Hebbian learning

NSC3270 / NSC5270
Computational Neuroscience

Tu/Th 9:35-10:50am
Wilson 316

Professor Thomas Palmeri
Professor Sean Polyn

ROAD MAP (partial)

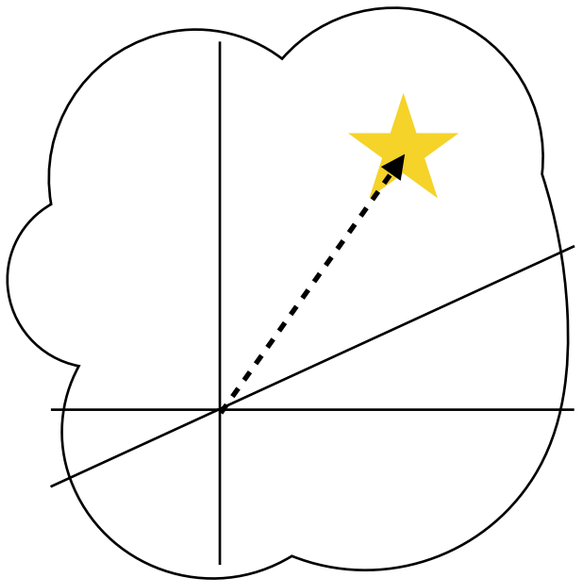
- Basics of Computational Neuroscience
- A simple 1-layer classification network
- Hebbian learning
- Error-driven learning
- The backpropagation algorithm
- Autoassociative networks

Hebbian learning

- Vector spaces
- The simplest Hebbian learning rule
- Weight normalization & Oja’s learning rule
- Activity normalization & inhibition
- Learning a running average of the environment

Vector spaces

An activation vector in a neural network defines a point in a vector space. The space itself can be thought of as the set of all possible activation states of that layer.

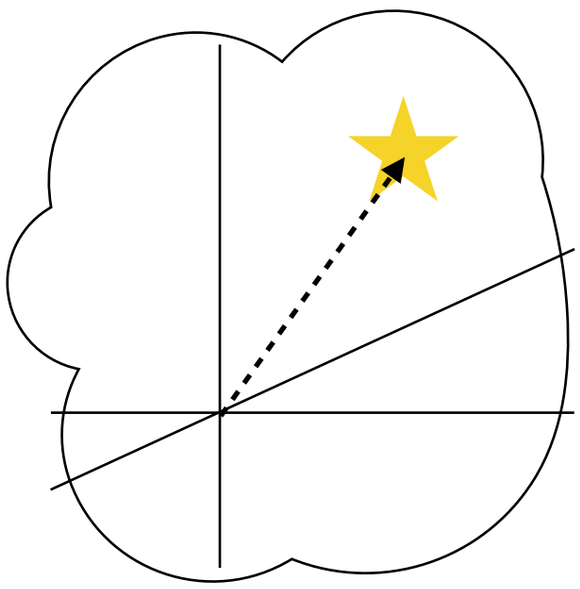


F, a 3-d space

Neural interpretation of vector spaces

When you create a model in computational neuroscience, you have to decide what kind of vector space you want to use.

Not just how many dimensions there should be, but also what numbers are allowable as activation states in that space!

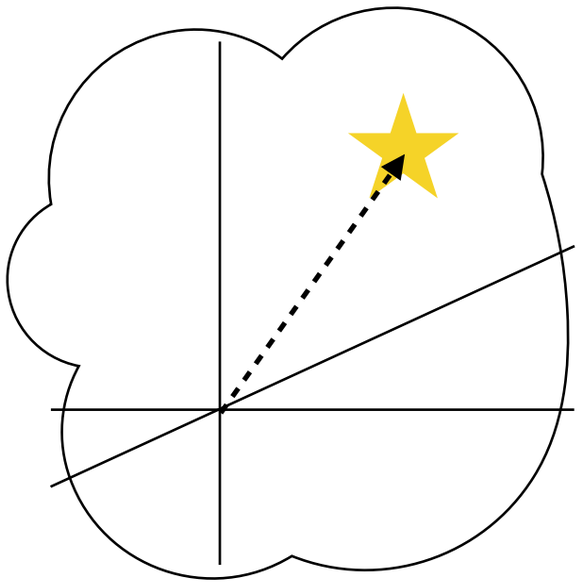


F, a 3-d space

There usually won't be a right answer, it will depend what kind of system you want to simulate.

Cognitive interpretation of vector spaces

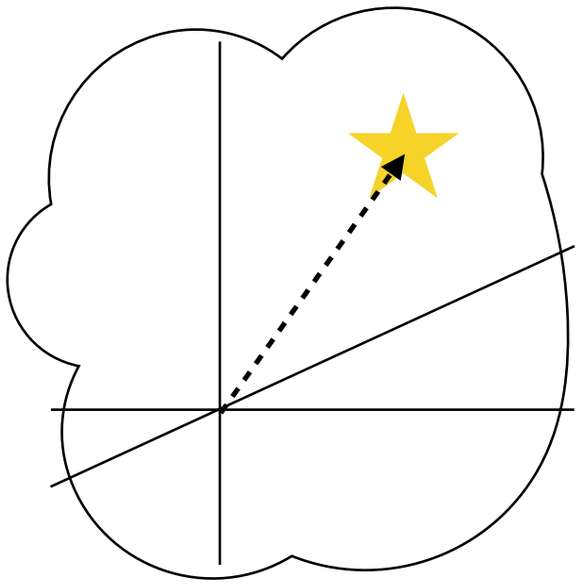
From a cognitive perspective, what sorts of information might we want to represent in a vector space?



F, a 3-d space

Cognitive interpretation of vector spaces

From a cognitive perspective, what sorts of information might we want to represent in a vector space?



F, a 3-d space

Perceptual (visual, auditory, olfactory, somatosensory)

Conceptual

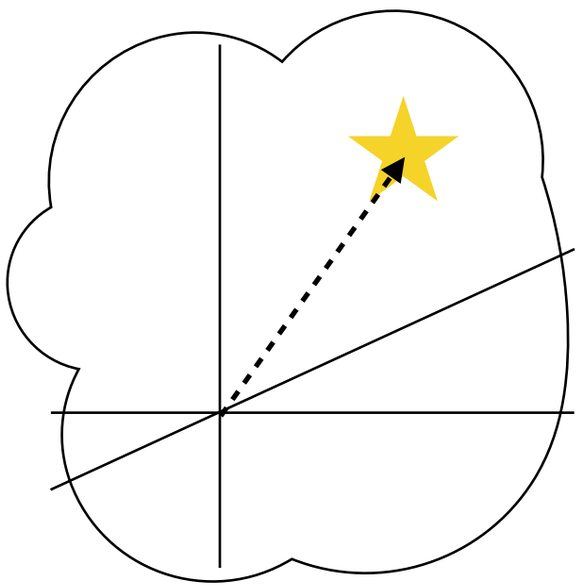
Linguistic (phonological, orthographic, lexical, motor)

Executive

Our neural model informs our vector space

A rate coding model could have activation values ranging across zero and the positive real numbers. $\mathbf{f} \in \mathbb{R}^N_{\geq 0}$

If zero is defined as the mean firing rate for a neuron, then positive and negative reals could reflect deviations from the mean rate. $\mathbf{f} \in \mathbb{R}^N$

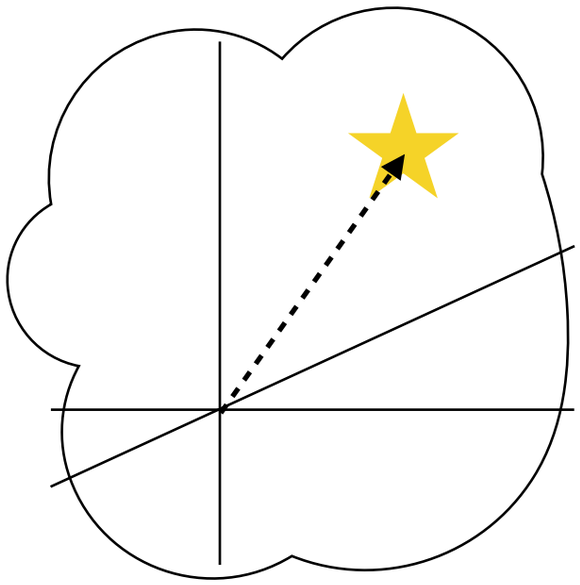


F, an N-d space

A system where each unit represents a population of excitatory and inhibitory cells could also range over the positive and negative real numbers, but now positive vs negative could reflect whether the excitatory or the inhibitory cells are more influential at that moment.

Our neural model informs our vector space

In HW#3 the vector space of the input units was 784 dimensional, with allowable values being integers from 0 to 255 (corresponding to grayscale pixel values in a 28x28 image)



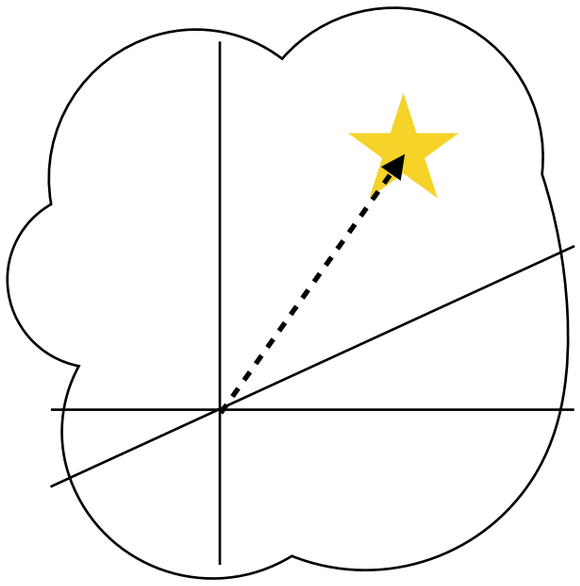
F, an N-d space

In the logical AND, OR, and XOR examples, the input layer was a 2d binary vector space, and the output was a 1d binary vector space.

Our neural model informs our vector space

If we want to investigate a simplified spiking network, activation values could be binary, 0 or 1.

A Hopfield network uses a different binary formalism, where activation values can be -1 or +1.



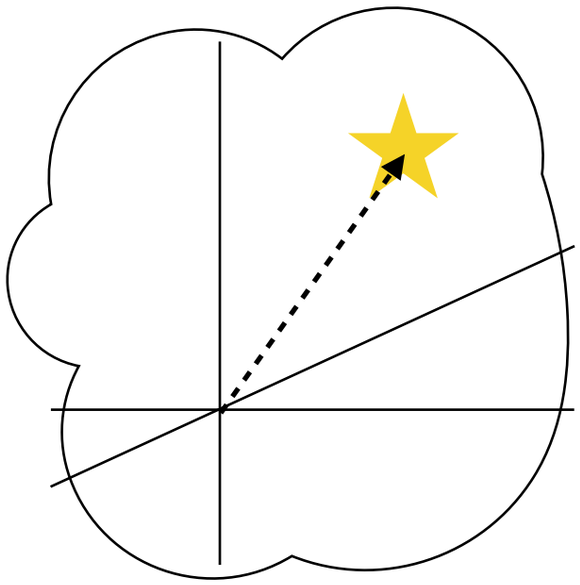
F, an N-d space

In these spaces, particular activation states can be thought of as corners of a hypercube.

With 1-d there are 2 possible states. How about with 2-d, 3-d, 4-d, N-d?

Our neural model informs our vector space

Fun fact: A 300-dimensional binary code (a string of 0s and 1s, or -1s and +1s) has 2^{300} possible states. That is enough states to give every particle in the observable universe its own unique code.



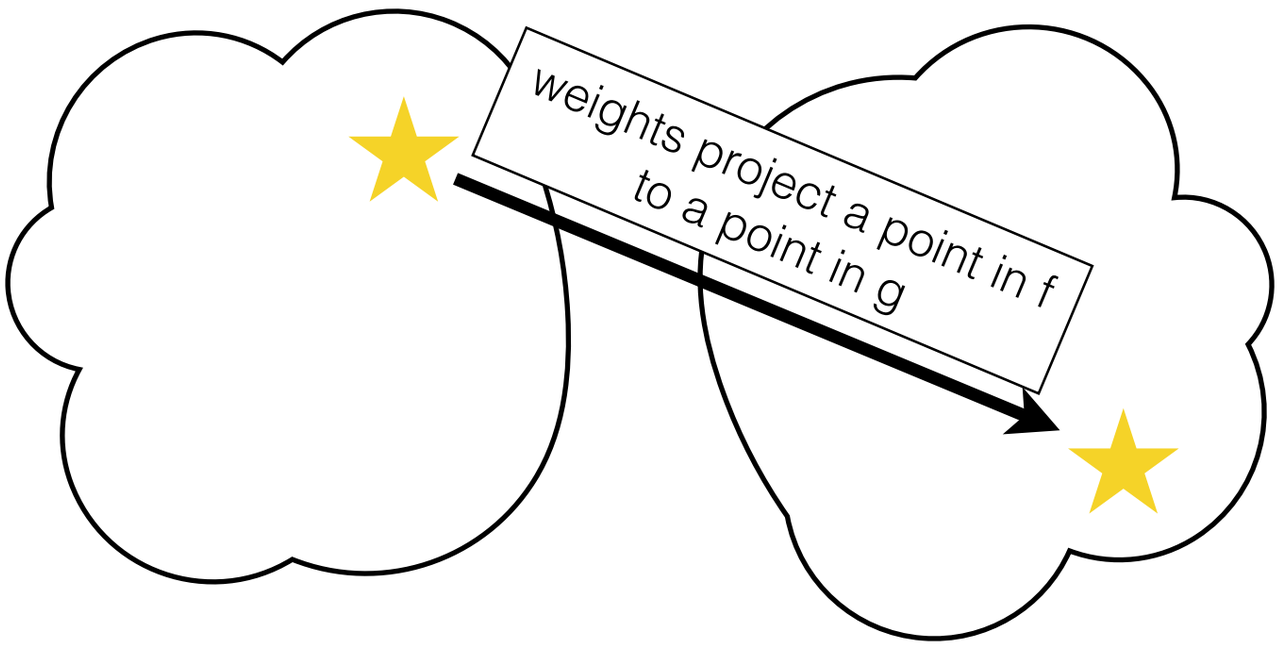
F, an N-d space

Transformation

What is a weight matrix doing?

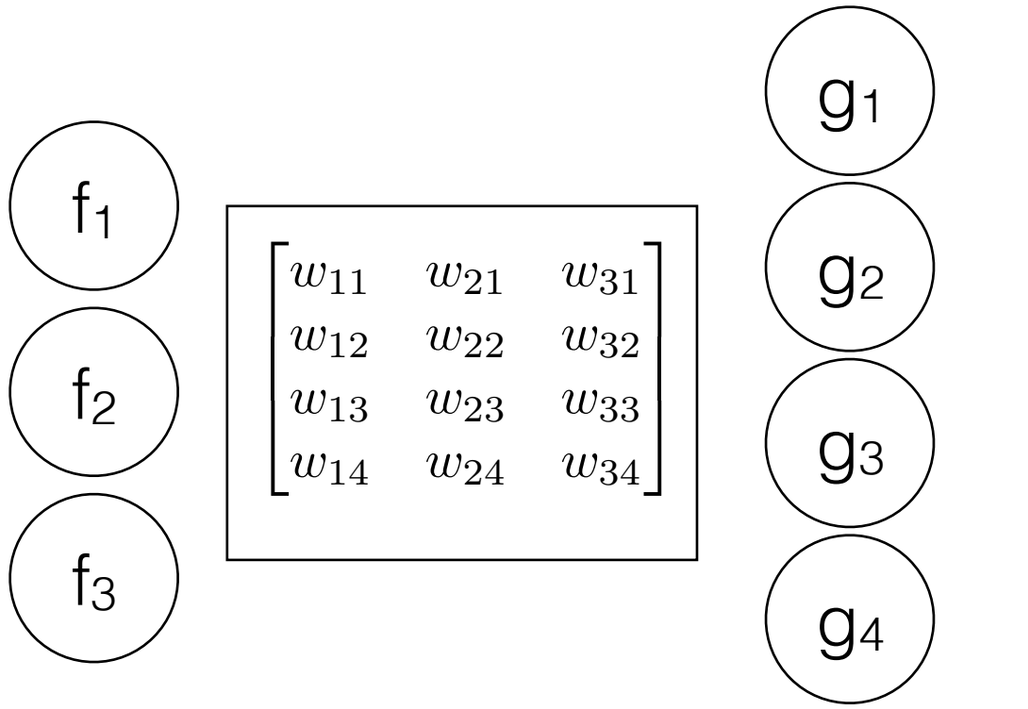
A given weight matrix transforms a pattern in the input space to a pattern in the output space.

The weight matrix defines a mapping between the two spaces.



f lives in
a 3-d space

g lives in
a 4-d space



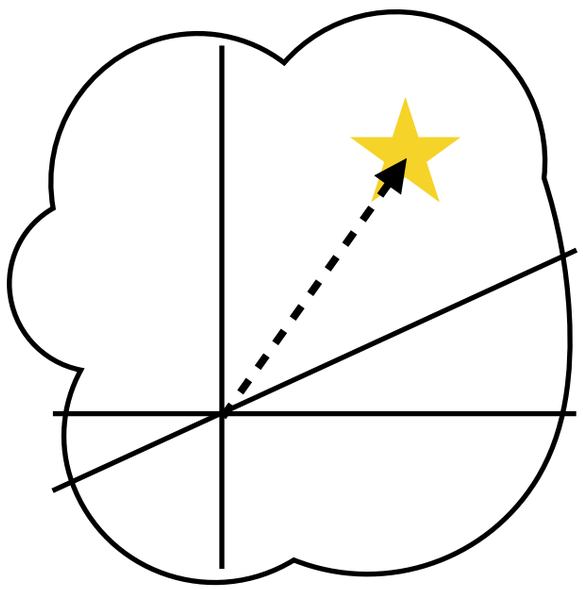
layer (1)

layer (2)

Vector spaces

Summary: An activation vector defines a point in a vector space. The space itself can be thought of as the set of all possible activation states of that layer.

An activation vector can also be thought of as a ray, a line going from the origin to that particular point.



F, a 3-d space

What is learning?

Learning as synaptic modification

Forming receptive fields

Association formation

Learning as creating mappings between vector spaces

Getting to know:

Donald Olding Hebb (1904-1985)

Graduate student at McGill, then at Harvard, working with Karl Lashley.

Postdoc in 1937 with Wilder Penfield. Prof at Queens Univ for 3 years, then did a second postdoc at the Yerkes National Lab in Atlanta (when Lashley became director). Started working on his book, *The Organization of Behavior*, while there.

1948 became a Professor at McGill, and then chair of the department. He was involved in training many great scientists, like Brenda Milner, Lynn Nadel, Michael Posner, & Mort Mishkin.

Donald Hebb's big ideas

Synaptic plasticity

Cell assemblies

Phase sequences

Neurophysiology - vocab corner

LTP: Long-term potentiation

Tetanus: rapidly repeated stimulation

EPSP: excitatory post-synaptic potential

IPSP: inhibitory post-synaptic potential

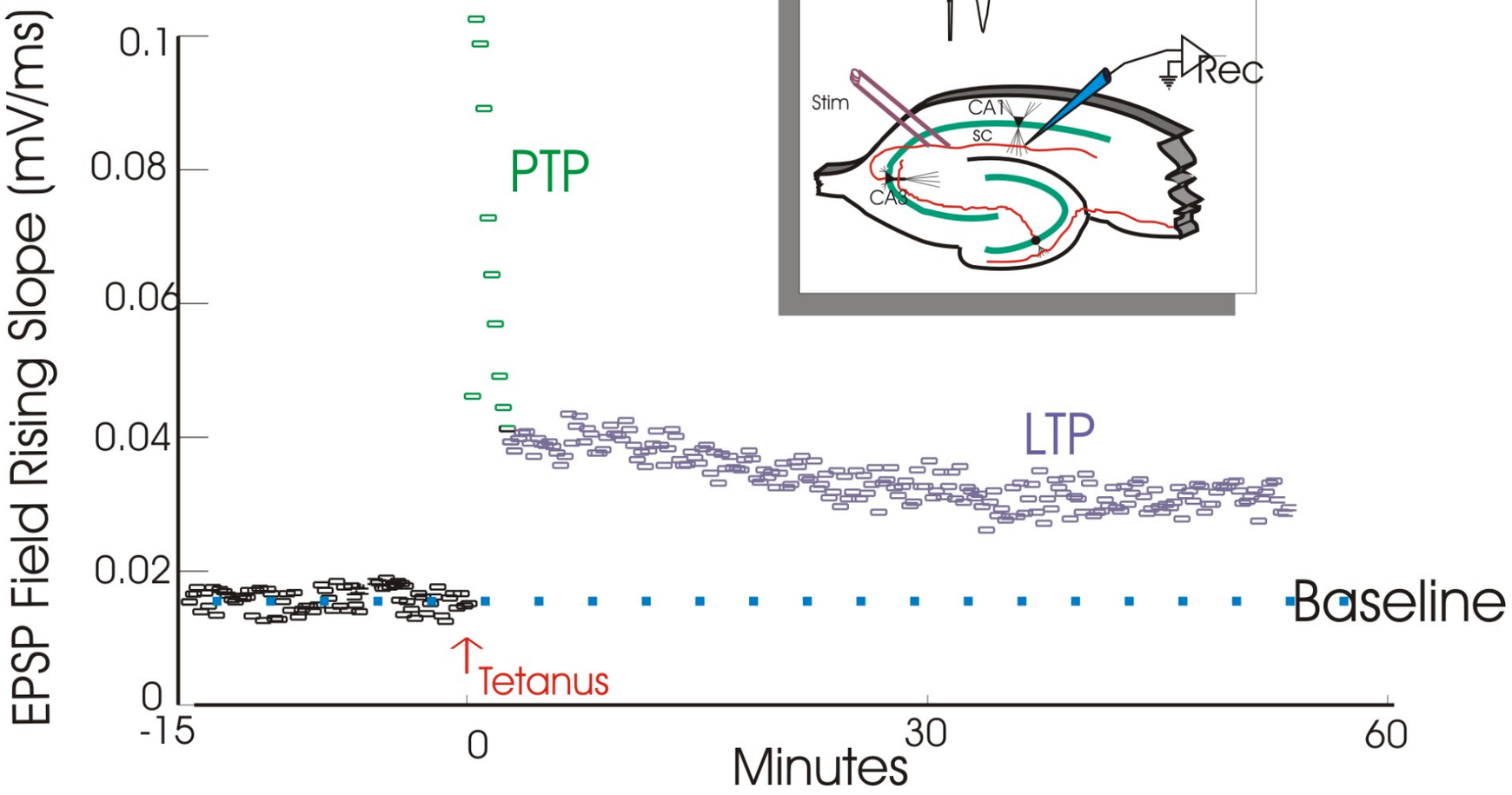
PTP: post-tetanic potential

Neurophysiology

Long-term potentiation (LTP)

A tetanus (rapidly repeated stimulation) that activates both pre- and post-synaptic neurons causes a given pre-synaptic input to produce a larger EPSP. The input is **potentiated**. This potentiation is specific to the synapses coming from the active pre-synaptic neurons (it doesn't spread to all synapses touching the post-synaptic neuron).

Neurophysiology



baseline before tetanus

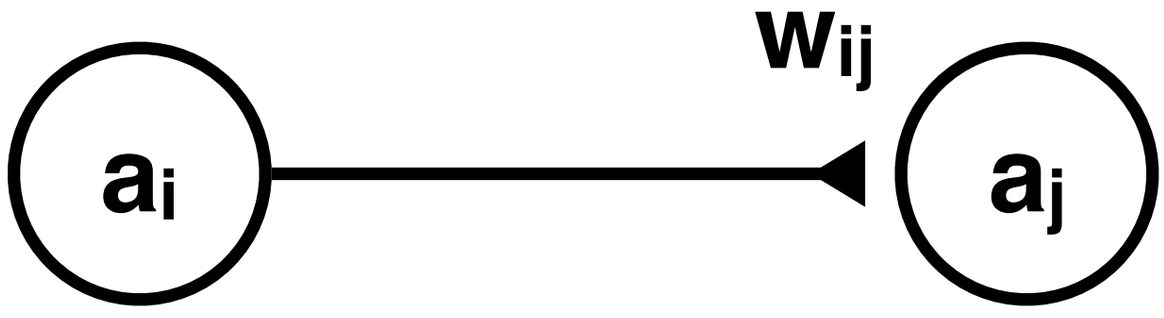
Neurophysiology

Long-term depression (LTD)

A slow or weak tetanus that doesn't cause the post-synaptic neuron to activate as strongly causes the same pre-synaptic input to produce a smaller EPSP.

Inhibitory inputs are also plastic: IPSPs can get potentiated or depressed.

The simplest Hebbian learning rule



$$\Delta W = \epsilon a_i a_j$$

epsilon / learning rate controls the step size of weight change

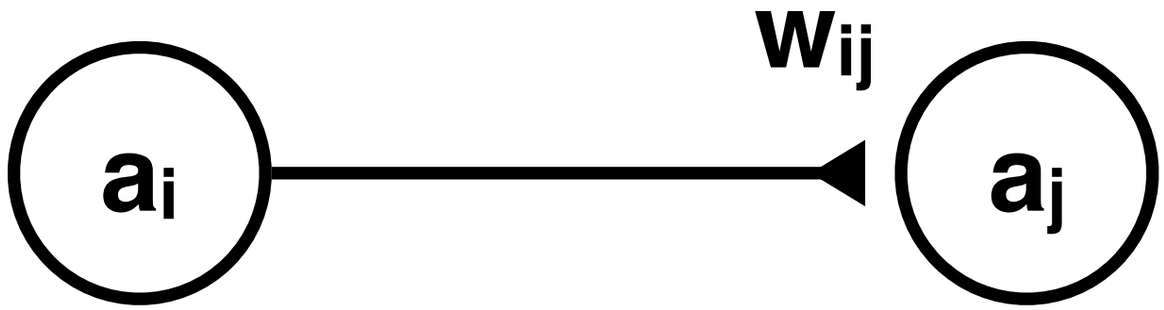
```
dW = lrate * act[i] * act[j]

W = W + dW
```

learning rate (traditional value is 0.05)

The simplest Hebbian learning rule

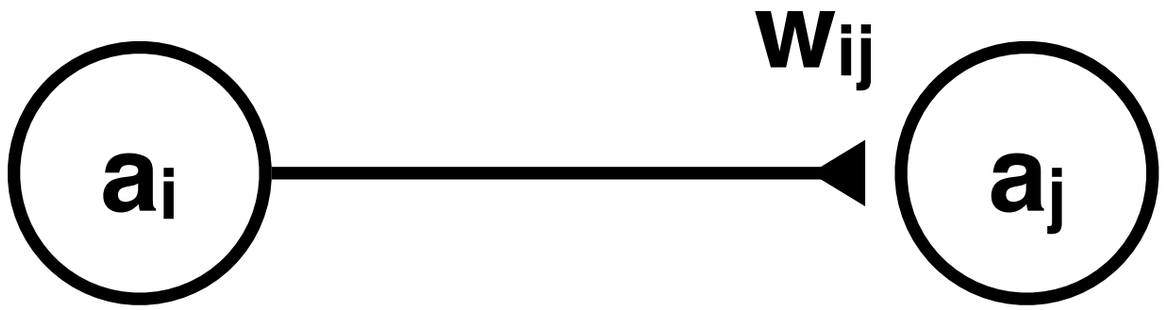
```
lrate = 0.05
W = 0.0
act1 = r.rand() # uniform, 0 to 1
act2 = r.rand()
dW = lrate * act1 * act2
W = W + dW
```

Consequences of a multiplicative learning rule?



$$\Delta W = \epsilon a_i a_j$$

Consequences of a multiplicative learning rule?



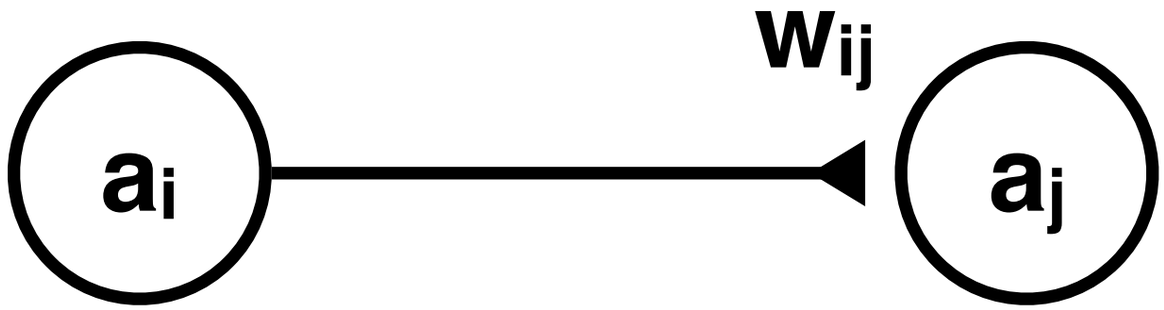
$$\Delta W = \epsilon a_i a_j$$

No weight change if either unit is inactive (if a_i or a_j == 0).

If a_i and a_j have the same sign, weight change will be positive.

If a_i and a_j have different signs, weight change will be negative.

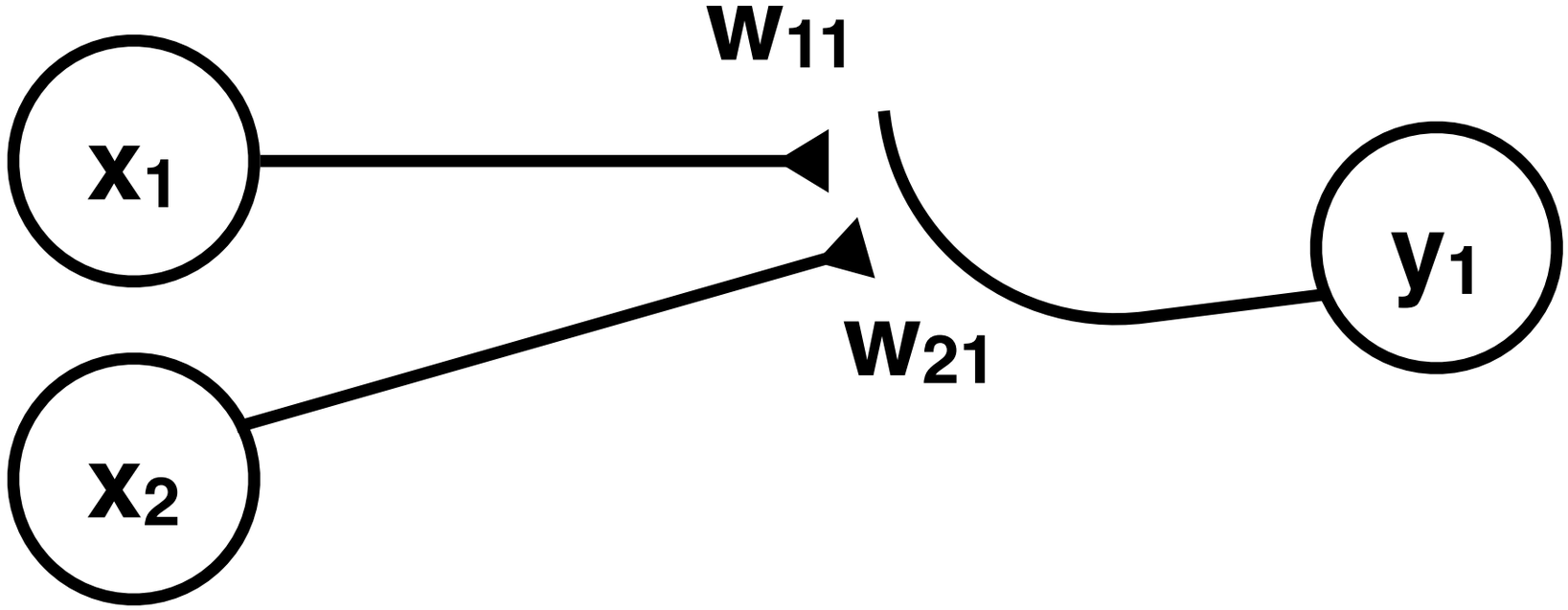
Consequences of a multiplicative learning rule?



$$\Delta W = \epsilon a_i a_j$$

A problem? W is unbounded! If a_i and a_j are correlated, the weights will steadily approach positive or negative infinity.

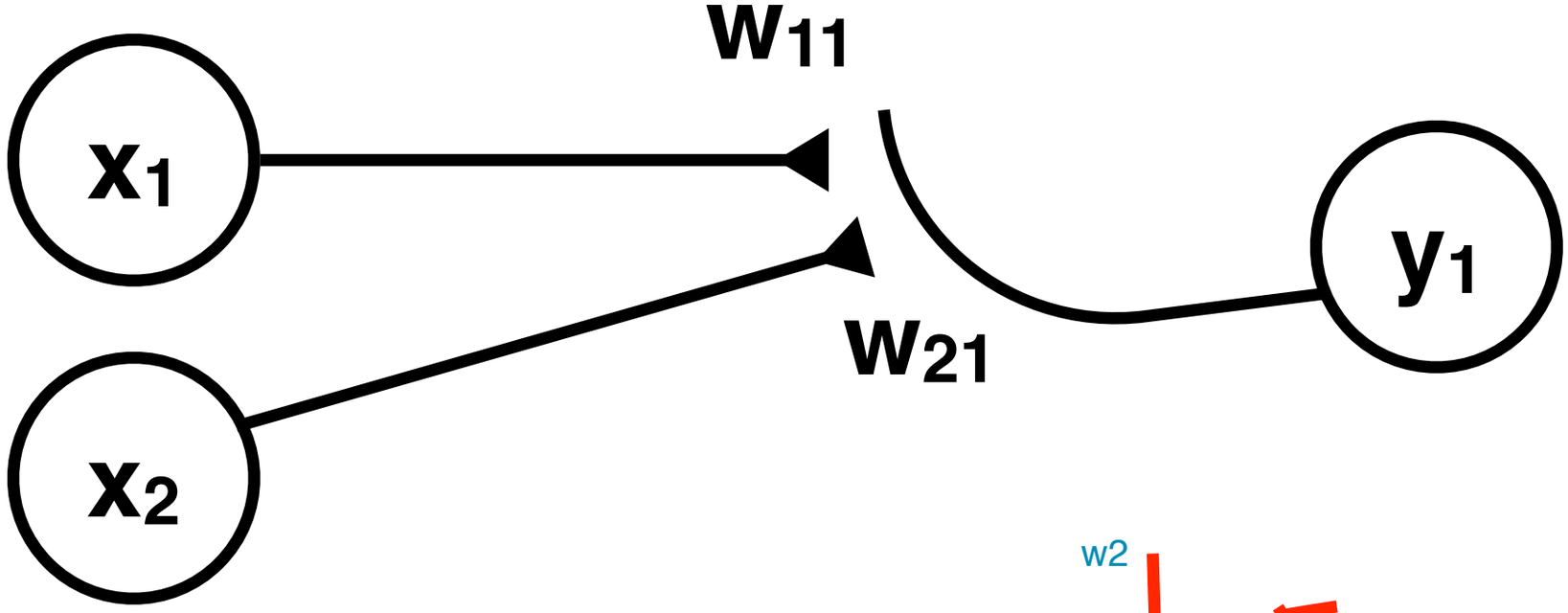
Solution to unbounded weights: weight normalization



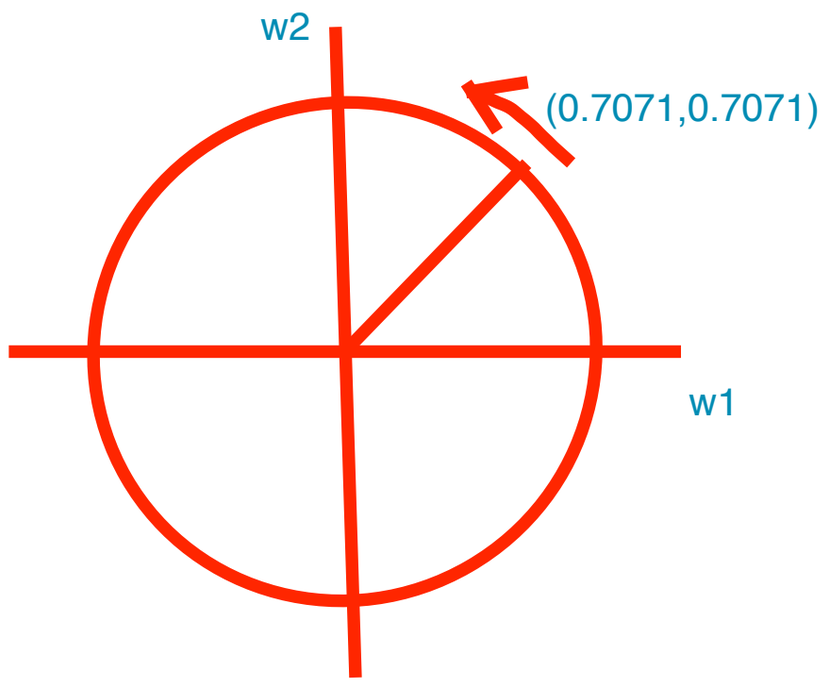
Idea of weight normalization was introduced in the 1970s and 80s. Idea: The postsynaptic cell has limited resources to spread across its synapses, which are neurophysiological structures that must be maintained, and there is competition for these resources.

Cartesian normalization: force the weight vector to have length 1 by dividing the vector by its norm

Solution to unbounded weights: weight normalization




```
lrate = 0.05
x = np.array([0.3,0.5])
W = np.array([0.7071,0.7071])
y = 1
dW = lrate * x * y
W = W + dW
W = W / np.linalg.norm(W)
W -> [0.7022,0.7119]
```



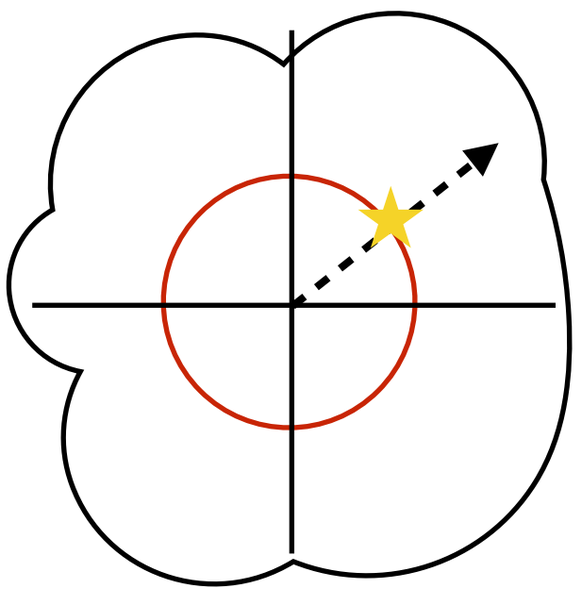
Normalization limits the weight bound to 1

Vector normalization

Divisive normalization shows up in many contexts in computational neuroscience.

It also gets used as a simple model of the influence of inhibitory circuitry controlling activation levels in neural networks.

so, where ever the point is on the plane, the normalization:
if point within boundary:
    pushes out to the circle
if point outside:
    pulls inward to the circle



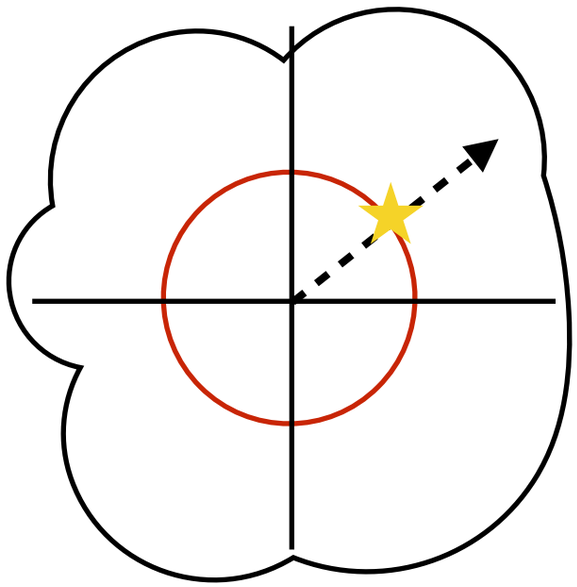
This unit circle is a 1-d manifold embedded in a 2-d plane

Mathematical operation: Normalization of a vector to the unit hypersphere.

Consider a 2d real-valued vector space, divisive normalization projects any vector to the surface of the unit circle.

Vector normalization

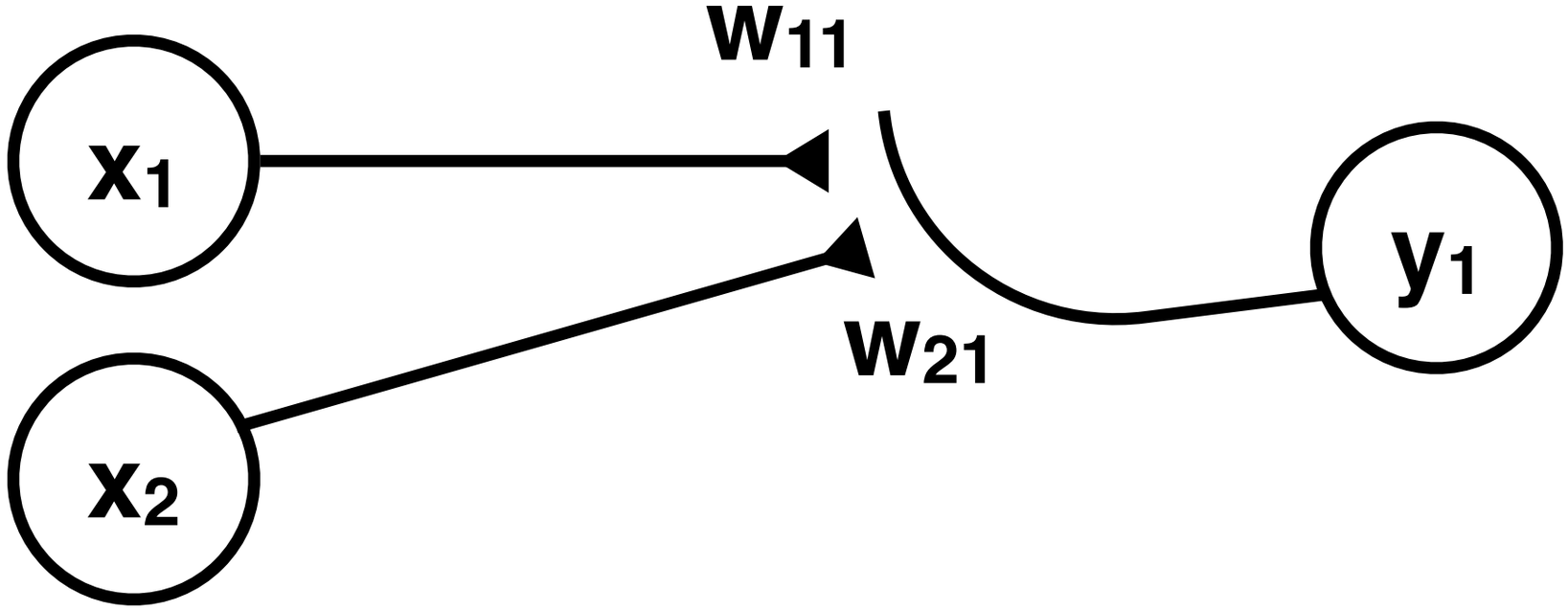
Divisive normalization: Often the magnitude of a signal is not as important as the multivariate pattern of the signal



This unit circle is a 1-d manifold embedded in a 2-d plane

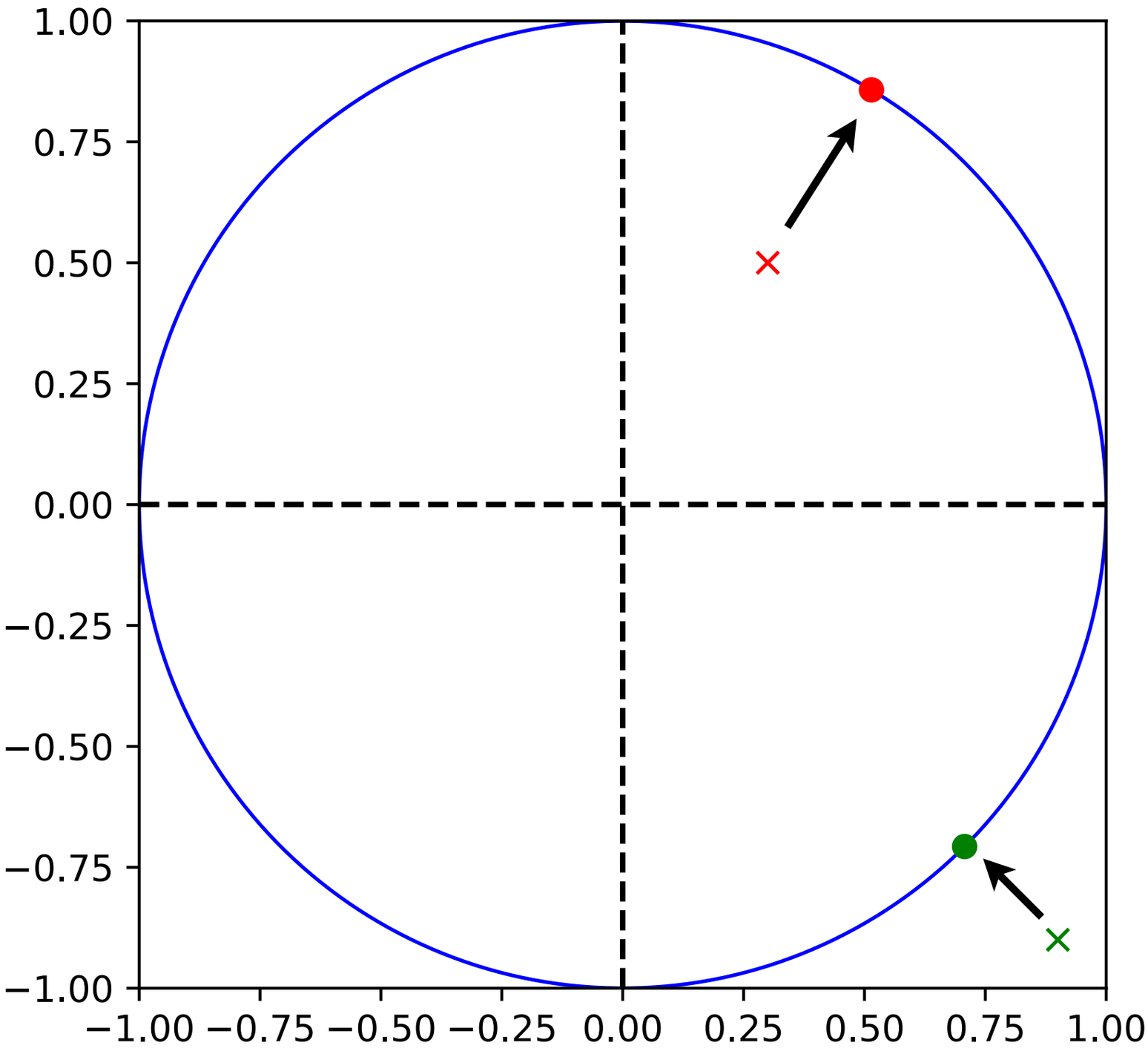
Mathematical operation: Normalization of a vector to the unit hypersphere

Control of activation level: activation normalization




```
x = np.array([0.3,0.5])
x = x / np.linalg.norm(x)
x -> [0.5144,0.8574]
```

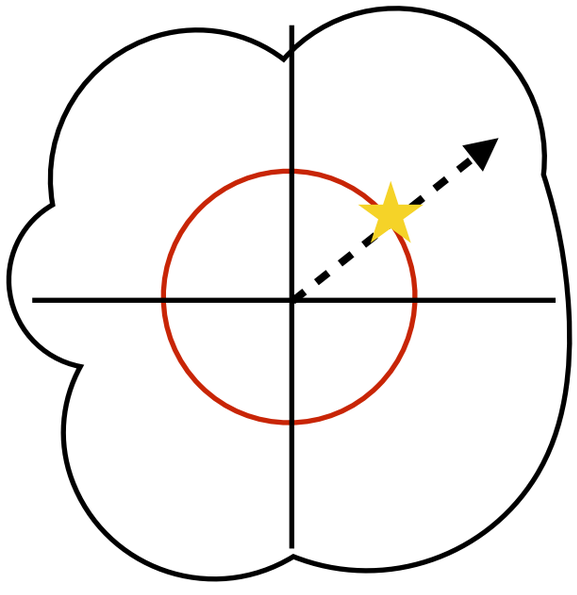
2d vector normalization projects a vector to the unit circle



Why vector normalization?

The simple assumptions of input activation normalization and weight normalization imbues Hebbian learning with a special power:

“Excitation of the postsynaptic neuron is greatest when the input vector matches the weight vector.”



This unit circle is a 1-d manifold embedded in a 2-d plane

This idea is at the heart of pattern recognition

& Hebbian learning allows weights to approximate the average input pattern

Constructing a model world

Our neural network models are meant to be abstracted and simplified pieces of real nervous systems

Containing what we believe to be the core computational elements of neurons

Proposal: if the microscopic elements of a neural system can carry out a certain computation, this computational ability is preserved in the broader neural system

William B. Levy (Chip), UVA, Dept of Neurological Surgery

Constructing a model world

In order to explore the properties of these models, we need to construct a world for them to live in, an environment for them to experience

Using random variables and probability distributions to create model worlds with known statistical properties

Learning the average input pattern

With very few assumptions (just activation normalization and weight normalization) the simple Hebbian learning rule allows a neuron to learn the average pattern of excitation across its synapses, and tune itself to respond maximally to this pattern.

The weights will reflect a running average of the input patterns. So, even without training labels we have the rudiments of a system that can figure out the structure of the world.

Oja's rule

bounding the limits by subtracting output y * current weight

```
dW = lrate * y * (x - y*W)
```

equivalently

```
dW = lrate * (x*y - y**2 * W)
```

This rule produces weight change equivalent to what you get when you use weight normalization after each learning event!

Under certain circumstances this causes the system to learn the first principal component of the input patterns. We will come back to this idea!

Oja's rule, a closer look

How does this rule stop weights from growing without bound? (from O'Reilly & Munakata) Looking at a simple case with one input pattern, under what circumstances will the system reach equilibrium?

$$dW = \varepsilon (x_i y_j - y_j^2 W_{ij})$$

$$0 = \varepsilon (x_i y_j - y_j^2 W_{ij})$$

$$y_j^2 W_{ij} = x_i y_j$$

$$W_{ij} = x_i / y_j$$

by definition

$$y_j = \sum_k x_k W_{kj}$$

$$W_{ij} = x_i / \sum_k x_k W_{kj}$$

System is in equilibrium when the weight coming from x_i is scaled by that input unit's activity level normalized by the sum of all weighted input unit activities

Correlation

The equation to calculate a correlation coefficient has a lot in common with a Hebbian weight change equation. For a correlation coeff. you subtract the mean off each variable, and you divide through by the standard deviation of each variable, but at the heart of it, that multiplicative measure determines the relationship between x and y (or a_i and a_j).

$$r_{xy} = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{n s_x s_y}$$