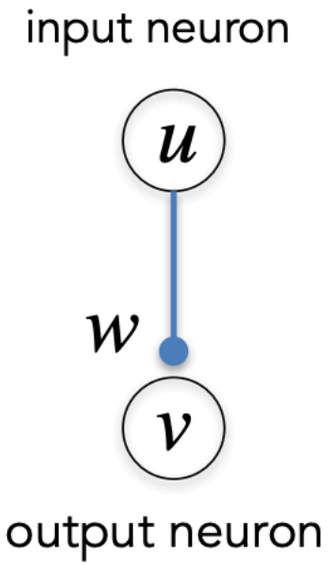
$$v = F[I_s] = F[wu]$$



binary threshold neuron



linear neuron $v = wu$

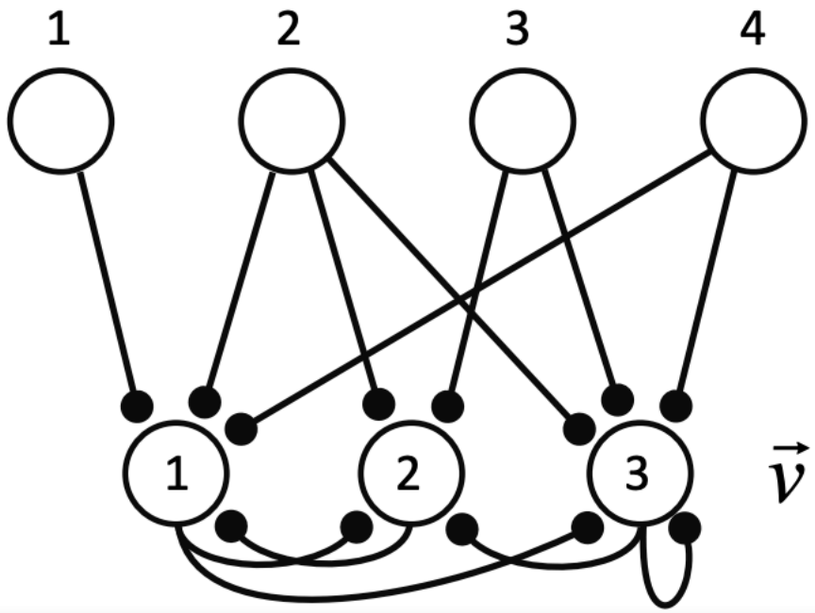


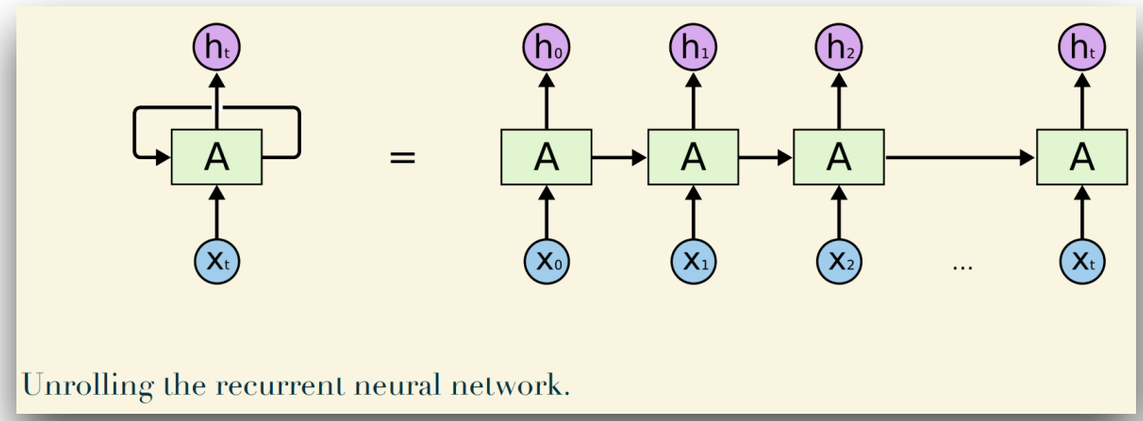
Recurrent networks

- Today we will consider the case where there are also connections between different neurons in the output layer

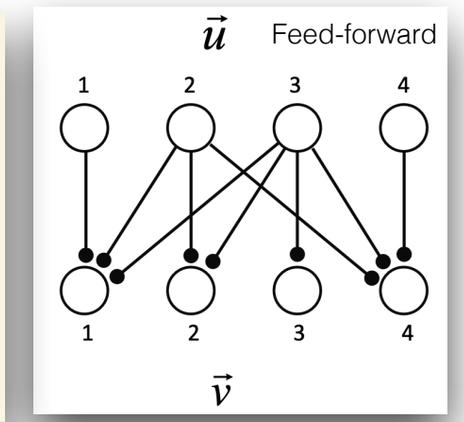
- Develop an intuition for how recurrent networks respond to their inputs

- Examine computations performed by recurrent networks (amplifier, integrator, sequence generation, short term memory





Unrolling the recurrent neural network.



Time dependence

- The steady state firing rate of our output neuron looks like this…

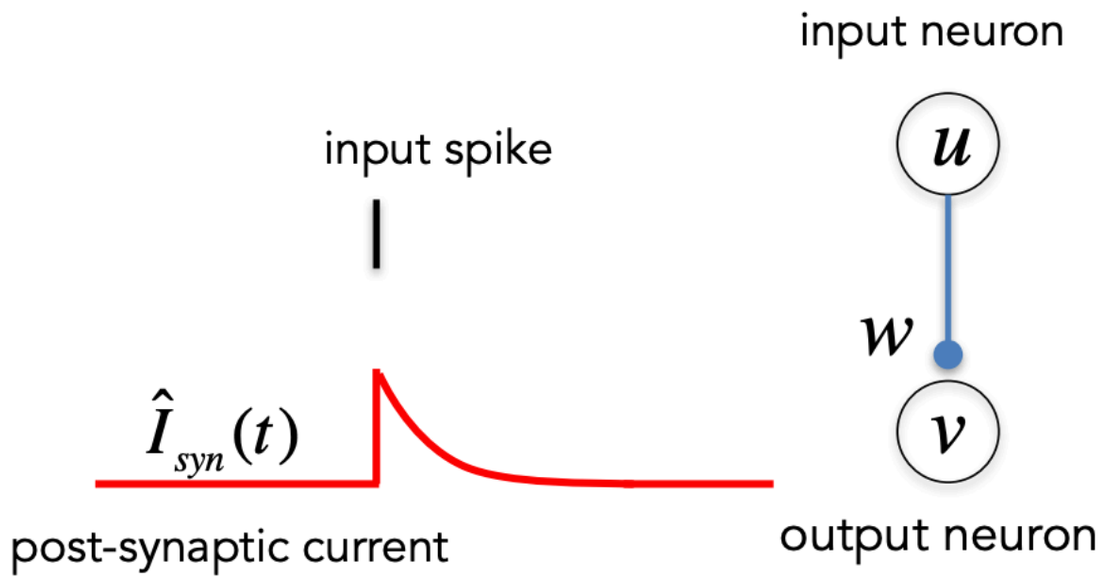
$$v_\infty = F\left[I_s\right] = F\left[wu\right]$$

- But neurons don't respond instantaneously to current inputs

Synaptic delays

Dendritic propagation

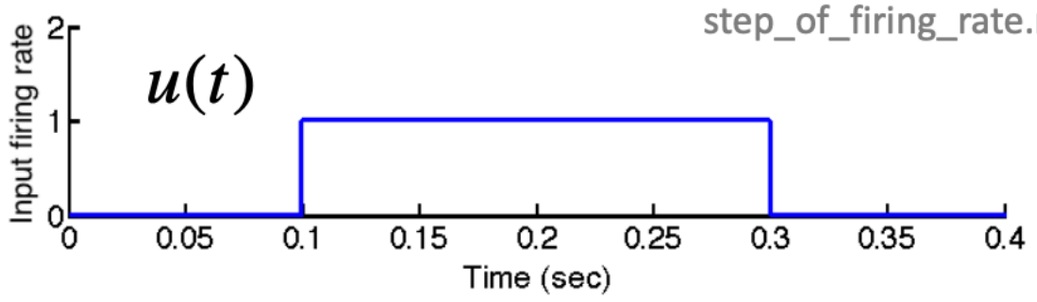
Membrane time constant



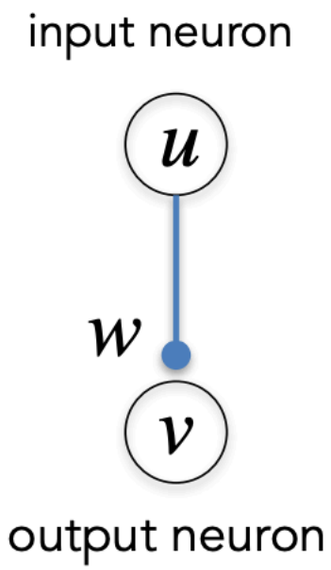
Time dependence

- We model the firing rate of our model neuron as follows:

step_of_firing_rate.m



$$\tau_n \frac{dv}{dt} = -v + v_\infty$$



- We will look at how networks respond to changes in their inputs

$$v_\infty(t) = F\left[wu(t)\right]$$

$$\tau_n \frac{dv}{dt} = -v + F\left[wu(t)\right]$$

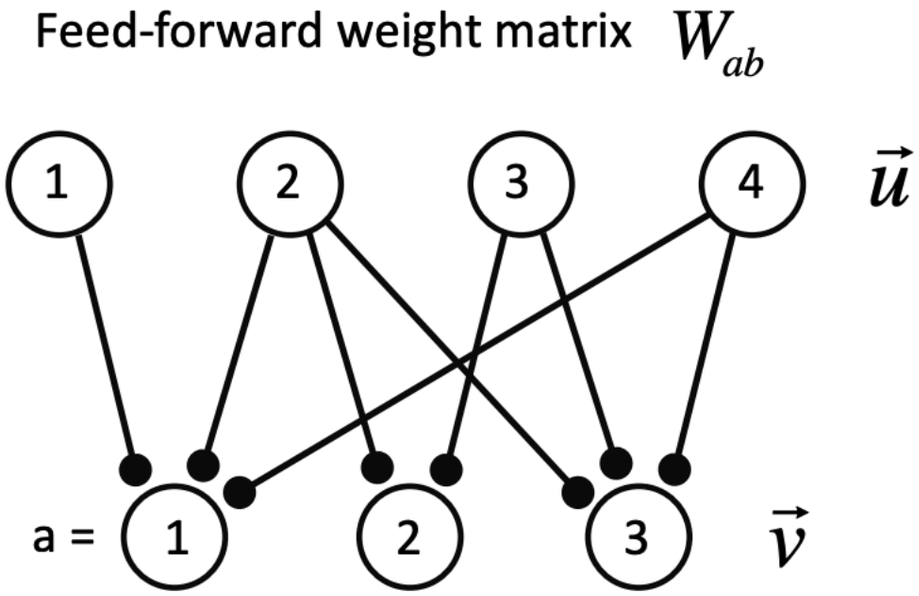
Time dependence

- We can incorporate time-dependence into our general feed-forward network…

$$\tau_n \frac{d\vec{v}}{dt} = -\vec{v} + \vec{v}_\infty$$

$$\vec{v}_\infty = F\left[W\,\vec{u}\right]$$

$$\tau_n \frac{d\vec{v}}{dt} = -\vec{v} + F\left[W\,\vec{u}\right]$$



- The time dependence is really boring in a feedforward network, but it is extremely important in RNNs.

Recurrent networks

- We will now consider the case where there are connections between different neurons in the output layer

Feed-forward input

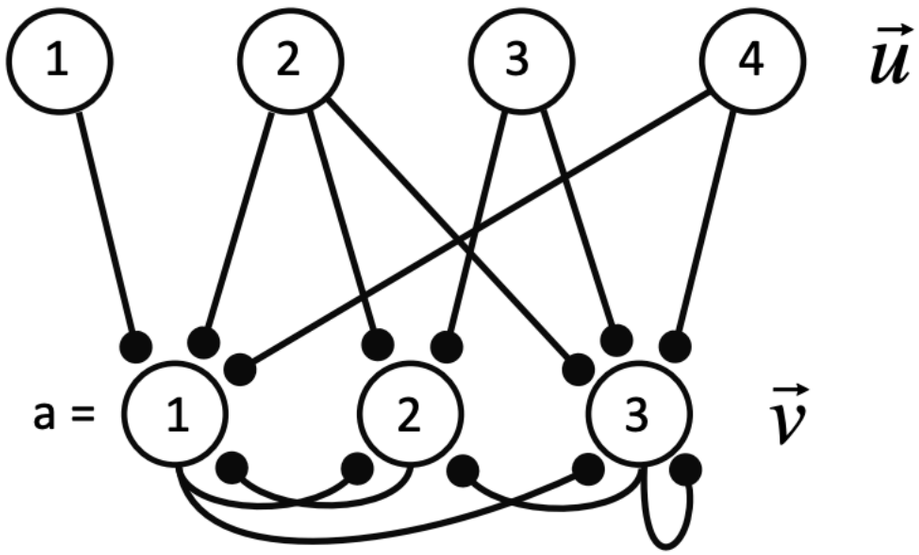
$$W\vec{u} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

Recurrent input

$$M\vec{v} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$\tau_n \frac{d\vec{v}}{dt} = -\vec{v} + F\left[W\vec{u} + M\vec{v}\right]$$

Feed-forward weight matrix W_{ab}



Recurrent weight matrix $M_{aa'}$

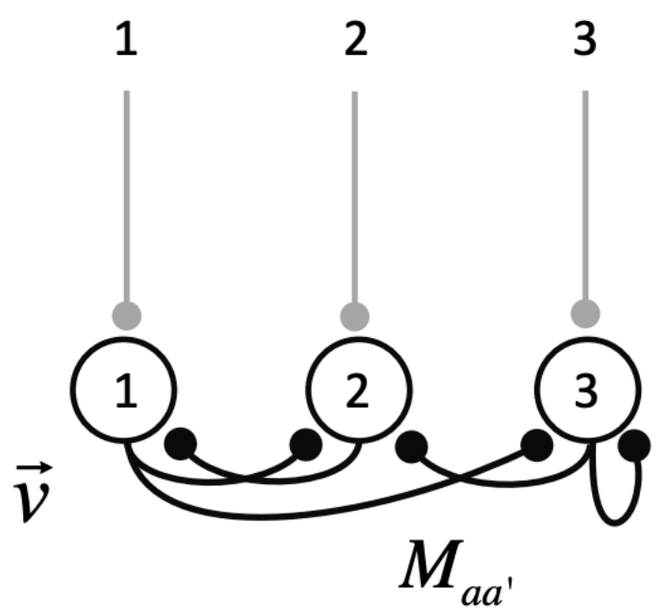
to **a** from **a'**

Recurrent networks

- We will simplify this equation to focus on the recurrent network

- Rather than writing the input as a vector of input firing rates, write a vector of effective inputs to each output neuron.

$$\vec{h} = W\,\vec{u}$$



$$\tau_n \frac{d\vec{v}}{dt} = -\vec{v} + F\left[\vec{h} + M\,\vec{v}\right]$$

Recurrent networks

- We will start by analyzing the case with linear neurons

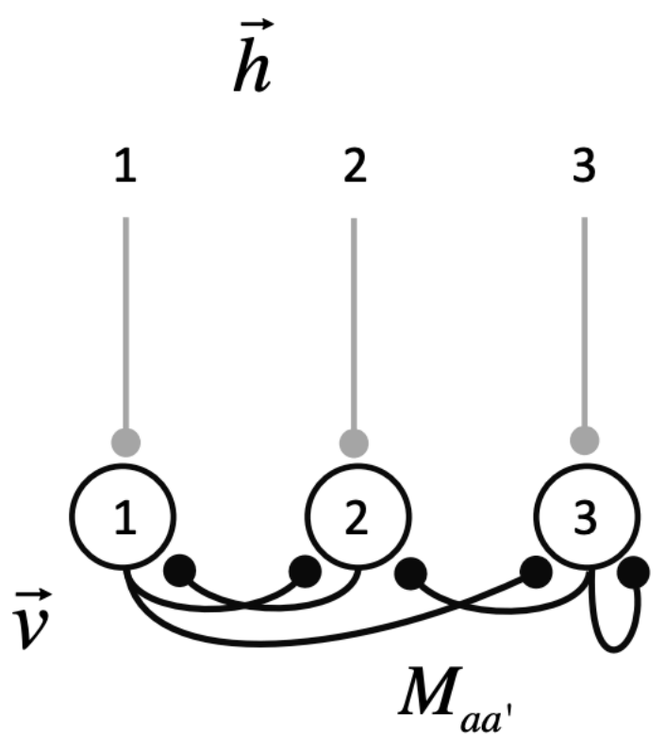
$$\tau_n \frac{d\vec{v}}{dt} = -\vec{v} + F\left[\vec{h} + M\,\vec{v}\right]$$

- For linear neurons

$$F(\vec{x}) = \vec{x}$$

Thus…

$$\tau_n \frac{d\vec{v}}{dt} = -\vec{v} + M\,\vec{v} + \vec{h}$$



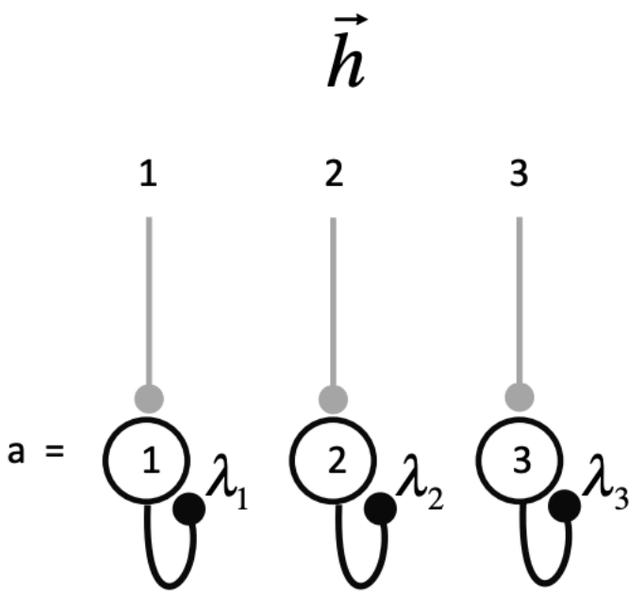
This is a system of coupled equations!

Recurrent networks

Autapse

- Note that if M is a diagonal matrix

$$M = \Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \lambda_3 \\ & & & \ddots \end{pmatrix}$$

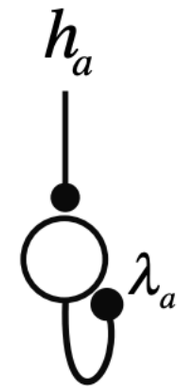


$$\tau_n \frac{d\vec{v}}{dt} = -\vec{v} + \Lambda\vec{v} + \vec{h}$$

$$\tau_n \frac{dv_a}{dt} = -v_a + \lambda_a v_a + h_a$$

We have n independent equations – each neuron acts independently of all the others

Recurrent networks



- Rewrite our equation:

$$\tau_n \frac{dv_a}{dt} = -v_a + \lambda_a v_a + h_a$$

- There are three cases to consider

$$\tau_n \frac{dv_a}{dt} = -\underbrace{(1-\lambda_a)}_{>0 \quad =0 \quad <0} v_a + h_a$$

- Start with the case that: $\lambda_a < 1$

$$\underbrace{\frac{\tau_n}{1-\lambda_a}}\frac{dv_a}{dt} = -v_a + \underbrace{\frac{h_a}{1-\lambda_a}}$$

$$\tau_a \frac{dv_a}{dt} = -v_a + v_{a,\infty}$$

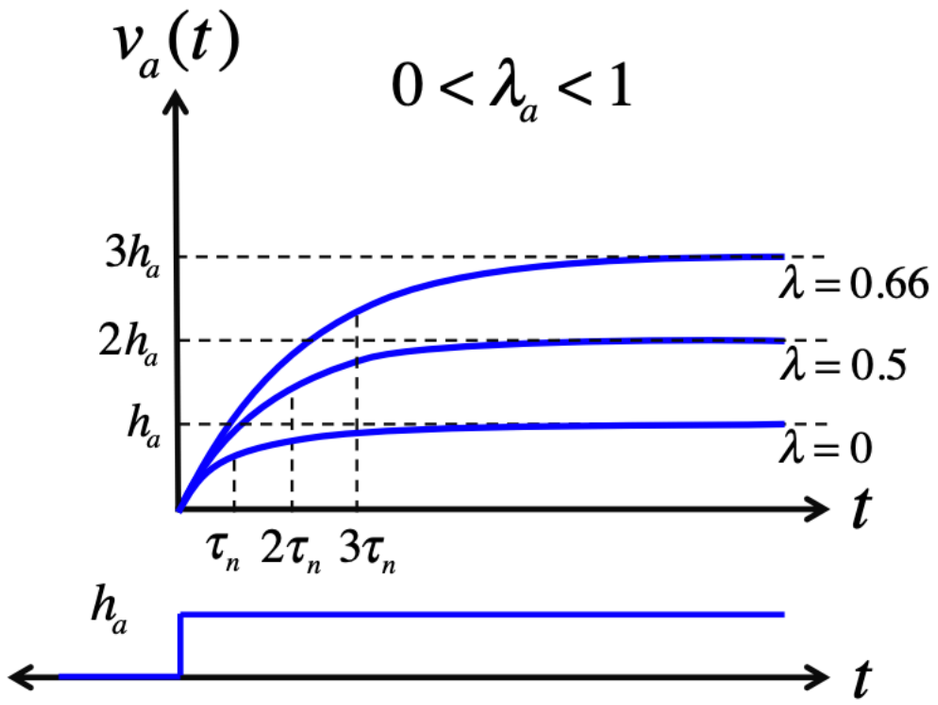
A solution

$$v_a(t) = v_{a,\infty} + (v_0 - v_{a,\infty})e^{-t/\tau_a}$$

Exponential relaxation

Recurrent networks

- Positive (excitatory) feedback acts to amplify the steady state activity of each neuron by an amount that depends on the strength of the feedback!



$$0 < \lambda_a < 1$$

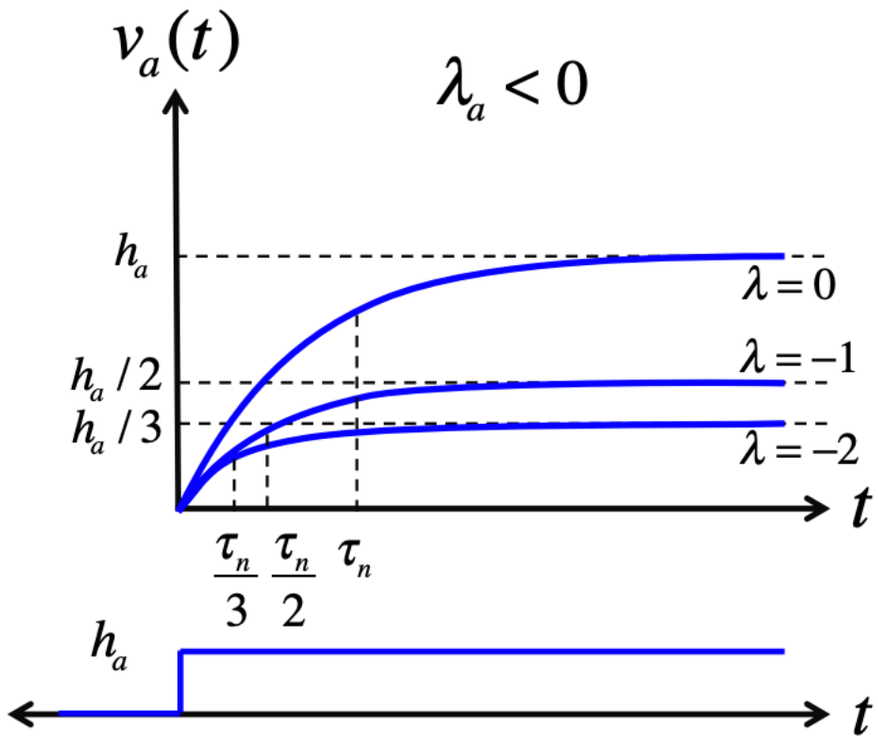
$$v_{a,\infty} = \frac{h_a}{1-\lambda_a}$$

$$\tau_a = \frac{\tau_n}{1-\lambda_a}$$

- Positive feedback amplifies the response and slows the time-constant of the response

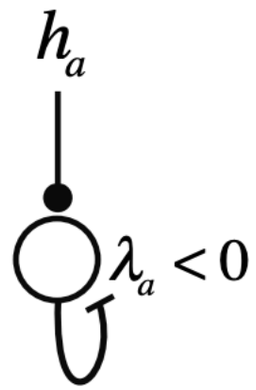
Recurrent networks

- Negative (inhibitory) feedback acts to suppress the steady state activity of a neuron by an amount that depends on the strength of the feedback.



$$v_{a,\infty} = \frac{h_a}{1-\lambda_a}$$

$$\tau_a = \frac{\tau_n}{1-\lambda_a}$$



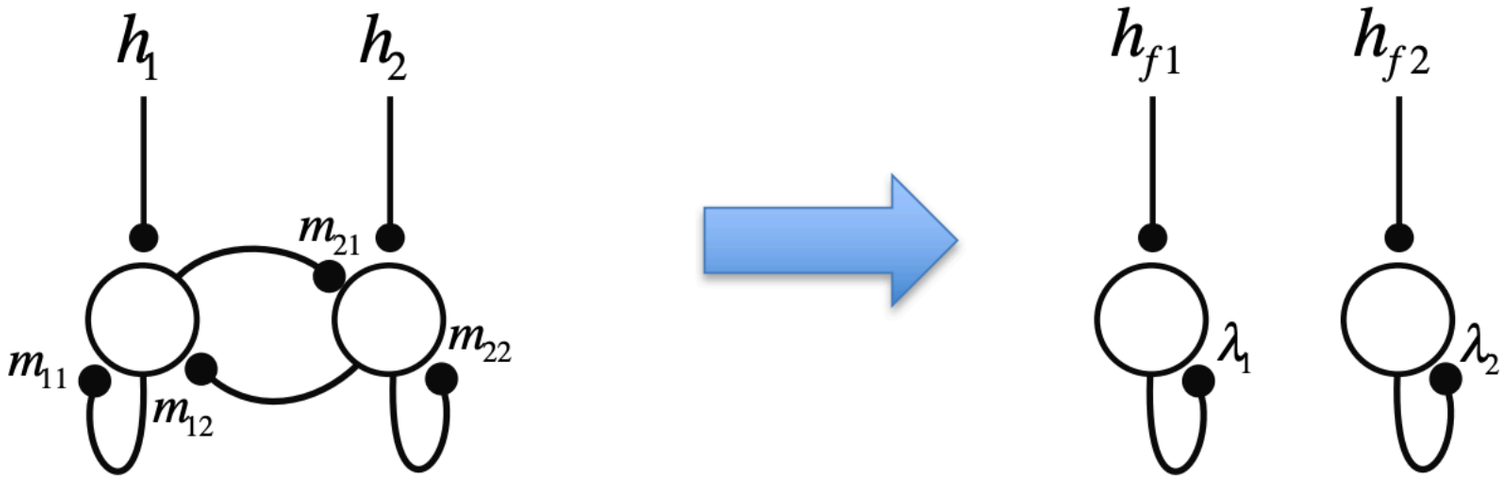
- Negative feedback suppresses the response and speeds the time-constant of the response

Recurrent networks

- Now let's look at the more general case of recurrent connectivity.
- We saw how the behavior of a recurrent network is extremely simple to describe if M is diagonal.
- So let's make M diagonal! Rewrite M as follows

$$M = \Phi \Lambda \Phi^T$$

where Λ is a diagonal matrix.



Recurrent networks

$$M\Phi = \Phi\Lambda \qquad M\,\hat{f}_\mu = \lambda_\mu \hat{f}_\mu$$

- If M is a symmetric matrix, then …
  - the eigenvalues are real
  - Φ is a rotation matrix. The eigenvectors give us an orthogonal basis set:

$$\hat{f}_i \cdot \hat{f}_j = \delta_{ij}$$

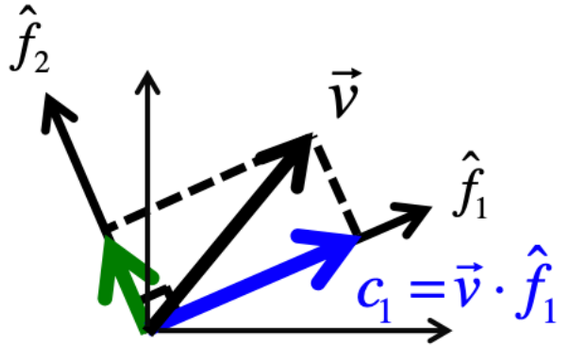
$$\Phi^T \Phi = I$$

Recurrent networks

- Now we are going to write our vector of output firing rates $\vec{v}$ in this new basis.

Project $\vec{v}$ onto each of the new basis vectors.

$$c_\alpha = \vec{v} \cdot \hat{f}_\alpha$$



- Express $\vec{v}$ as a linear combination of basis vectors

$$\vec{v} = c_1 \hat{f}_1 + c_2 \hat{f}_2 + c_2 \hat{f}_3 + \ldots$$

Recurrent networks

- Let's rewrite our network equation in this new basis set…

$$\tau_n \frac{d\vec{v}}{dt} = -\vec{v} + M\vec{v} + \vec{h} \qquad \vec{v} = \Phi\vec{c}$$

$$\tau_n \Phi \frac{d\vec{c}}{dt} = -\Phi\vec{c} + M\Phi\vec{c} + \vec{h}$$

- But we have chosen a basis set Φ such that

$$M\Phi = \Phi\Lambda$$

- Thus…

$$\tau_n \Phi \frac{d\vec{c}}{dt} = -\Phi\vec{c} + \Phi\Lambda\vec{c} + \vec{h}$$

Recurrent networks

$$\tau_n \Phi \frac{d\vec{c}}{dt} = -\Phi\vec{c} + \Phi\Lambda\vec{c} + \vec{h}$$

- Multiply both sides from the left by Φ^T

$$\tau_n \underbrace{\Phi^T\Phi}\frac{d\vec{c}}{dt} = -\underbrace{\Phi^T\Phi}\vec{c} + \underbrace{\Phi^T\Phi}\Lambda\vec{c} + \Phi^T\vec{h}$$

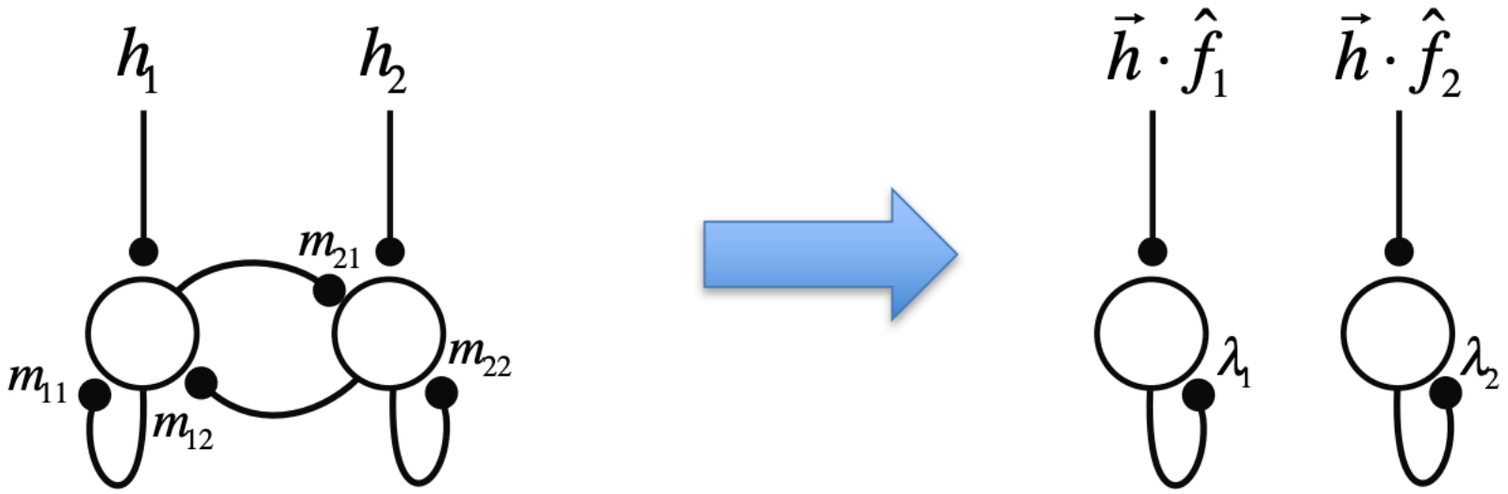
$$\tau_n \frac{d\vec{c}}{dt} = -\vec{c} + \Lambda\vec{c} + \vec{h}_f \qquad \vec{h}_f = \Phi^T\vec{h}$$

- But this is just our original network equation with a diagonal weight matrix!

31

Recurrent networks

- We can rewrite the equation for our network as n independent equations for n independent 'modes' of the network
- We can think of this transformation as making a new network with only autapses.



- The activities $c_\alpha(t)$ of our network modes represent activity of linear combinations of neurons in our original network

32

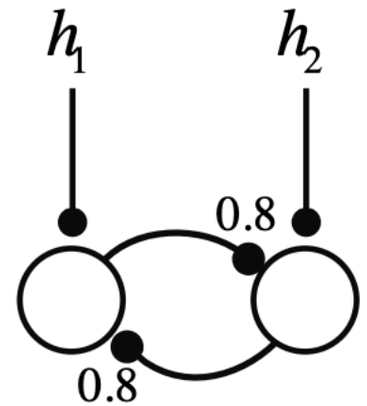
Recurrent networks

- Now let's look at a case where two output neurons are connected to each other by mutual excitation.

What is the weight matrix?

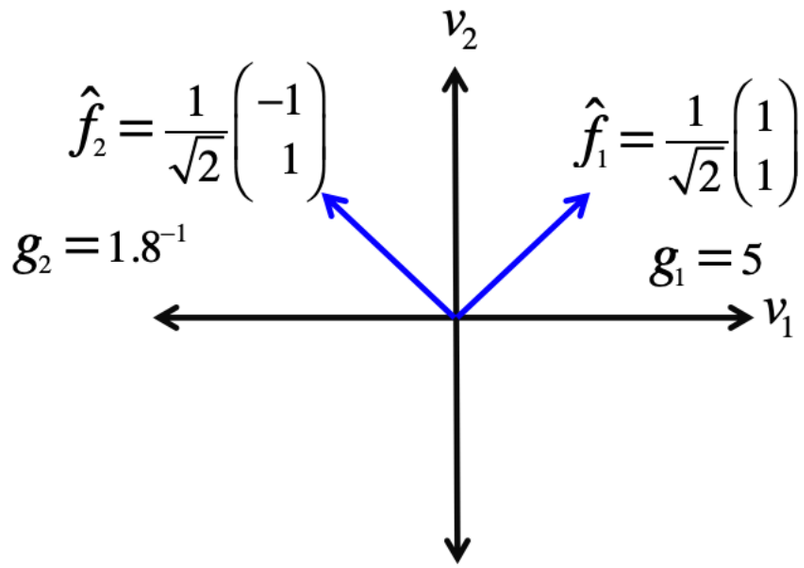
$$M = \begin{pmatrix} 0 & 0.8 \\ 0.8 & 0 \end{pmatrix}$$

$$M\Phi = \Phi\Lambda$$



$$\Phi = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

$$\Lambda = \begin{pmatrix} 0.8 & 0 \\ 0 & -0.8 \end{pmatrix}$$

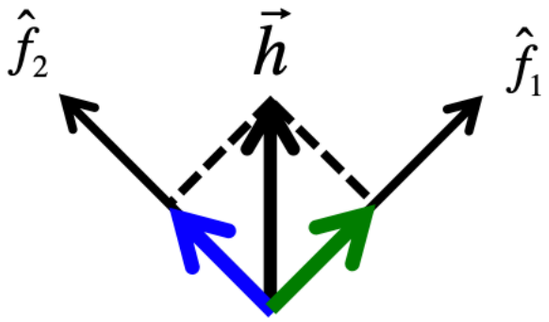


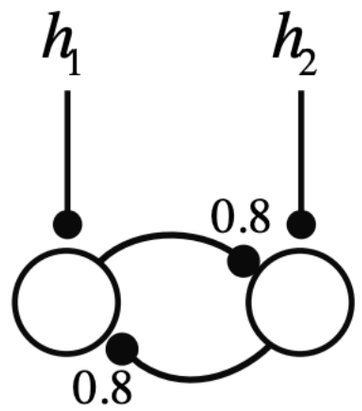
36

Recurrent networks

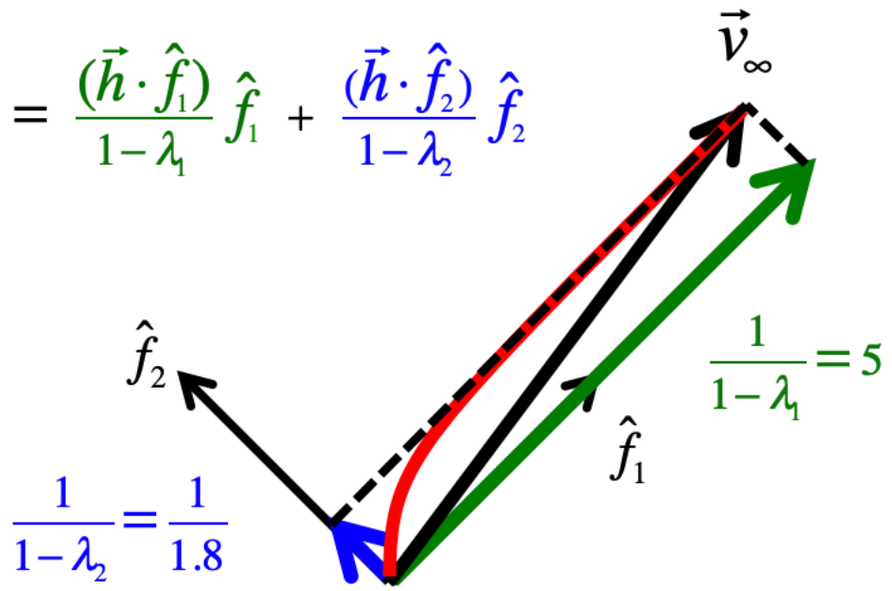
- If the input is not parallel to an eigenvector, we break the input into a component along each mode

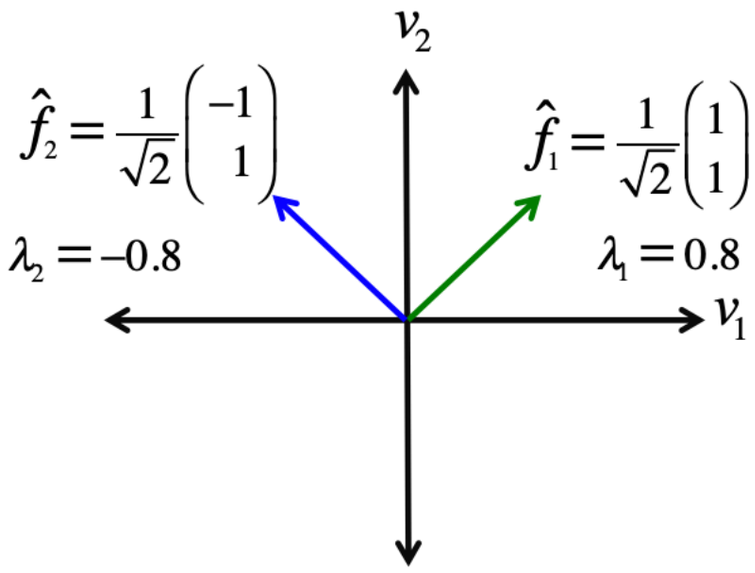
$$\vec{h} = (\vec{h}\cdot\hat{f}_1)\,\hat{f}_1 + (\vec{h}\cdot\hat{f}_2)\,\hat{f}_2$$





$$\vec{v}_\infty = \frac{(\vec{h}\cdot\hat{f}_1)}{1-\lambda_1}\,\hat{f}_1 + \frac{(\vec{h}\cdot\hat{f}_2)}{1-\lambda_2}\,\hat{f}_2$$





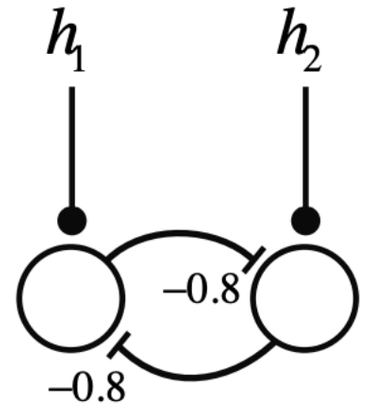
Recurrent networks

- Now let's look at a case where two output neurons are connected to each other by mutual inhibition.

What is the weight matrix?

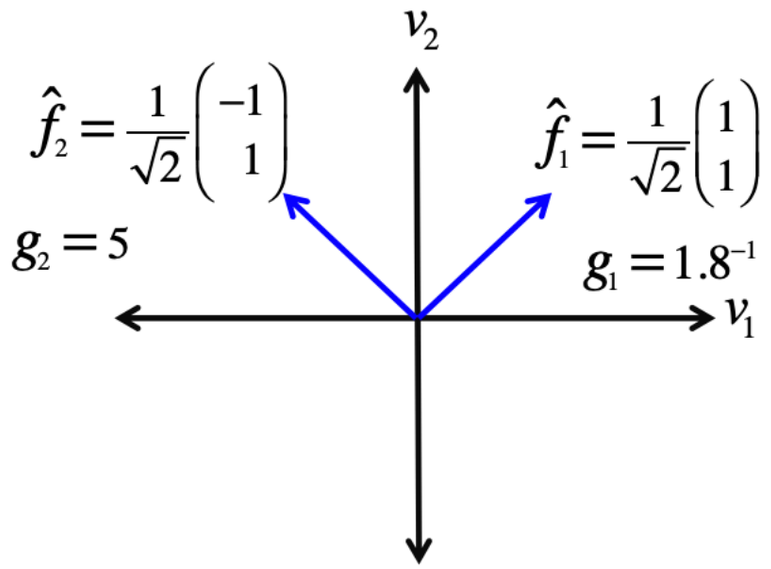
$$M = \begin{pmatrix} 0 & -0.8 \\ -0.8 & 0 \end{pmatrix}$$

$$M\Phi = \Phi\Lambda$$



$$\Phi = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

$$\Lambda = \begin{pmatrix} -0.8 & 0 \\ 0 & 0.8 \end{pmatrix}$$



40

Recurrent networks

- We have described the case where $\lambda < 1$.

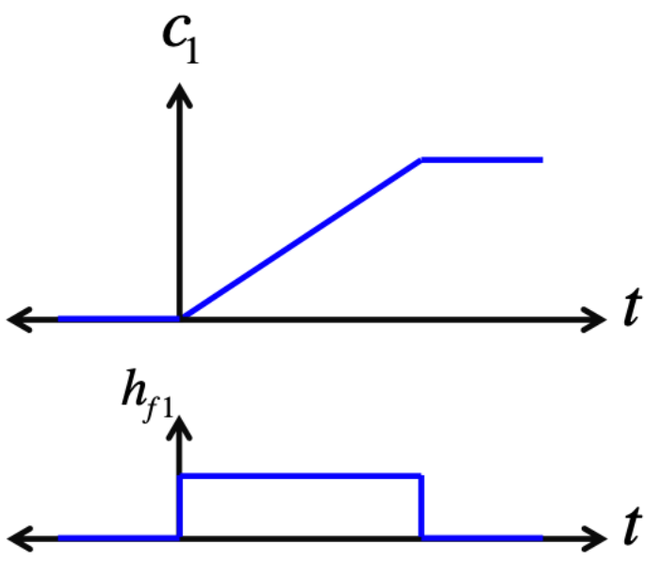
  What happens when $\lambda = 1$?

$$\tau_n \frac{dc_\alpha}{dt} = -\underbrace{(1-\lambda_\alpha)}_{0} c_\alpha + \hat{f}_\alpha \cdot \vec{h}(t)$$

$$\tau_n \frac{dc_1}{dt} = \hat{f}_1 \cdot \vec{h}(t) = h_{f1}(t)$$

$$c_1(t) = c_1(0) + \frac{1}{\tau_n} \int_0^t h_{f1}(\tau) d\tau$$

Integrator!



$$h_{f1}(t) = \hat{f}_1 \cdot \vec{h}(t)$$

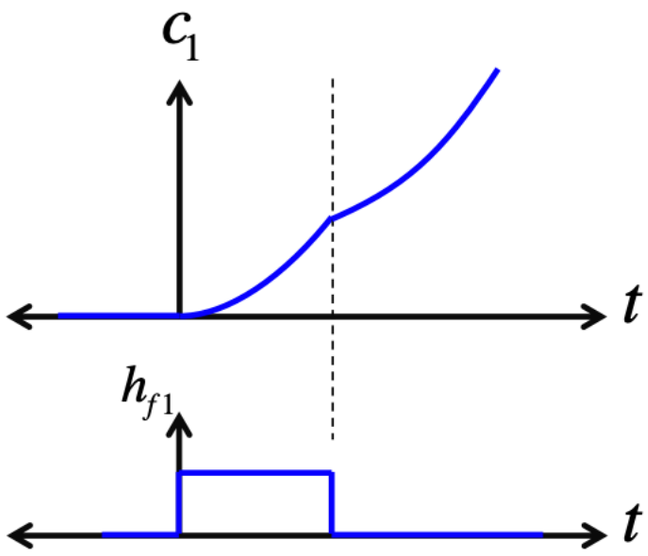
Recurrent networks

- What happens when $\lambda > 1$?

$$\tau_n \frac{dc_1}{dt} = -(1-\lambda_1)c_1 + \hat{f}_1 \cdot \vec{h}(t)$$

$$\tau_n \frac{dc_1}{dt} = \underbrace{(\lambda_1 - 1)}_{>0} c_1 + \hat{f}_1 \cdot \vec{h}(t)$$

Exponential growth!



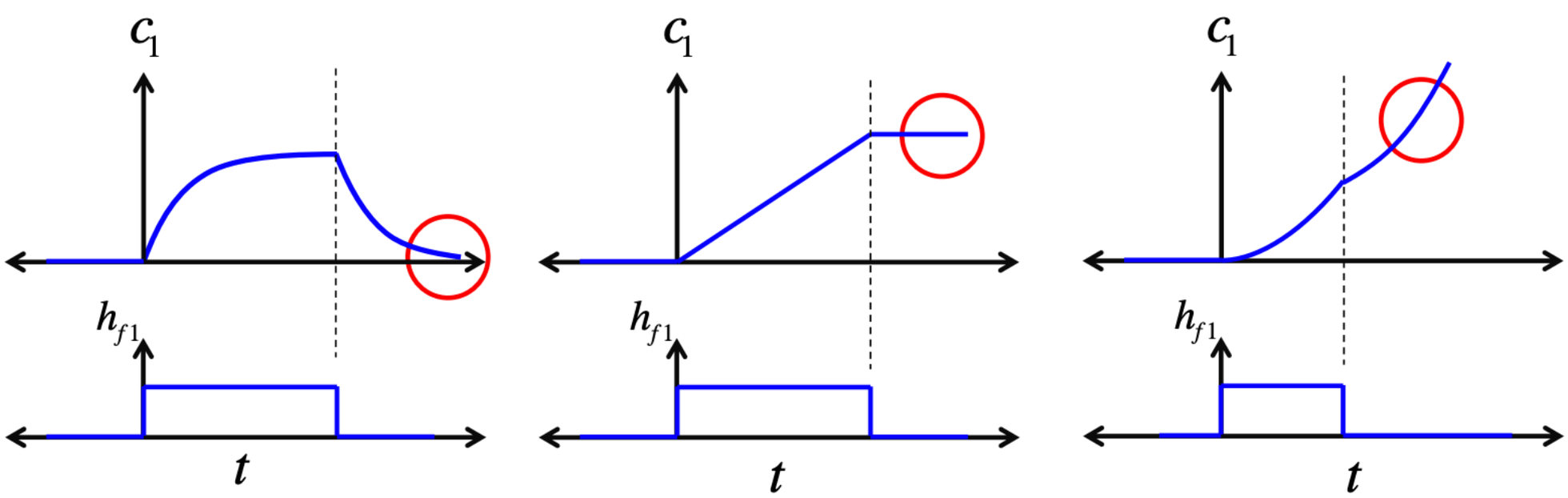
$$h_{f1}(t) = \hat{f}_1 \cdot \vec{h}(t)$$

44

Recurrent networks

- The behavior of the network depends critically on λ

$\lambda < 1$	$\lambda = 1$	$\lambda > 1$
Exponential relaxation	Integration	Exponential growth



With zero input…
relaxation back to zero

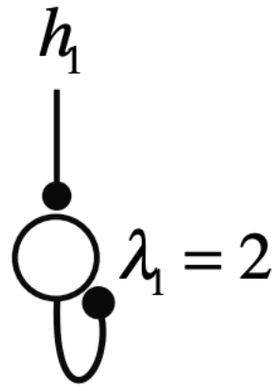
With zero input…
persistent activity!

MEMORY!

45

Recurrent networks

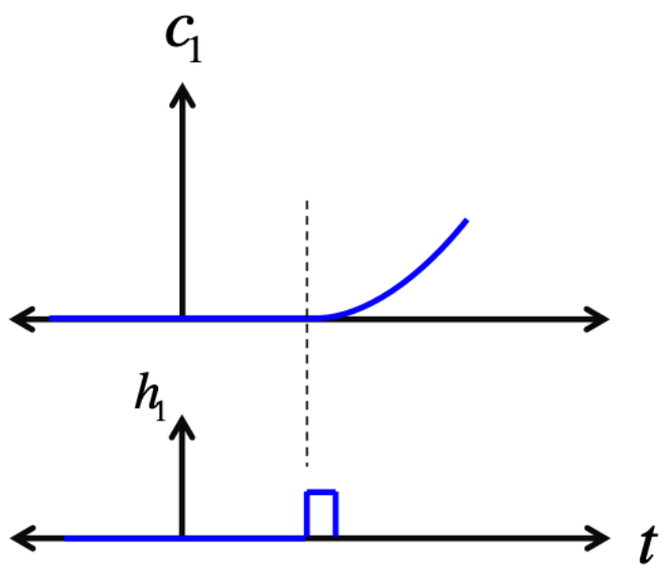
- Networks with $\lambda \geq 1$ have memory!

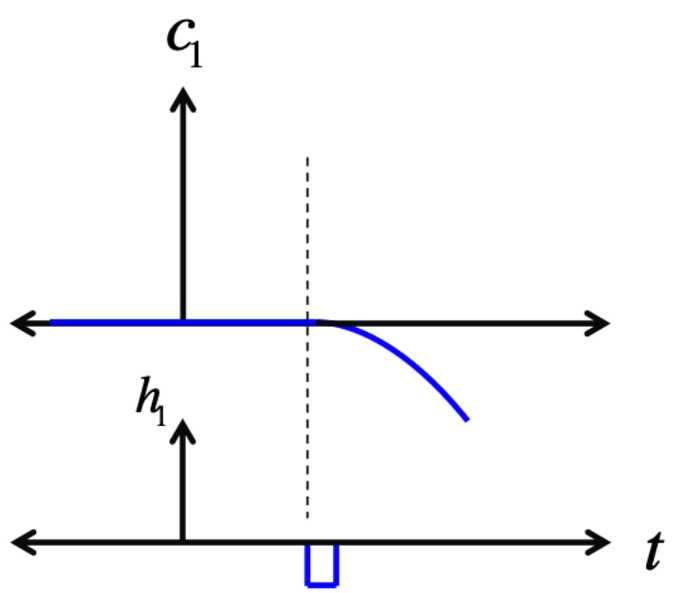


$$\tau_n \frac{dc_1}{dt} = (\lambda_1 - 1)c_1 + h_{f1}(t)$$

$$\tau_n \frac{dc_1}{dt} = c_1 \qquad c_1(t) = 0$$

- With zero input, zero is an 'unstable fixed point' of the network

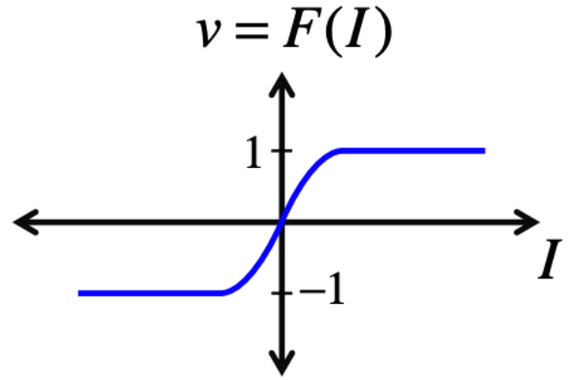


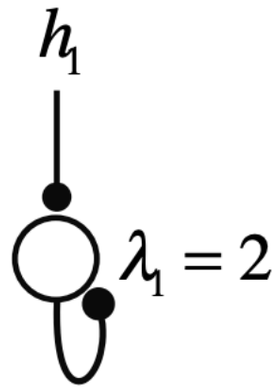


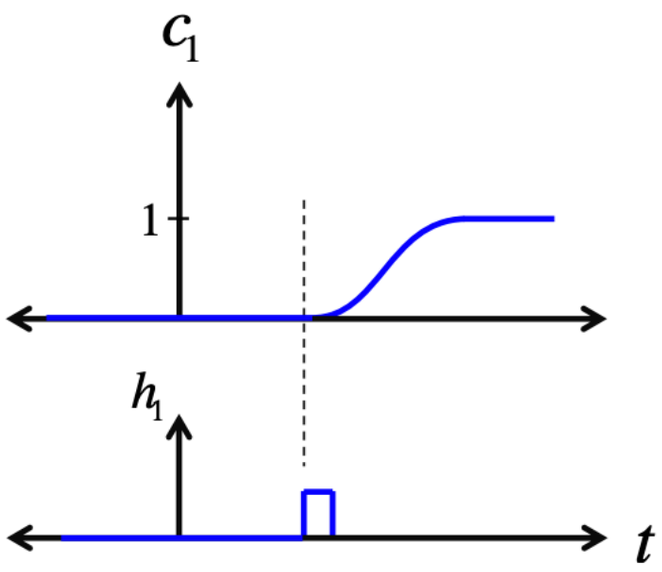
46

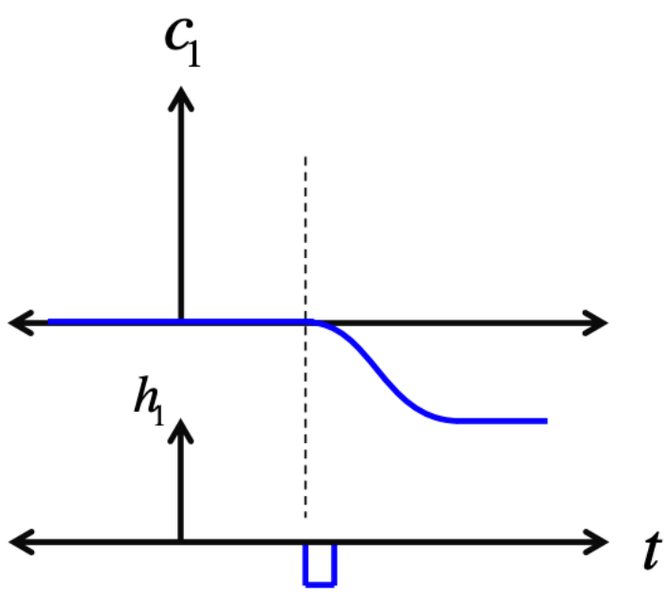
Recurrent networks

- Add a saturating activation function $F(x)$





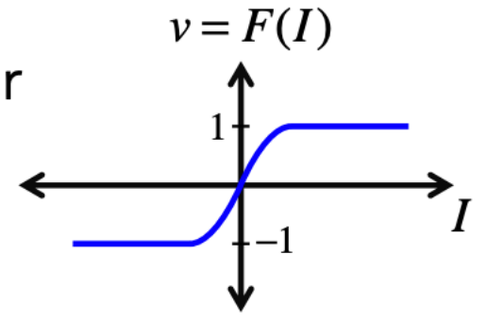


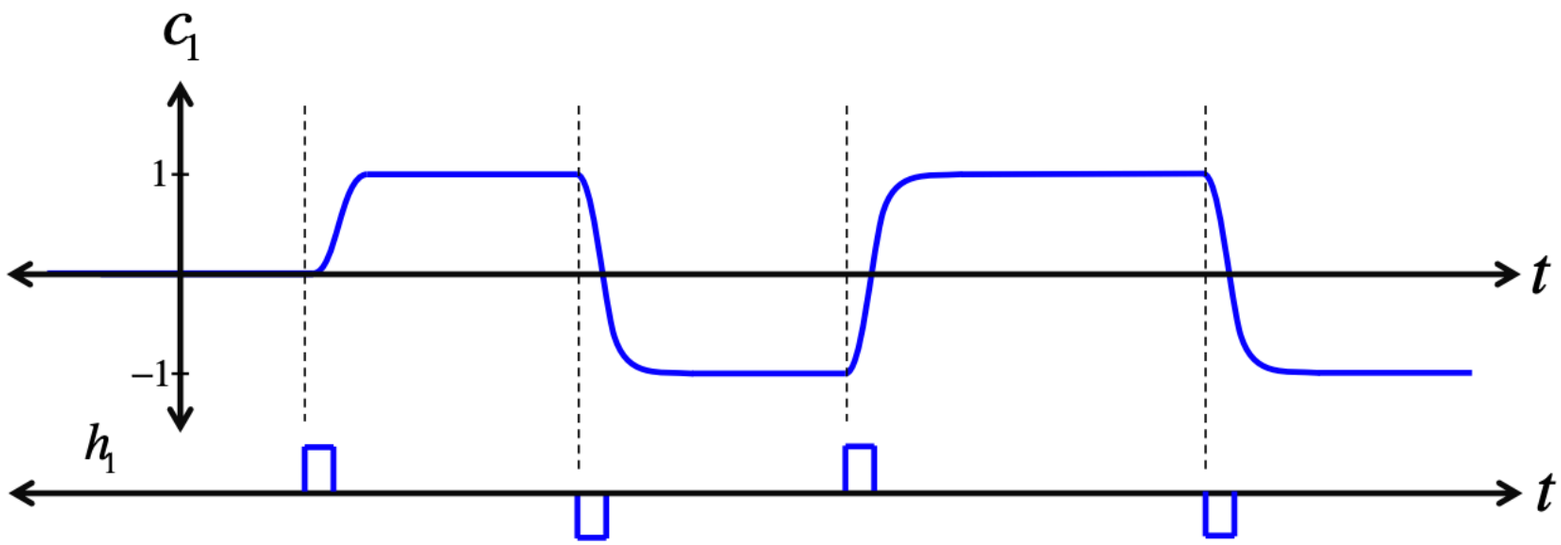


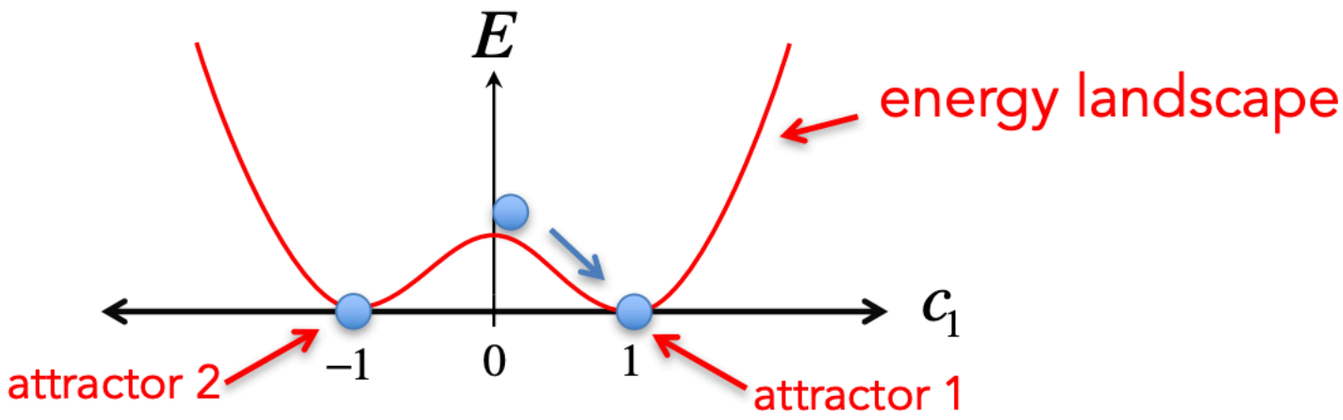
47

Recurrent networks

- Saturating activation function plus eigenvalues greater than 1 lead to stable states other than zero!

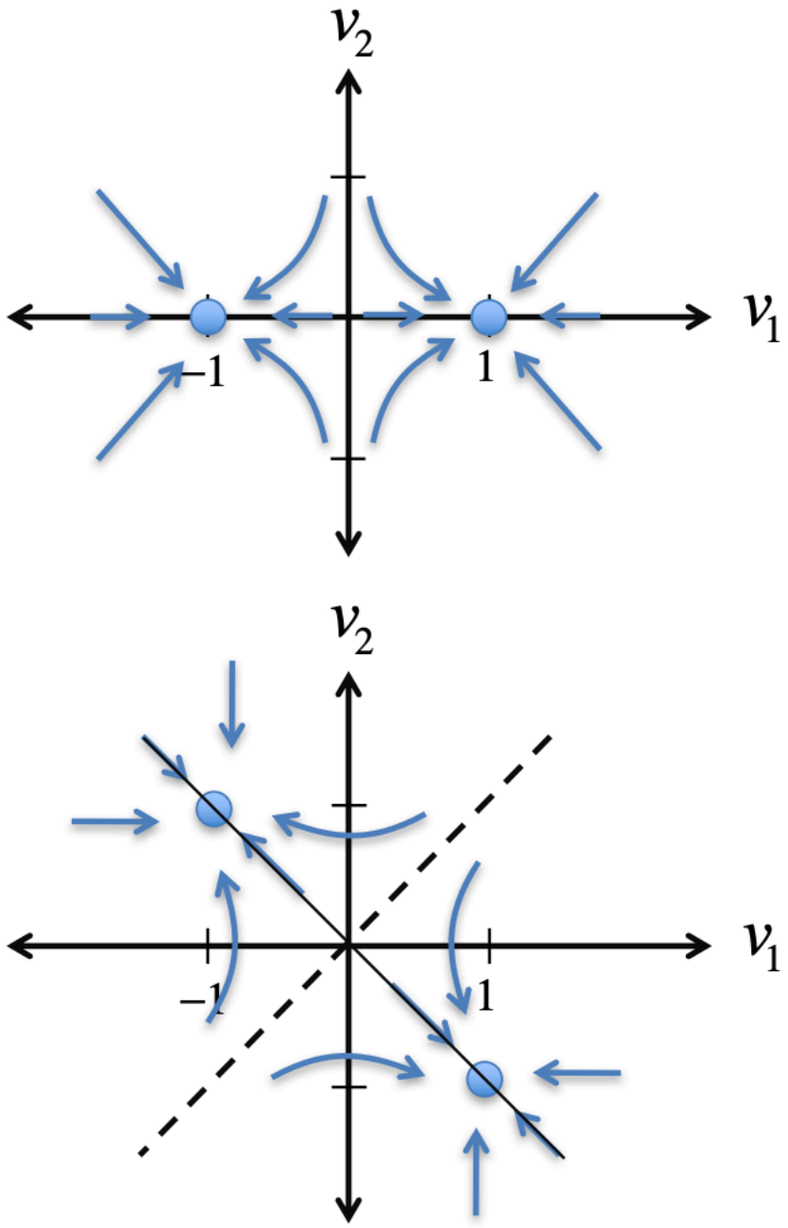


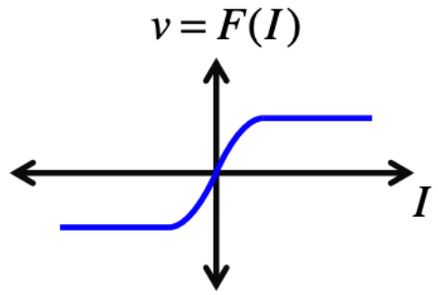


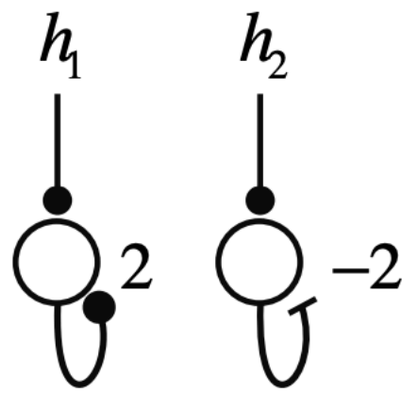


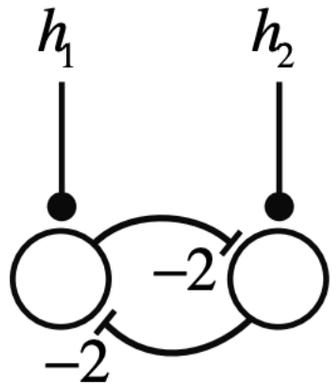
Recurrent networks

- Two-neuron network that has two attractors









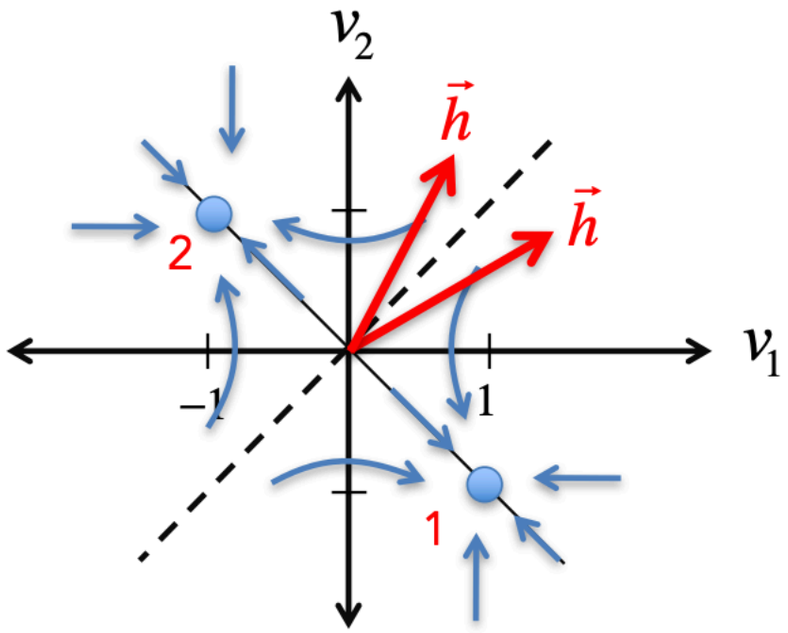
49

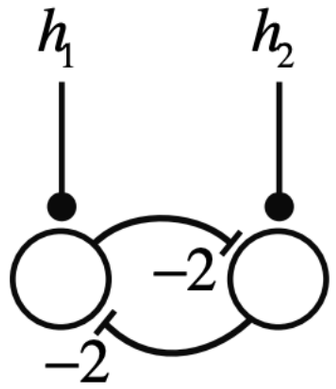
Winner-take-all network

- Implements decision making

  Network will remain in attractor 1 if $h_1 > h_2$

  Network will remain in attractor 2 if $h_2 > h_1$





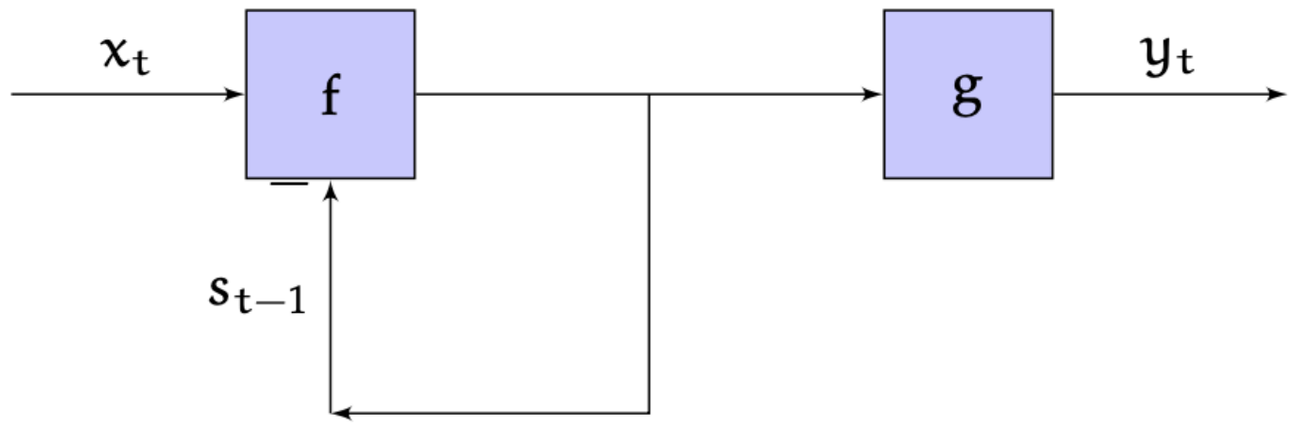
51

Back-propagation through time

Recall that the basic operation of the state machine is to start with some state s_0, then iteratively compute for $t \geqslant 1$::

$$s_t = f(s_{t-1}, x_t)$$
$$y_t = g(s_t)$$

as illustrated in the diagram below (remembering that there needs to be a delay on the feedback loop):



So, given a sequence of inputs $x_1, x_2, \ldots$ the machine generates a sequence of outputs

$$\underbrace{g(f(s_0, x_1))}_{y_1}, \underbrace{g(f(f(s_0, x_1), x_2,))}_{y_2}, \ldots \quad .$$

A *recurrent neural network* is a state machine with neural networks constituting functions f and g:

$$s_t = f_1\left(W^{sx}x_t + W^{ss}s_{t-1} + W_0^{ss}\right)$$
$$y_t = f_2\left(W^{O}s_t + W_0^{O}\right) \quad .$$

Sequence-to-sequence RNN

A training set has the form $\left[\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(q)}, y^{(q)}\right)\right]$, where

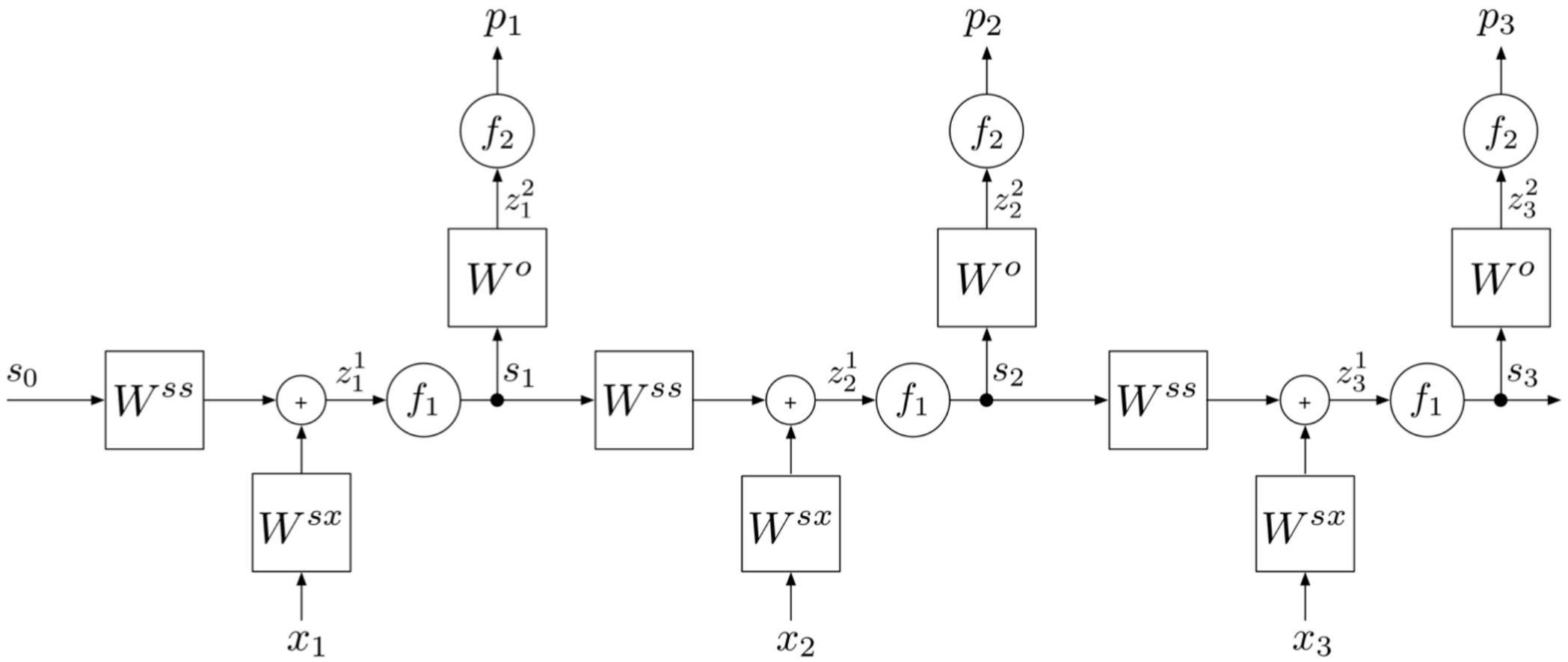
- $x^{(i)}$ and $y^{(i)}$ are length $n^{(i)}$ sequences;
- sequences in the *same pair* are the same length; and sequences in different pairs may have different lengths.

$$J(\theta) = \sum_{i=1}^{q} \text{Loss}_{\text{seq}}\left(\text{RNN}(x^{(i)};\theta), y^{(i)}\right) \quad , \qquad \text{Loss}_{\text{seq}}\left(p^{(i)}, y^{(i)}\right) = \sum_{t=1}^{n^{(i)}} \text{Loss}_{\text{elt}}\left(p_t^{(i)}, y_t^{(i)}\right) \quad .$$

The BPTT process goes like this:

(1) Sample a training pair of sequences (x, y); let their length be n.

(2) "Unroll" the RNN to be length n (picture for $n = 3$ below), and initialize s_0:



Now, we can see our problem as one of performing what is almost an ordinary back-propagation training procedure in a feed-forward neural network, but with the difference that the weight matrices are shared among the layers. In many ways, this is similar to what ends up happening in a convolutional network, except in the conv-net, the weights are re-used spatially, and here, they are re-used temporally.

(3) Do the *forward pass*, to compute the predicted output sequence p:

$$
\begin{aligned}
z_t^1 &= W^{sx} x_t + W^{ss} s_{t-1} + W_0^{ss} \\
s_t &= f_1(z_t^1) \\
z_t^2 &= W^{O} s_t + W_0^{O} \\
p_t &= f_2(z_t^2)
\end{aligned}
$$

(4) Do *backward pass* to compute the gradients. For both W^{ss} and W^{sx} we need to find

$$
\frac{dL_{seq}}{dW} = \sum_{u=1}^{n} \frac{dL_u}{dW}
$$

obtaining

$$
\begin{aligned}
\frac{dL_{seq}}{dW^{ss}} += \frac{\partial F_{t-1}}{\partial W^{ss}} &= \frac{\partial z_t^1}{\partial W^{ss}} \frac{\partial s_t}{\partial z_t^1} \frac{\partial F_{t-1}}{\partial s_t} \\
\frac{dL_{seq}}{dW^{sx}} += \frac{\partial F_{t-1}}{\partial W^{sx}} &= \frac{\partial z_t^1}{\partial W^{sx}} \frac{\partial s_t}{\partial z_t^1} \frac{\partial F_{t-1}}{\partial s_t}
\end{aligned}
\tag{12.1}
$$

We can handle W^O separately; it's easier because it does not effect future losses in the way that the other weight matrices do:

$$
\frac{\partial L_{seq}}{\partial W^{O}} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial W^{O}} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial z_t^2} \cdot \frac{\partial z_t^2}{\partial W^{O}}
$$

PRATICAL TASK

1. Explore the notebook RNN.ipynb
2. Change the “input sequence” to something useful…