on Brightspace

Homework 6
due Tue Oct 25

Required Video (link on Brightspace)
What is Backpropagation Really Doing? (3Blue1Brown)

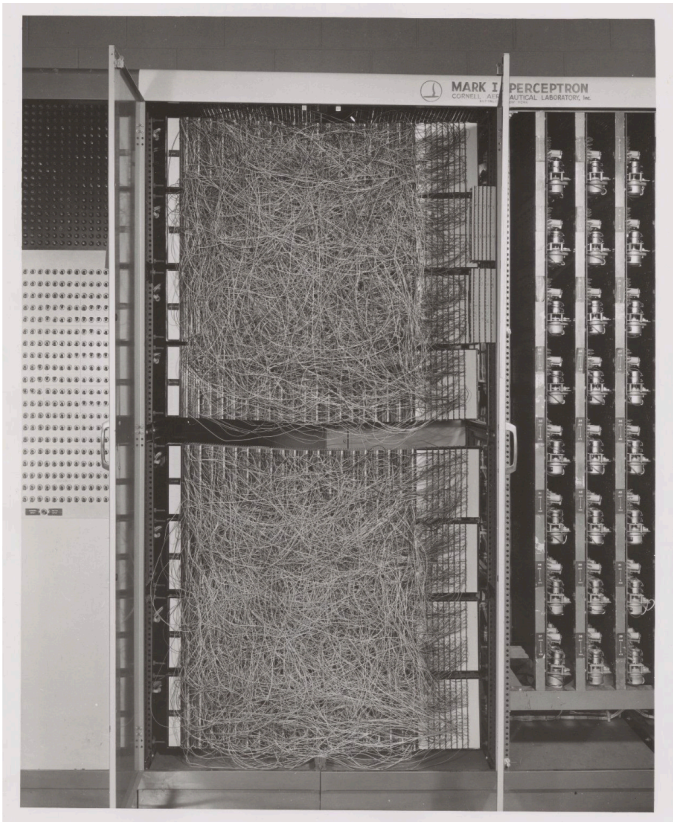
Recommended Video (link on Brightspace)
Backpropagation Calculus (3Blue1Brown)

Supplemental Reading
Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart & J.L. McClelland (Eds)., Parallel Distributed Processing: Explorations in the Microstructure of Cognition (Vol. 1), MIT Press.

Multi-layered Networks and Backpropagation

with a bit of history

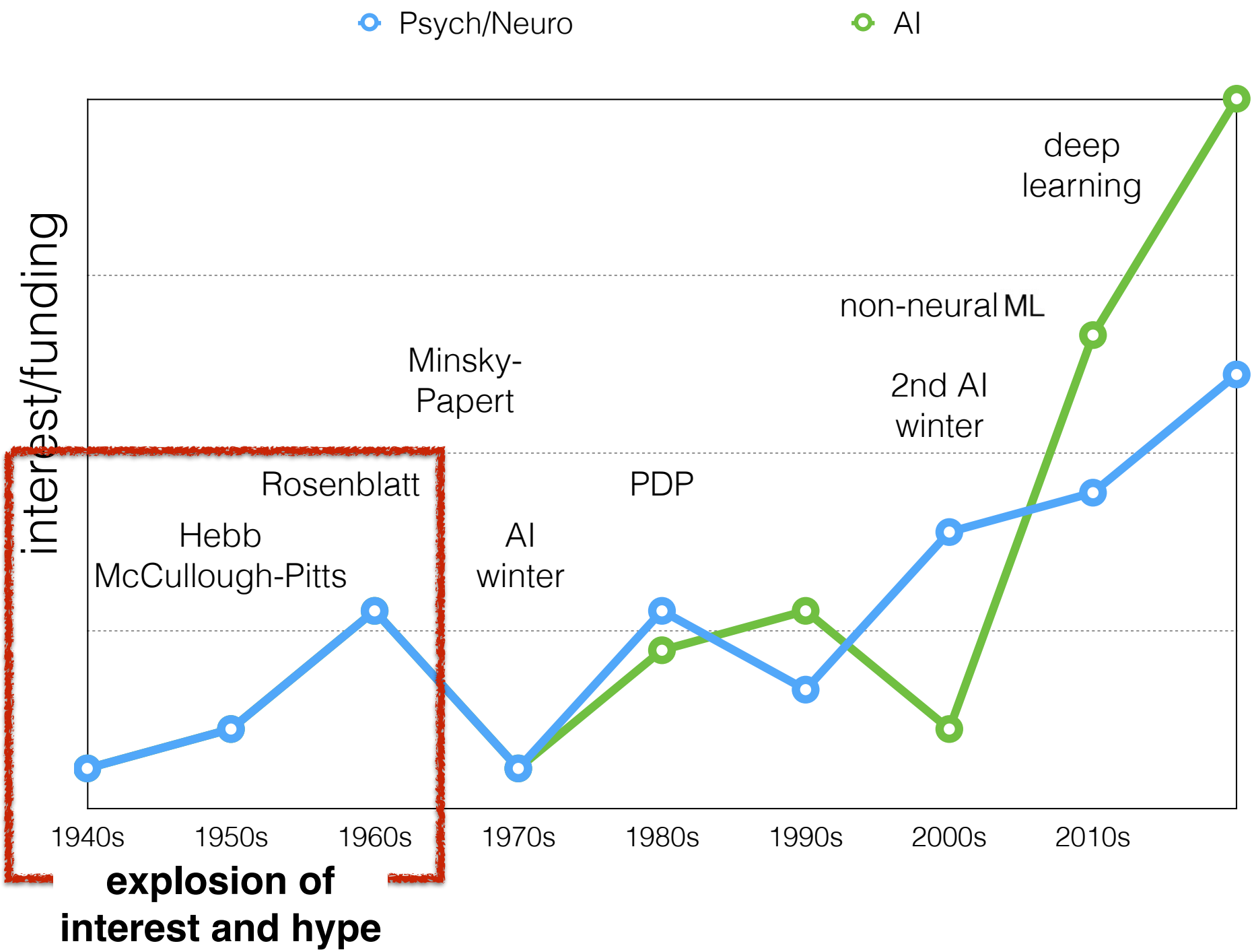
Mark I Perceptron (single-layer network)



Built on earlier work of McCulloch & Pitts (late 1940s/early 1950s) that "neural-like" elements could perform logic, and the work of Hebb on possible learning rules (though Perceptron Learning Rule is more error-driven than Hebbian)

Shows how simple neural networks could **learn**

a brief history of neural network models



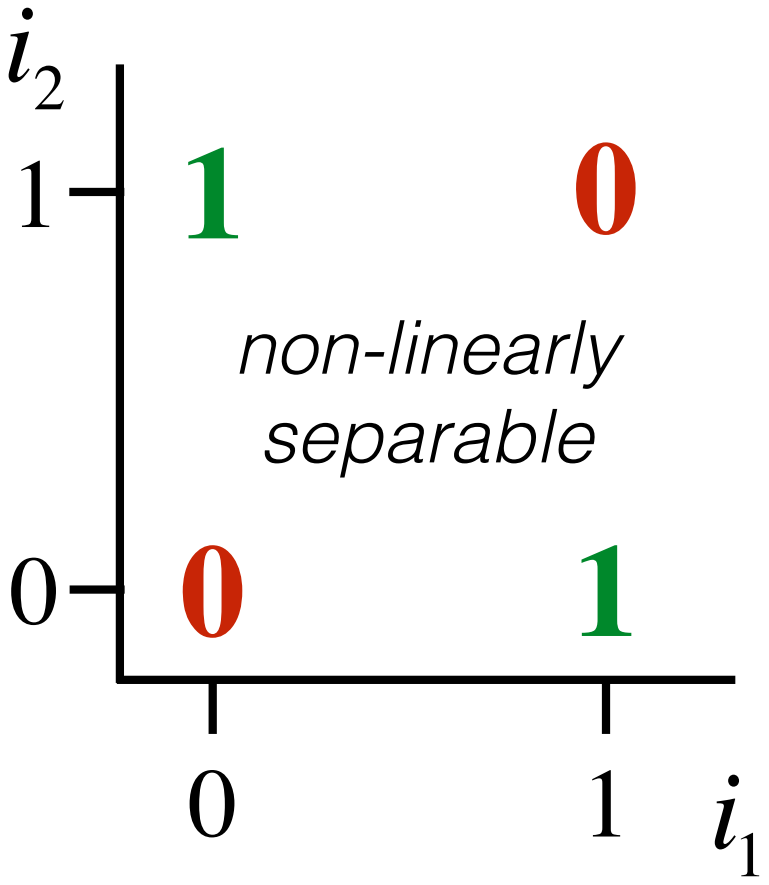
https://www.youtube.com/watch?v=aygSMgK3BEM

The Thinking Machine (Artificial Intelligence in the 1960s)

1960s was the first era of both neural and symbolic "AI"

However …

Example of a Simple Neural Network



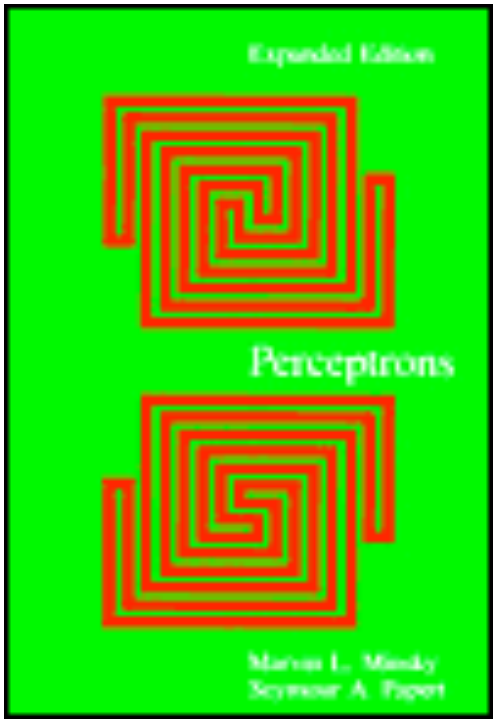
i_1	i_2	o
0	0	0
1	0	1
0	1	1
1	1	0

logical
XOR

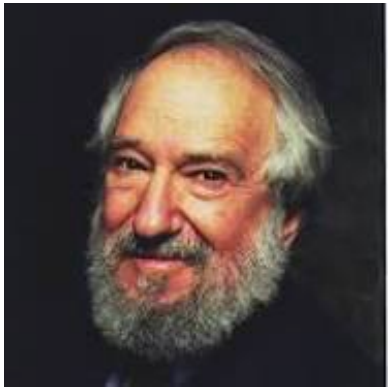
no set of weights can
solve this problem

Perceptrons

Marvin Minsky
1927-2016



Minsky & Papert, 1969

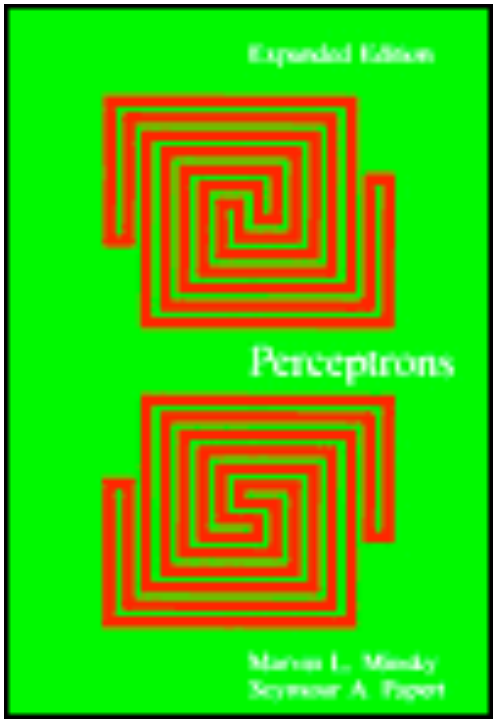


Seymour Papert
1928-2016

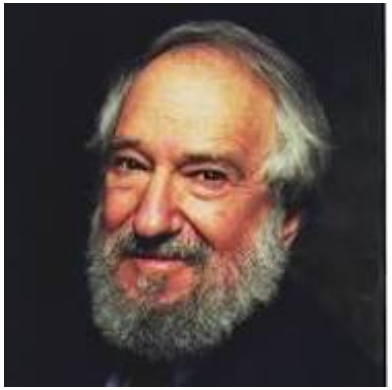
Minsky and Paper wrote, “Perceptrons have been widely publicized as ‘pattern recognition' or 'learning machines' and as such have been discussed in a large number of books, journal articles, and voluminous 'reports'. Most of this writing ... is without scientific value ..”

Perceptrons

Marvin Minsky
1927-2016



Minsky & Papert, 1969

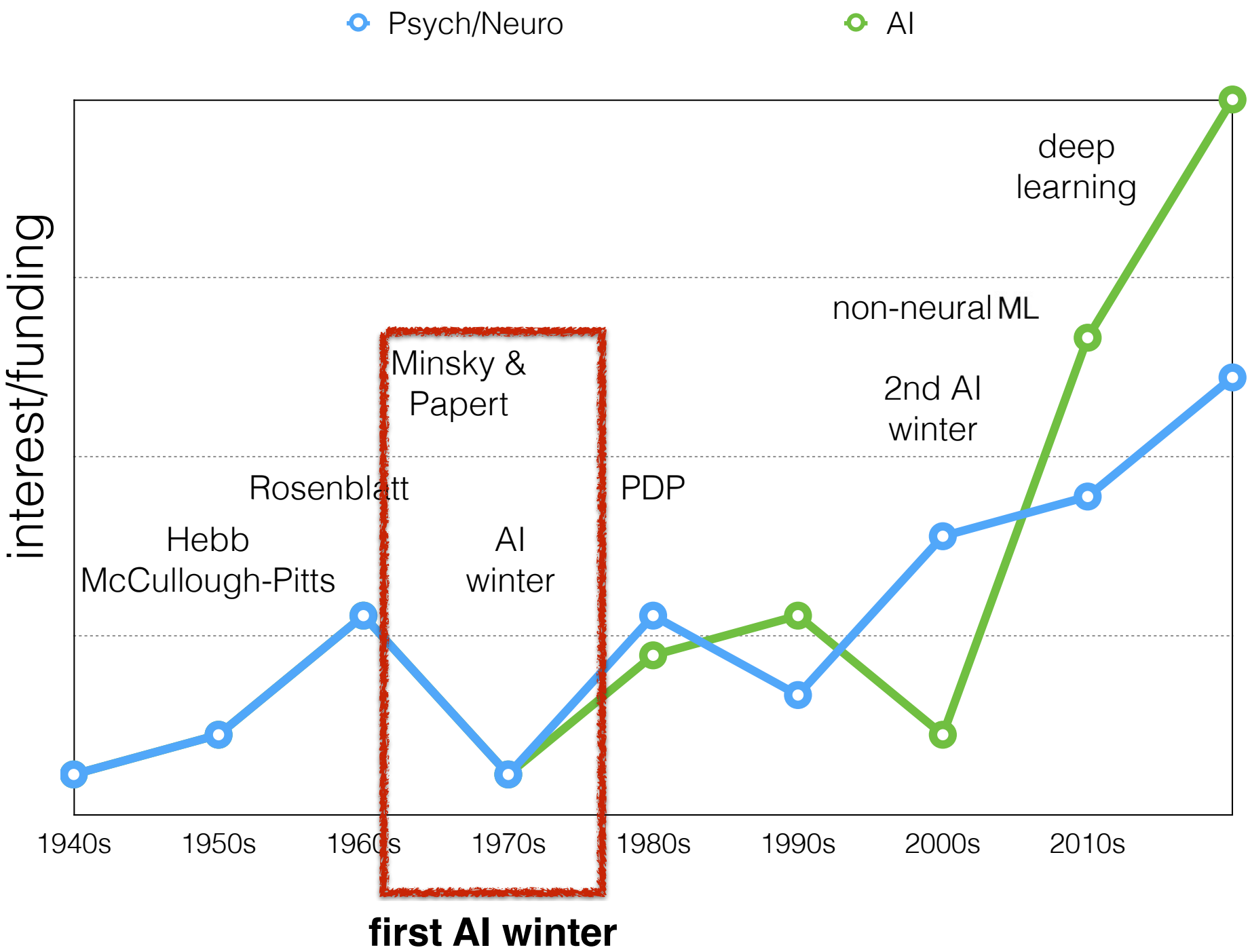


Seymour Papert
1928-2016

Minsky and Papert proved that simple perceptron (single-layer) models could not solve the XOR problem. Rosenblatt (really, everyone) knew, via work of McCullough and Pitts two decades earlier, that multi-layer networks could solve any logic problem, including the XOR. Minsky and Paper highlighted some constraints on multi-layer networks and **suggested - without any proof - that no algorithm existed that could ever train multi-layer networks**.

Funding and interest in neural network research was largely cut off, and apart from Grossberg, Anderson, Kohonen, and a few other stalwarts, research on neural network models largely languished for a decade, blossoming again in the 1980s.

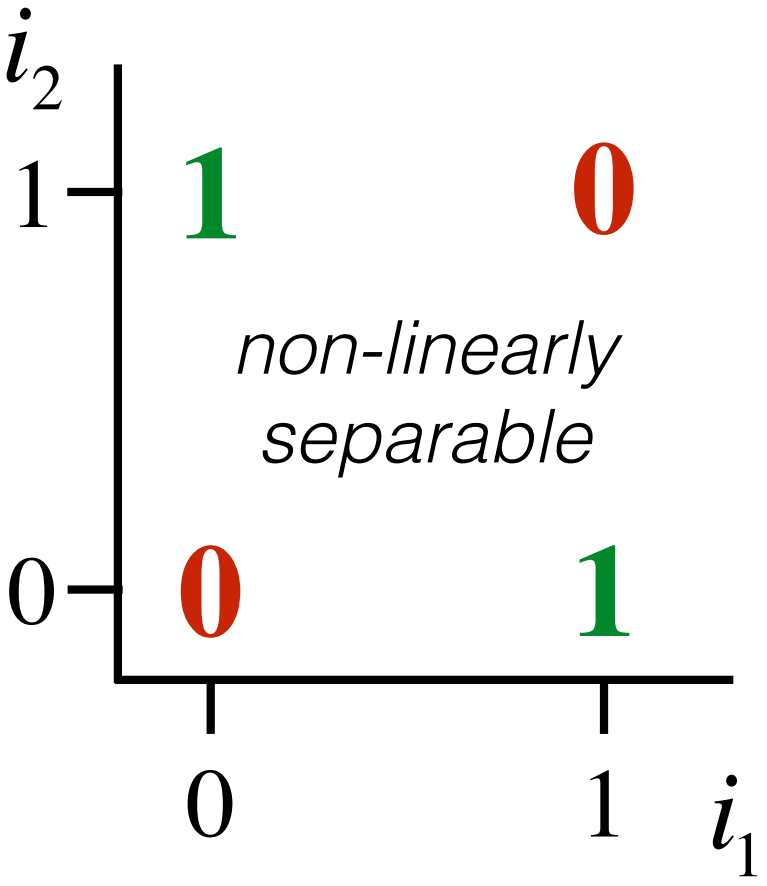
a brief history of neural network models



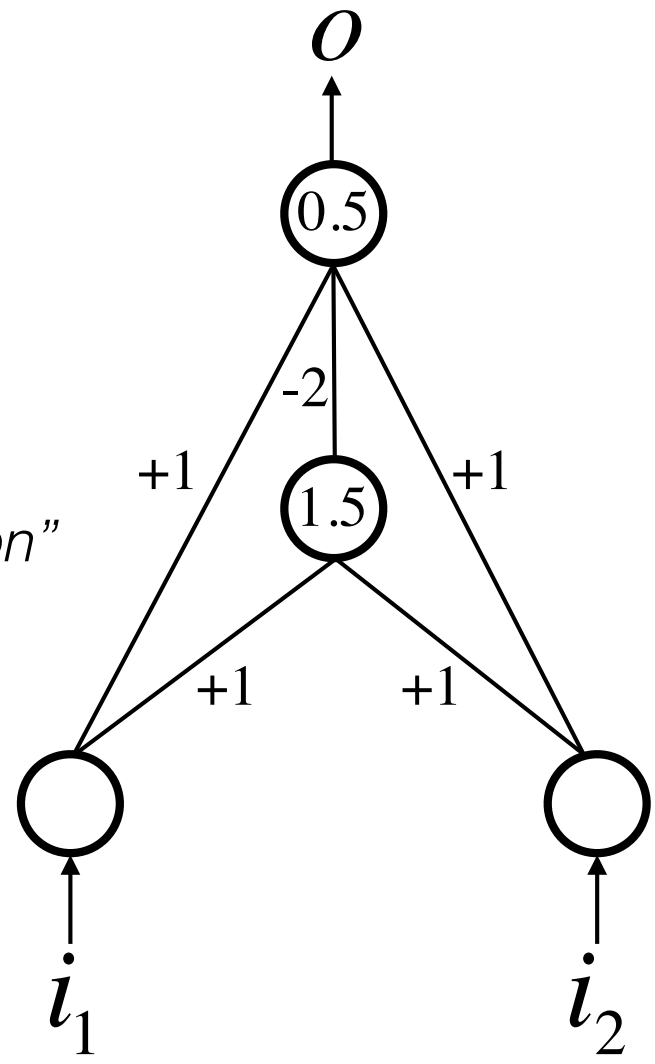
how can a multi-layer network do the XOR?

(we'll address how they can learn in a bit)

assume a step function with threshold given by number inside unit



"internal representation"

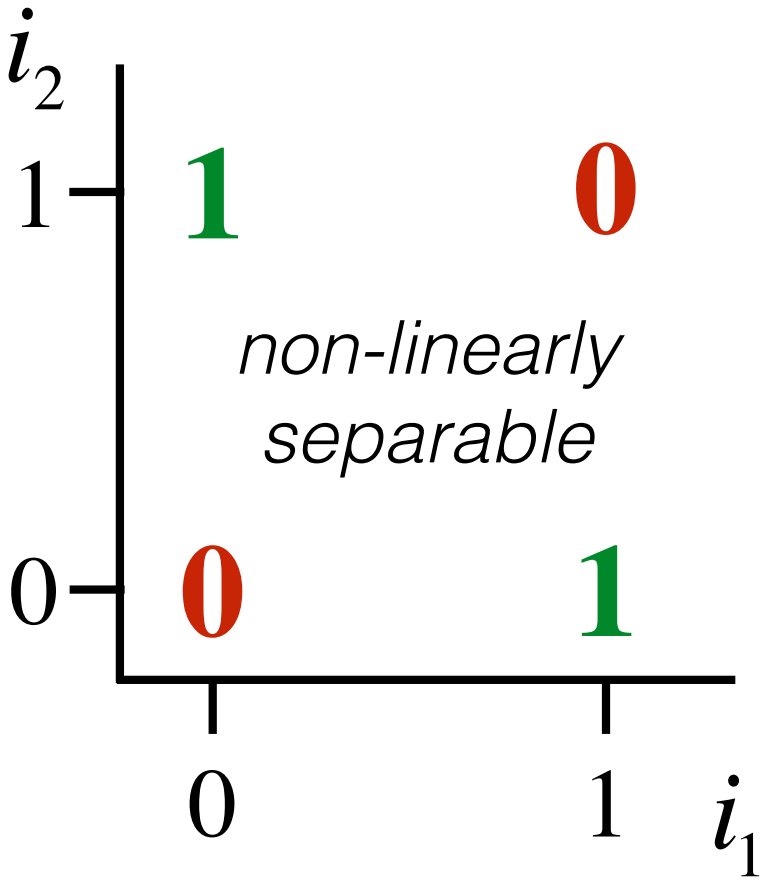


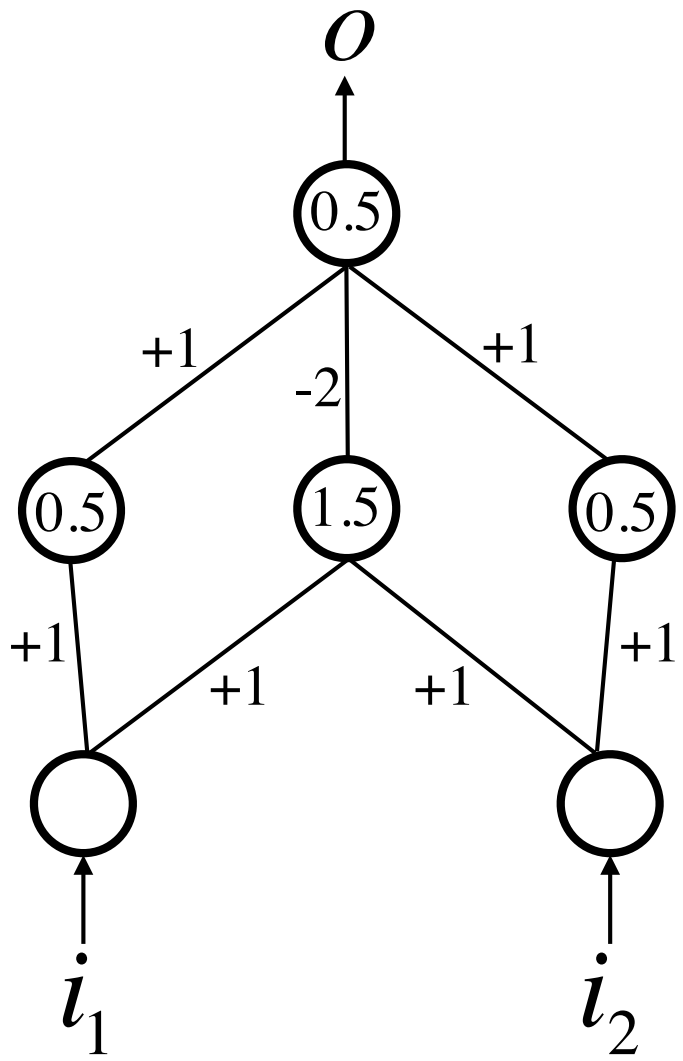
what is the hidden unit doing?

how can a multi-layer network do the XOR?

(we'll address how they can learn in a bit)

assume a step function with threshold given by number inside unit

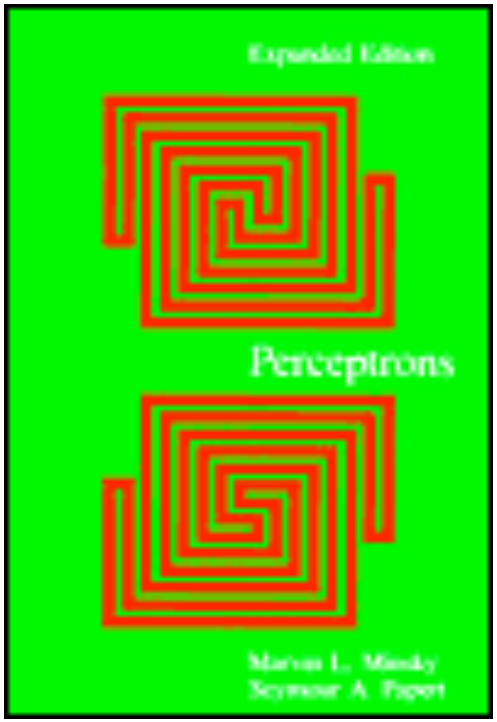




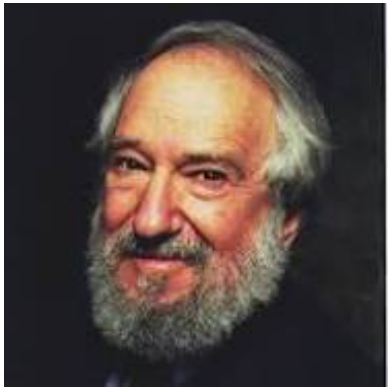
what is the hidden unit doing?

Perceptrons

Marvin Minsky
1927-2016



Minsky & Papert, 1969

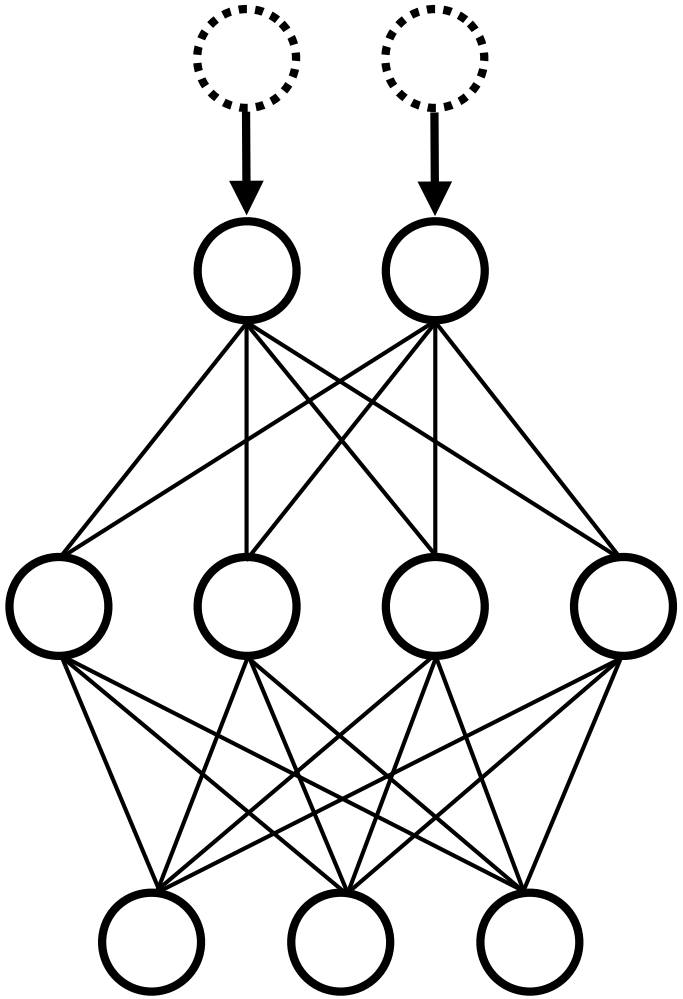


Seymour Papert
1928-2016

Paul Werbos, in 1974, for his PhD thesis, discovered a way to train multi-layer neural networks - such networks could learn the XOR problem. But no one was interested. Neural networks were dead.

Werbos visited Minsky at MIT and presented his work and even discussed collaborating (Werbos was a mathematician and physicist) but Minksy reportedly had no interest. Werbos did not publish his work until the early 1980s.

Multi-layer Neural Networks



teacher

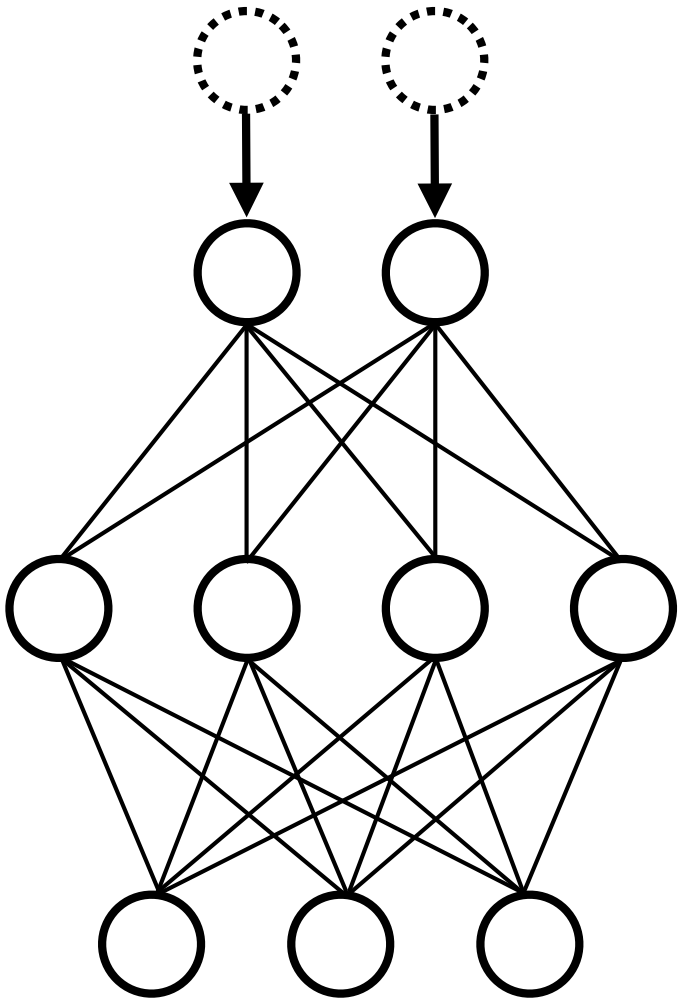
output layer

hidden layer

input layer

Multi-layer Neural Networks

* if activations on hidden layers are linear, then a multi-layer neural network collapses to a single-layer network (same limitations)



teacher

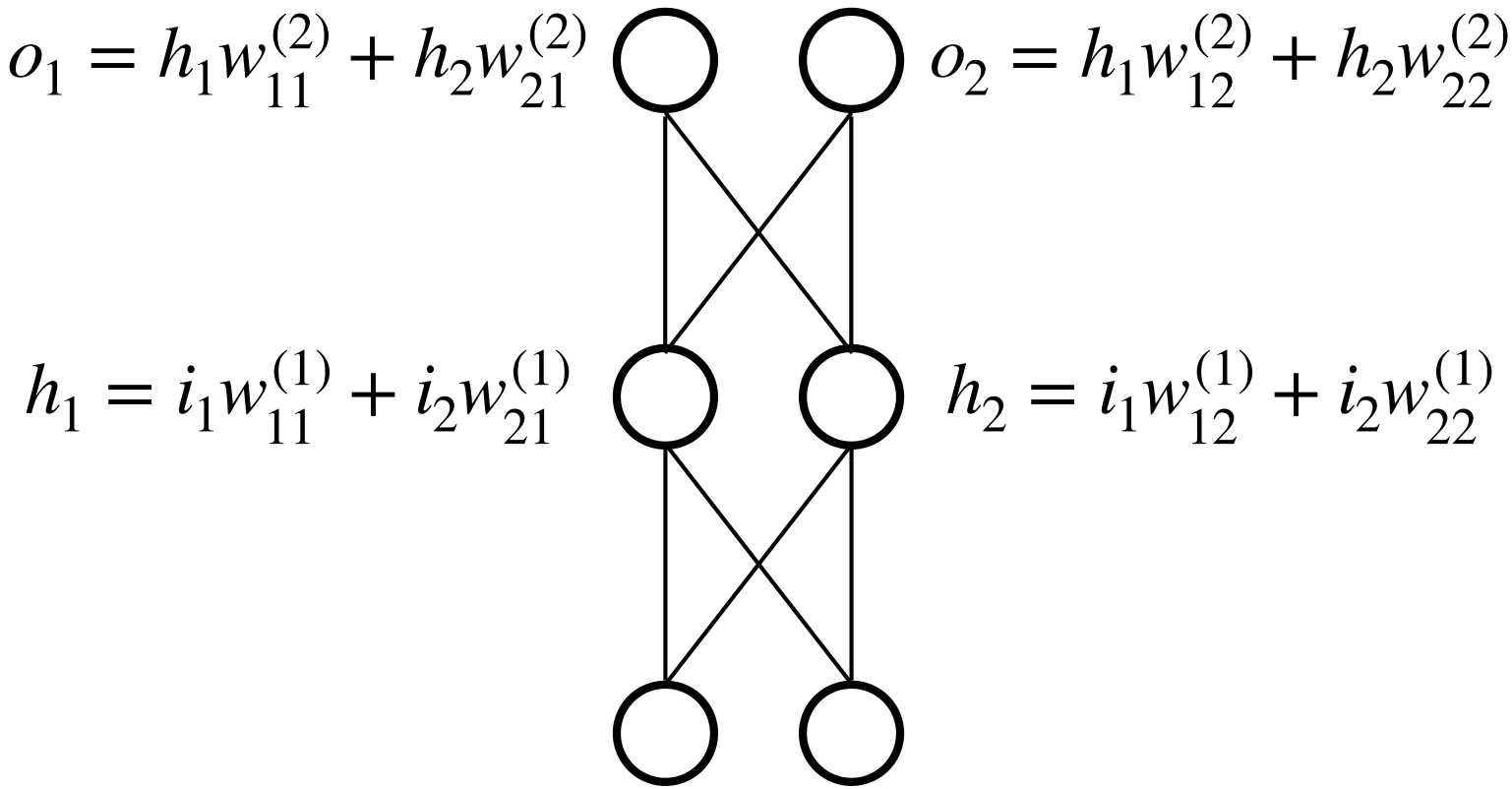
output layer

hidden layer

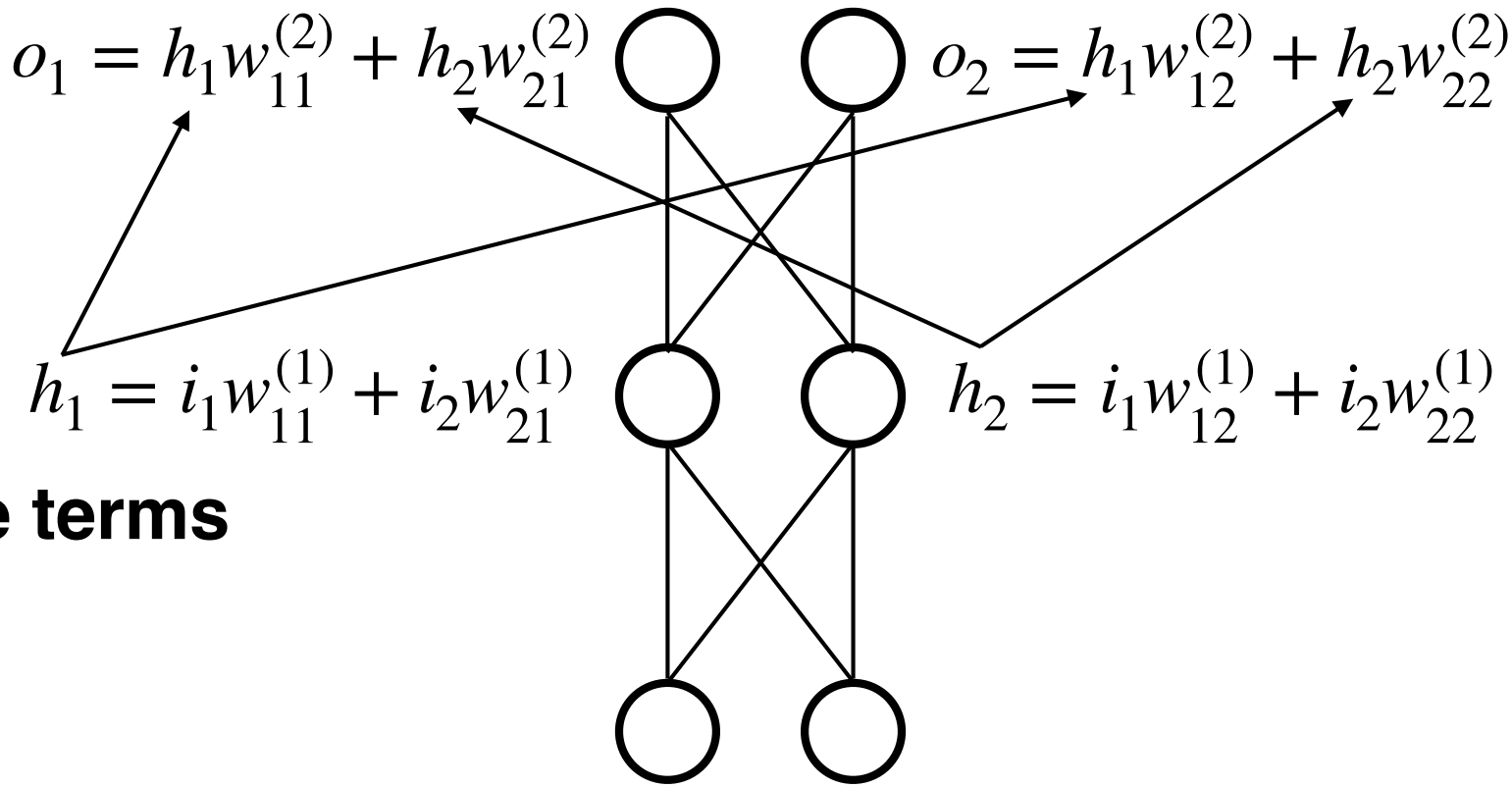
input layer

* *must have nonlinear activation functions on hidden layers*

* if activations on hidden layers are linear, then a multi-layer neural network collapses to a single-layer network (same limitations)



* if activations on hidden layers are linear, then a multi-layer neural network collapses to a single-layer network (same limitations)



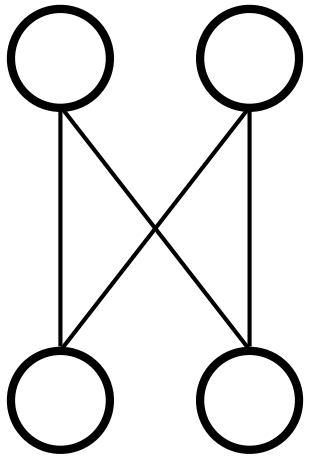
replace terms

* if activations on hidden layers are linear, then a multi-layer neural network collapses to a single-layer network (same limitations)

$$o_1 = (i_1 w_{11}^{(1)} + i_2 w_{21}^{(1)}) w_{11}^{(2)} + (i_1 w_{12}^{(1)} + i_2 w_{22}^{(1)}) w_{21}^{(2)}$$

$$o_2 = (i_1 w_{11}^{(1)} + i_2 w_{21}^{(1)}) w_{12}^{(2)} + (i_1 w_{12}^{(1)} + i_2 w_{22}^{(1)}) w_{22}^{(2)}$$

do some algebra

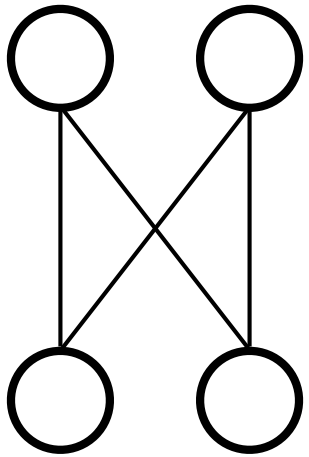


* if activations on hidden layers are linear, then a multi-layer neural network collapses to a single-layer network (same limitations)

$$o_1 = i_1 w_{11}^{(1)} w_{11}^{(2)} + i_2 w_{21}^{(1)} w_{11}^{(2)} + i_1 w_{12}^{(1)} w_{21}^{(2)} + i_2 w_{22}^{(1)} w_{21}^{(2)}$$

$$o_2 = i_1 w_{11}^{(1)} w_{12}^{(2)} + i_2 w_{21}^{(1)} w_{12}^{(2)} + i_1 w_{12}^{(1)} w_{22}^{(2)} + i_2 w_{22}^{(1)} w_{22}^{(2)}$$

and some more algebra

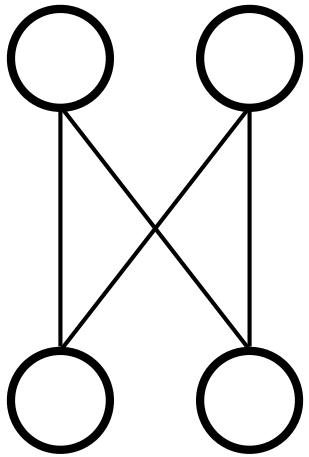


* if activations on hidden layers are linear, then a multi-layer neural network collapses to a single-layer network (same limitations)

$$o_1 = i_1(w_{11}^{(1)}w_{11}^{(2)} + w_{12}^{(1)}w_{21}^{(2)}) + i_2(w_{21}^{(1)}w_{11}^{(2)} + w_{22}^{(1)}w_{21}^{(2)})$$

$$o_2 = i_1(w_{11}^{(1)}w_{12}^{(2)} + w_{12}^{(1)}w_{22}^{(2)}) + i_2(w_{21}^{(1)}w_{12}^{(2)} + w_{22}^{(1)}w_{22}^{(2)})$$

and some more algebra

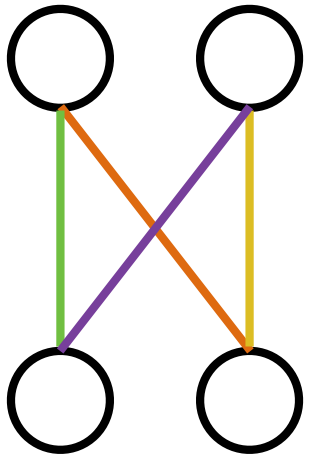


* if activations on hidden layers are linear, then a multi-layer neural network collapses to a single-layer network (same limitations)

$$o_1 = i_1(w_{11}^{(1)}w_{11}^{(2)} + w_{12}^{(1)}w_{21}^{(2)}) + i_2(w_{21}^{(1)}w_{11}^{(2)} + w_{22}^{(1)}w_{21}^{(2)})$$

$$o_2 = i_1(w_{11}^{(1)}w_{12}^{(2)} + w_{12}^{(1)}w_{22}^{(2)}) + i_2(w_{21}^{(1)}w_{12}^{(2)} + w_{22}^{(1)}w_{22}^{(2)})$$

$o_1 = i_1 w_{11}^* + i_2 w_{21}^*$

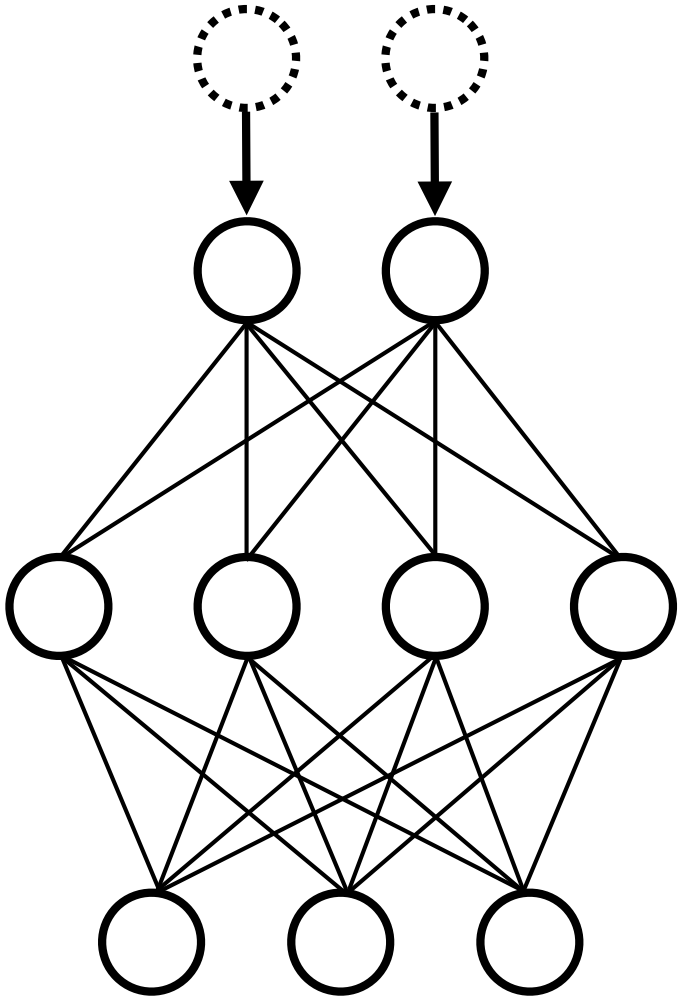


$o_2 = i_1 w_{12}^* + i_2 w_{22}^*$

demonstration that there is a mathematically equivalent network with only an input and an output layer

Multi-layer Neural Networks

* the output activations can be linear (for regression problems) or nonlinear (for classification problems)



teacher

output layer

hidden layer

input layer

linear = regression

output can be linear or nonlinear

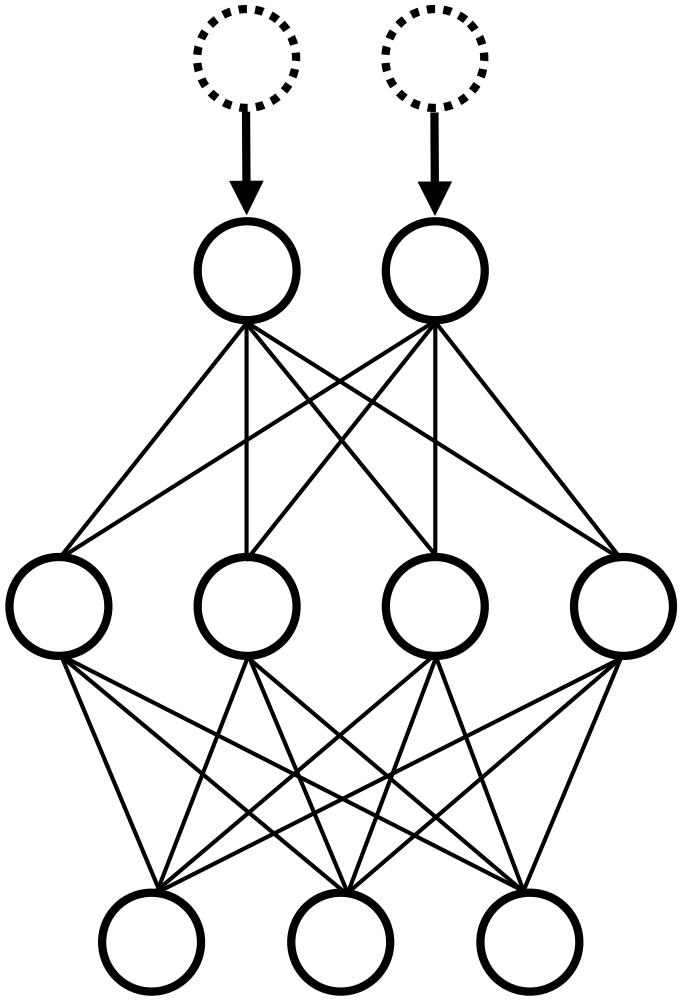
nonlinear = classification

must have nonlinear activation functions

on hidden layers

Multi-layer Neural Networks

How do we train weights in networks with multiple layers?
backpropagation (generalized delta rule)



teacher

output layer

hidden layer

input layer

linear = regression

output can be linear or nonlinear

nonlinear = classification

must have nonlinear activation functions
on hidden layers

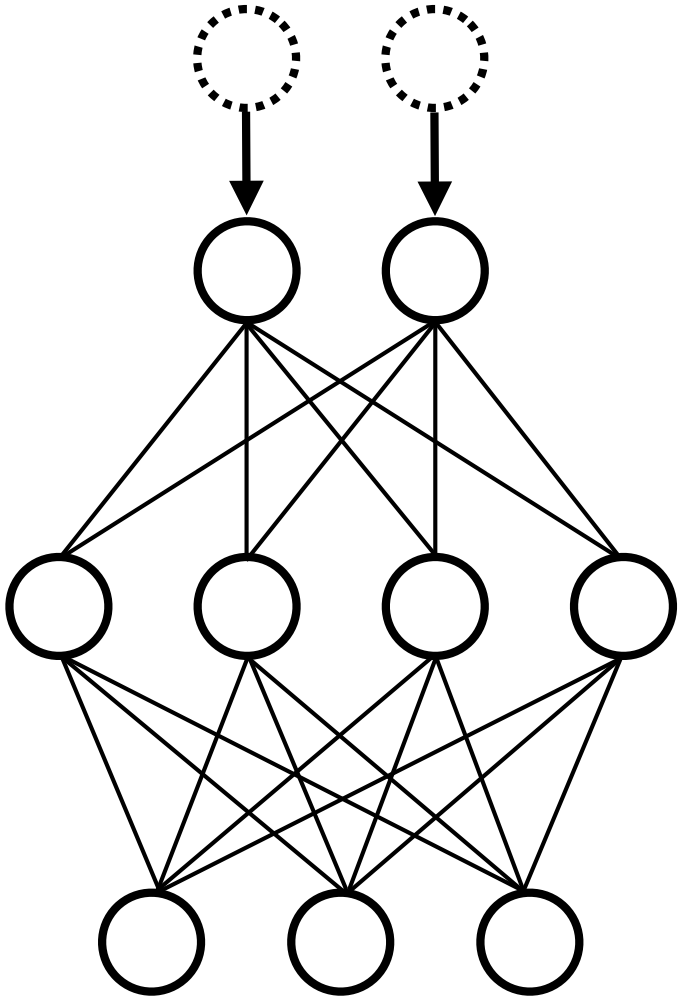
derivation of backpropagation
(generalized delta rule)

see **DeriveBackprop.pdf** on Brightspace for detailed derivations

Backpropagation / Generalized Delta Rule

supervised error-driven learning by
multi-layered feedforward neural networks

Rumelhart, Hinton & Williams, 1986; independently discovered in 1960's and 1970's



teacher

output layer

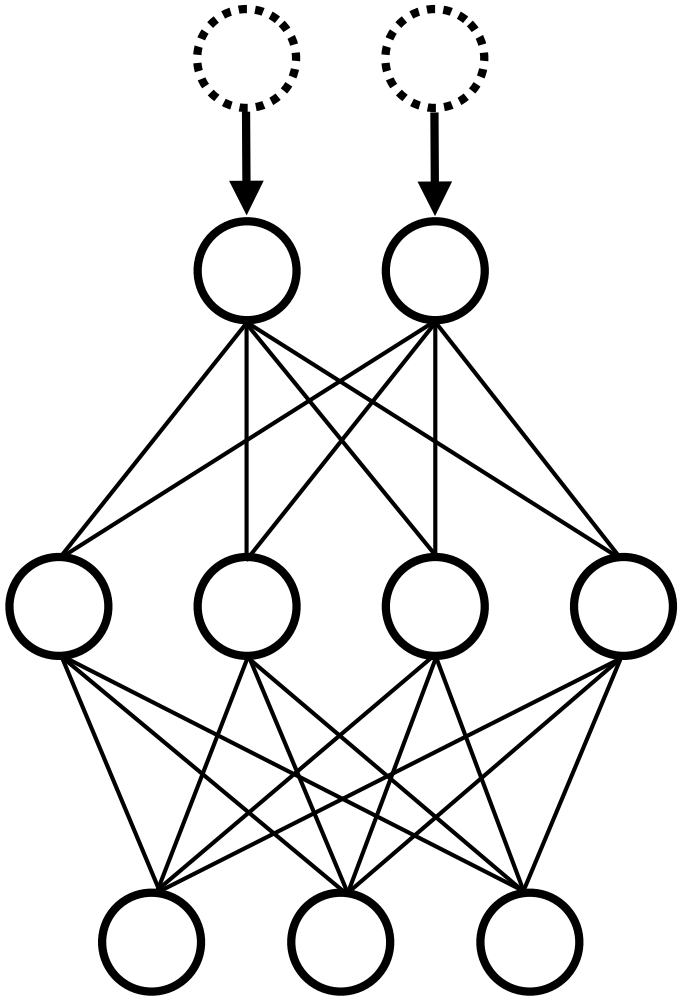
hidden layer

input layer

Backpropagation / Generalized Delta Rule

supervised error-driven learning by multi-layered feedforward neural networks

Rumelhart, Hinton & Williams, 1986; independently discovered in 1960's and 1970's



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

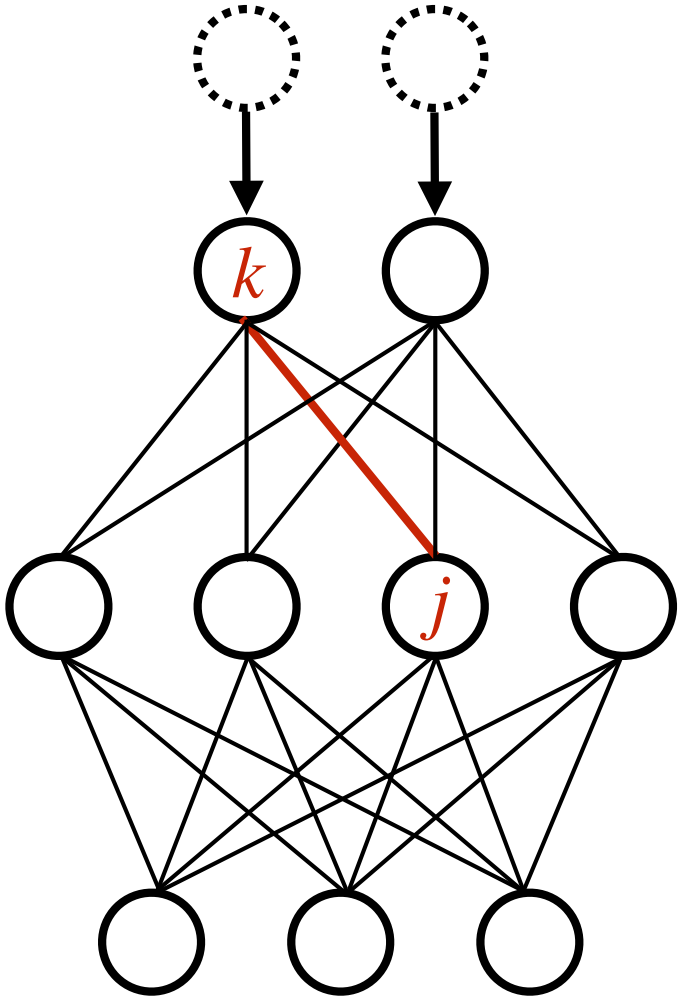
w_{ij}

$i_i^{(p)}$

Backpropagation / Generalized Delta Rule

supervised error-driven learning by multi-layered feedforward neural networks

Rumelhart, Hinton & Williams, 1986; independently discovered in 1960's and 1970's



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk} *hidden to output weight change*

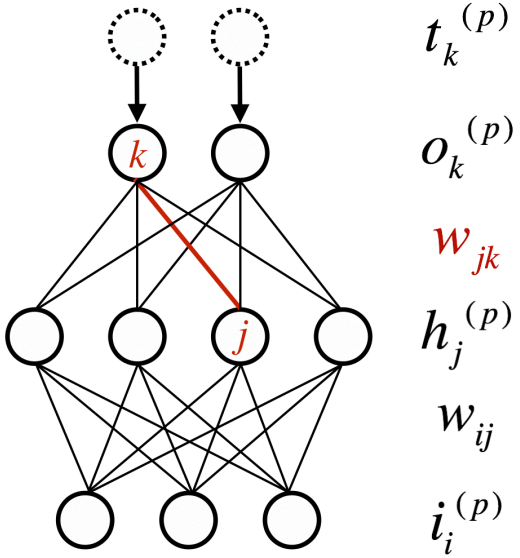
$h_j^{(p)}$

w_{ij}

$i_i^{(p)}$

$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} E_k^{(p)} = \sum_{p=1}^{P} \sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)})^2$$

error-driven learning



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

w_{ij}

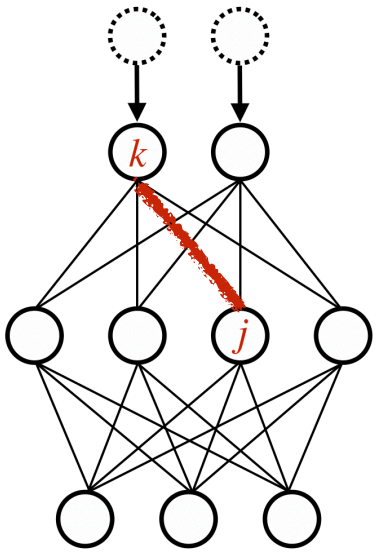
$i_i^{(p)}$

$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} E_k^{(p)} = \sum_{p=1}^{P} \sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)})^2$$

$$-\frac{\partial E}{\partial w_{jk}} = \sum_{p=1}^{P} -\frac{\partial E_k^{(p)}}{\partial o_k^{(p)}} \frac{\partial o_k^{(p)}}{\partial net_k} \frac{\partial net_k}{\partial w_{jk}}$$

only output o_k is affected by weight w_{jk}

so no need to sum over k for changing weight w_{jk}



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

w_{ij}

$i_i^{(p)}$

$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} E_k^{(p)} = \sum_{p=1}^{P} \sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)})^2$$

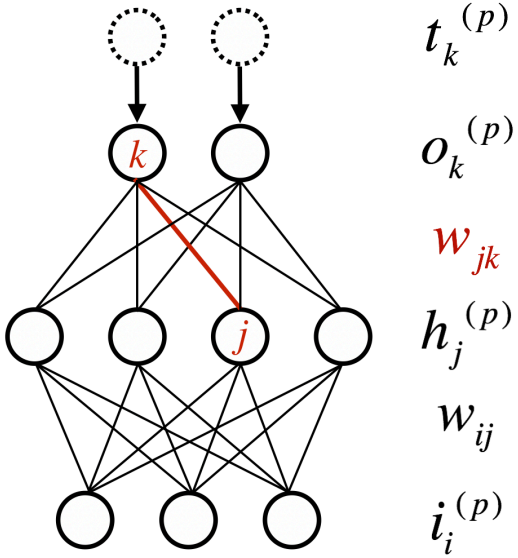
assuming sigmoidal activation

$$\Delta w_{jk} = \varepsilon \sum_{p=1}^{P} (t_k^{(p)} - o_k^{(p)}) o_k^{(p)} \left(1 - o_k^{(p)}\right) h_j^{(p)}$$

this is just the delta rule

(for a single-layer network)

can use all patterns, incremental, or (mini-)batch



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

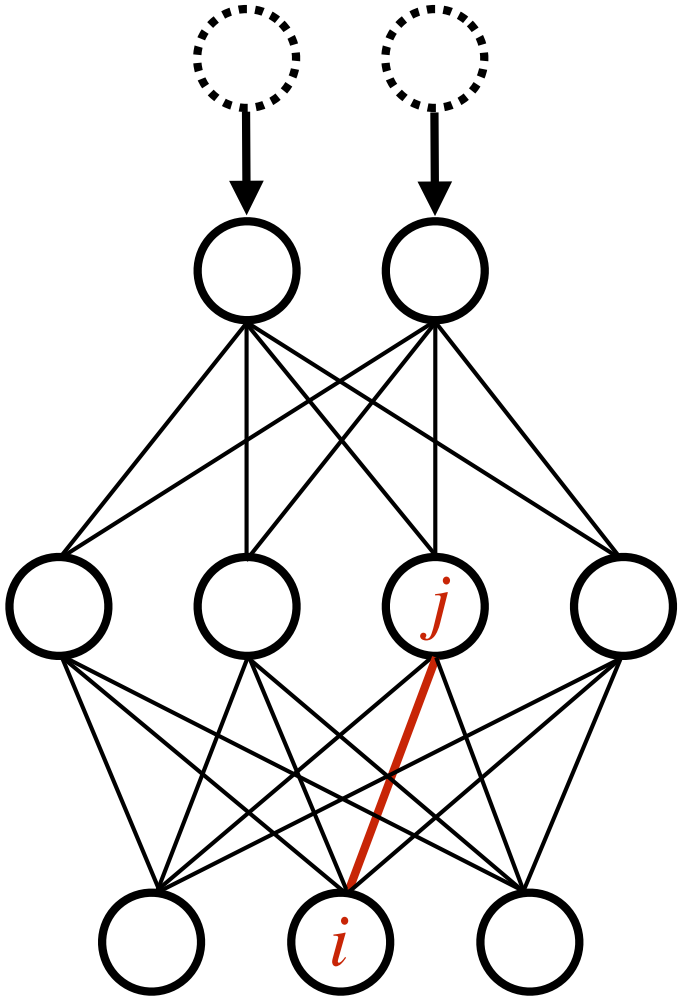
w_{ij}

$i_i^{(p)}$

Backpropagation / Generalized Delta Rule

supervised error-driven learning by multi-layered feedforward neural networks

Rumelhart, Hinton & Williams, 1986; independently discovered in 1960's and 1970's



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

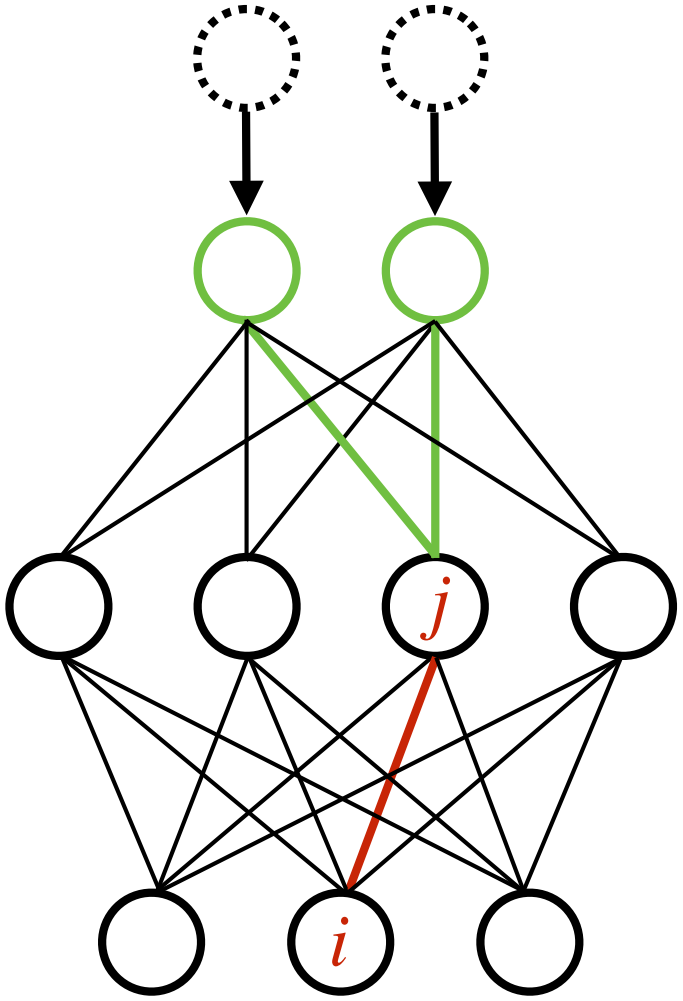
w_{ij} *input to hidden weight change*

$i_i^{(p)}$

Backpropagation / Generalized Delta Rule

supervised error-driven learning by multi-layered feedforward neural networks

Rumelhart, Hinton & Williams, 1986; independently discovered in 1960's and 1970's



$t_k^{(p)}$

$o_k^{(p)}$

note that changing w_{ij} affects all the output nodes, not just one

w_{jk}

$h_j^{(p)}$

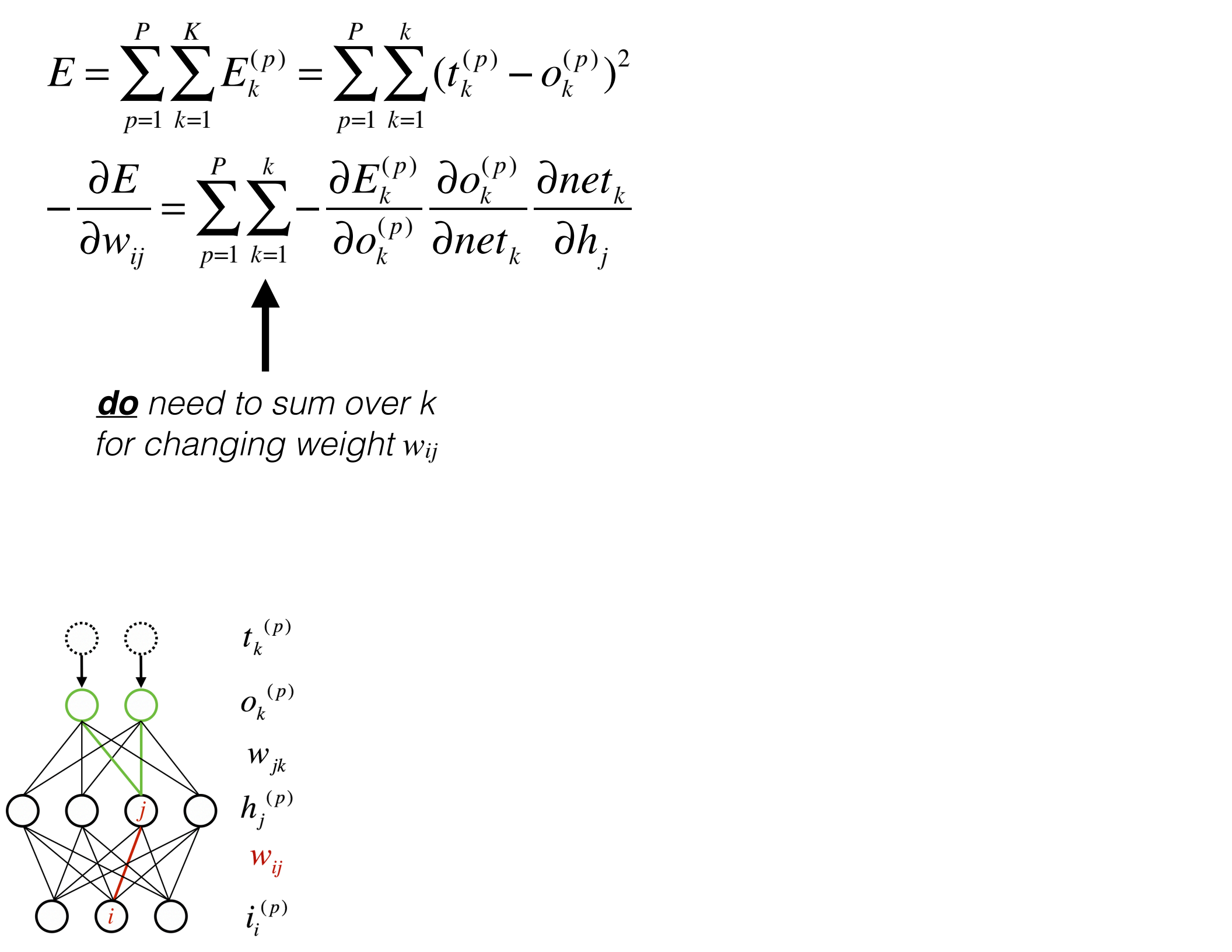
w_{ij}

input to hidden weight change

$i_i^{(p)}$

$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} E_k^{(p)} = \sum_{p=1}^{P} \sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)})^2$$

$$-\frac{\partial E}{\partial w_{ij}} = \sum_{p=1}^{P} \sum_{k=1}^{k} -\frac{\partial E_k^{(p)}}{\partial o_k^{(p)}} \frac{\partial o_k^{(p)}}{\partial net_k} \frac{\partial net_k}{\partial h_j}$$



do *need to sum over k*
for changing weight w_{ij}

$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

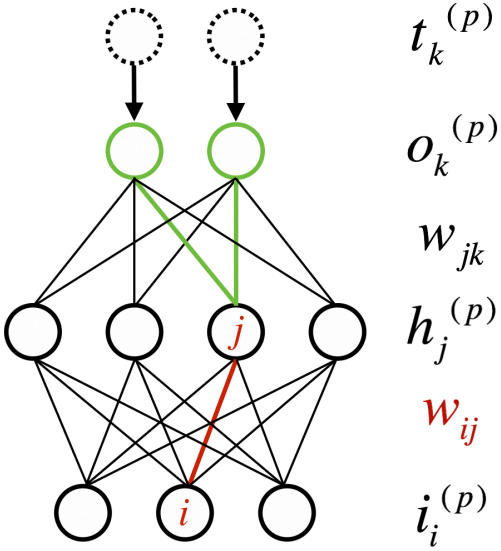
w_{ij}

$i_i^{(p)}$

$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} E_k^{(p)} = \sum_{p=1}^{P} \sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)})^2$$

$$-\frac{\partial E}{\partial w_{ij}} = \sum_{p=1}^{P} \sum_{k=1}^{k} -\frac{\partial E_k^{(p)}}{\partial o_k^{(p)}} \frac{\partial o_k^{(p)}}{\partial net_k} \frac{\partial net_k}{\partial h_j}$$

note that this is a derivative with respect to h_j



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

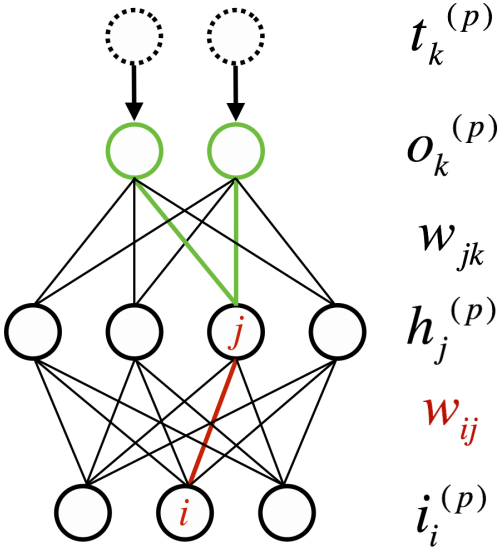
$h_j^{(p)}$

w_{ij}

$i_i^{(p)}$

$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} E_k^{(p)} = \sum_{p=1}^{P} \sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)})^2$$

$$-\frac{\partial E}{\partial w_{ij}} = \sum_{p=1}^{P} \left[\sum_{k=1}^{k} -\frac{\partial E_k^{(p)}}{\partial o_k^{(p)}} \frac{\partial o_k^{(p)}}{\partial net_k} \frac{\partial net_k}{\partial h_j} \right] \frac{\partial h_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

w_{ij}

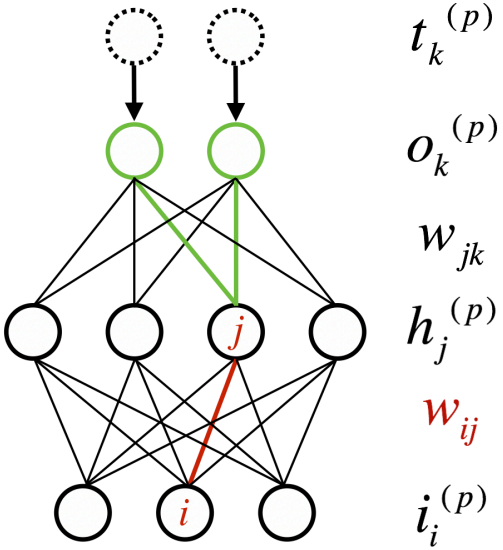
$i_i^{(p)}$

$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} E_k^{(p)} = \sum_{p=1}^{P} \sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)})^2$$

assuming sigmoidal activation

$$\Delta w_{ij} = \varepsilon \sum_{p=1}^{P} \left[\sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)}) o_k^{(p)} \left(1 - o_k^{(p)}\right) w_{jk} \right] h_j^{(p)} \left(1 - h_j^{(p)}\right) i_i^{(p)}$$

can use all patterns, incremental, or (mini-)batch



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

w_{ij}

$i_i^{(p)}$

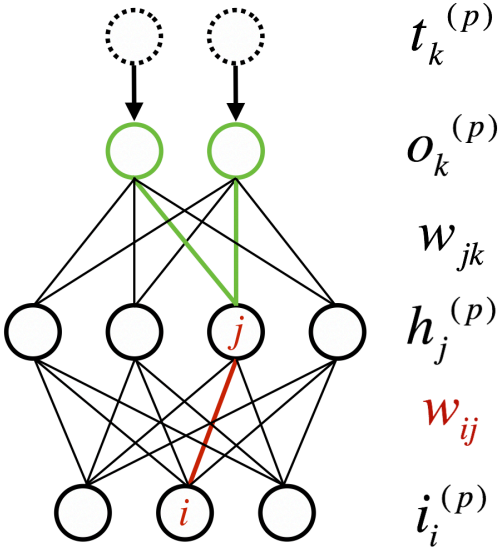
$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} E_k^{(p)} = \sum_{p=1}^{P} \sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)})^2$$

assuming sigmoidal activation

$$\Delta w_{ij} = \varepsilon \sum_{p=1}^{P} \left[\sum_{k=1}^{k} (t_k^{(p)} - o_k^{(p)}) o_k^{(p)} \left(1 - o_k^{(p)}\right) w_{jk} \right] h_j^{(p)} \left(1 - h_j^{(p)}\right) i_i^{(p)}$$

can use all patterns, incremental, or (mini-)batch

we will not ask you to implement backprop by hand
we will use Keras instead to take care of the bookkeeping



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

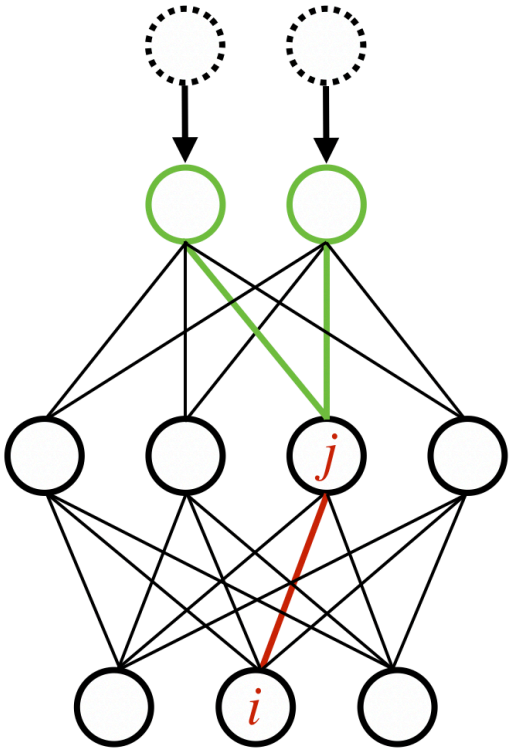
w_{ij}

$i_i^{(p)}$

let's step through the parts

$$\Delta w_{ij} = \epsilon \sum_{p=1}^{P} \left[\sum_{k=1}^{K} \left(t_k^{(p)} - o_k^{(p)} \right) o_k^{(p)} \left(1 - o_k^{(p)} \right) w_{jk} \right] h_j^{(p)} \left(1 - h_j^{(p)} \right) i_i^{(p)}$$

(remember, you don't need to use this in your code
Keras does this automatically for you)



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

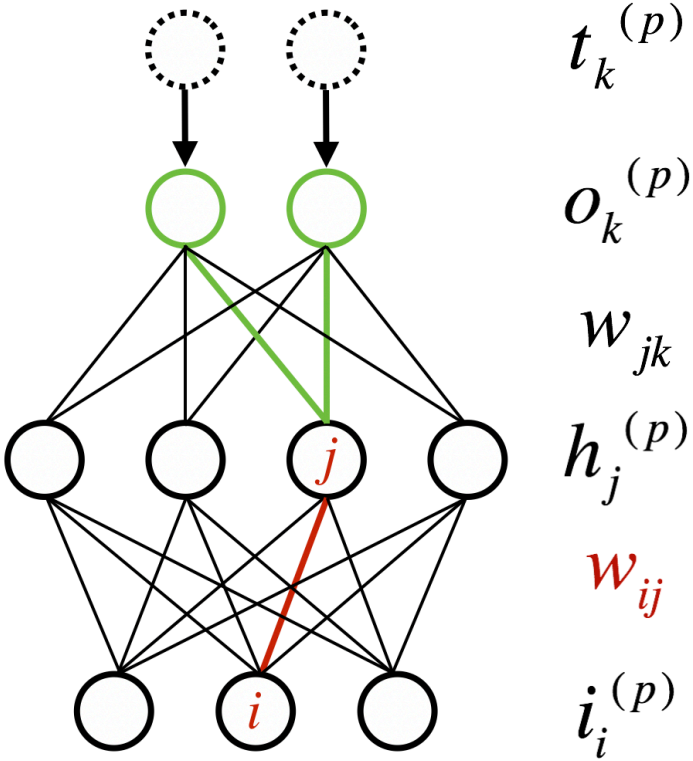
w_{ij}

$i_i^{(p)}$

let's step through the parts

$$\Delta w_{ij} = \epsilon \cancel{\sum_{p=1}^{P}} \left[\sum_{k=1}^{K} \left(t_k^{(p)} - o_k^{(p)} \right) o_k^{(p)} \left(1 - o_k^{(p)} \right) w_{jk} \right] h_j^{(p)} \left(1 - h_j^{(p)} \right) i_i^{(p)}$$

first, let's simplify by making this just stochastic gradient descent (weight update after every training pattern and teacher)



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

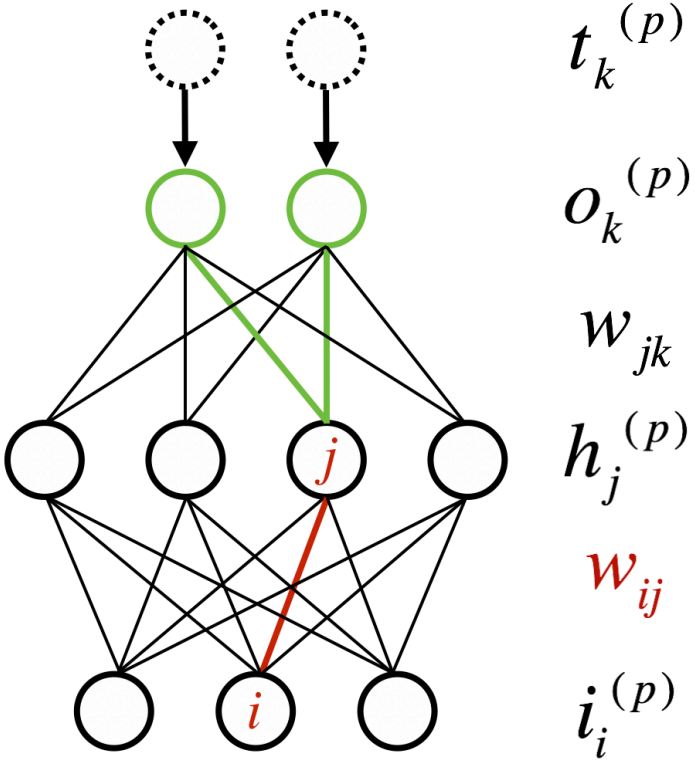
$h_j^{(p)}$

w_{ij}

$i_i^{(p)}$

let's step through the parts

$$\Delta w_{ij} = \epsilon \left[\sum_{k=1}^{K} \left(t_k - o_k\right) o_k \left(1 - o_k\right) w_{jk} \right] h_j \left(1 - h_j\right) i_i$$



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

w_{ij}

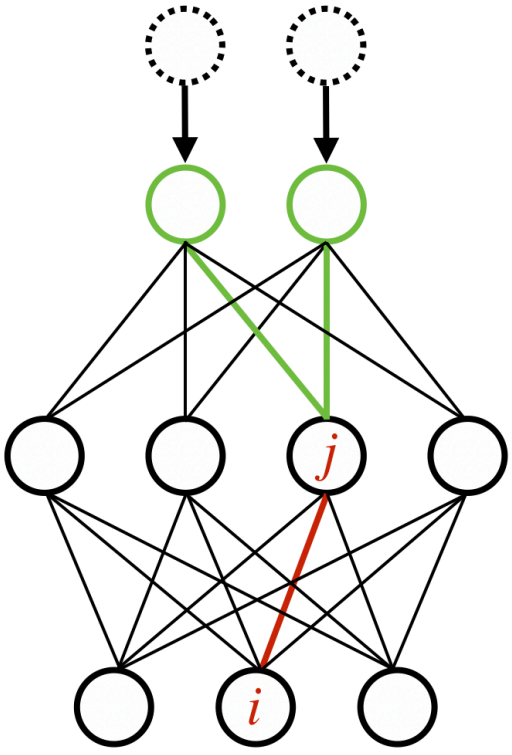
$i_i^{(p)}$

let's step through the parts

$$\Delta w_{ij} = \epsilon \left[\sum_{k=1}^{K} (t_k - o_k)\, o_k (1 - o_k)\, w_{jk} \right] h_j (1 - h_j)\, i_i$$

sigmoidal (logistic) on output nodes

sigmoidal (logistic) on hidden nodes



$t_k^{(p)}$

$o_k^{(p)}$ sigmoidal (logistic) on output nodes

w_{jk}

$h_j^{(p)}$ sigmoidal (logistic) on hidden nodes

w_{ij}

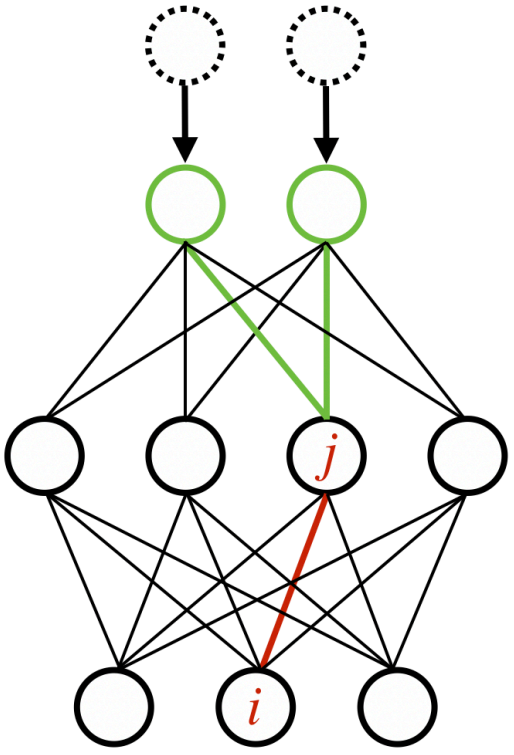
$i_i^{(p)}$

let's step through the parts

$$\Delta w_{ij} = \epsilon \left[\sum_{k=1}^{K} \left(t_k - o_k \right) 1 \; w_{jk} \right] \left(1 - h_j^2 \right) i_i$$

linear on output nodes

tanh on hidden nodes



$t_k^{(p)}$

$o_k^{(p)}$

linear
on output nodes

w_{jk}

$h_j^{(p)}$

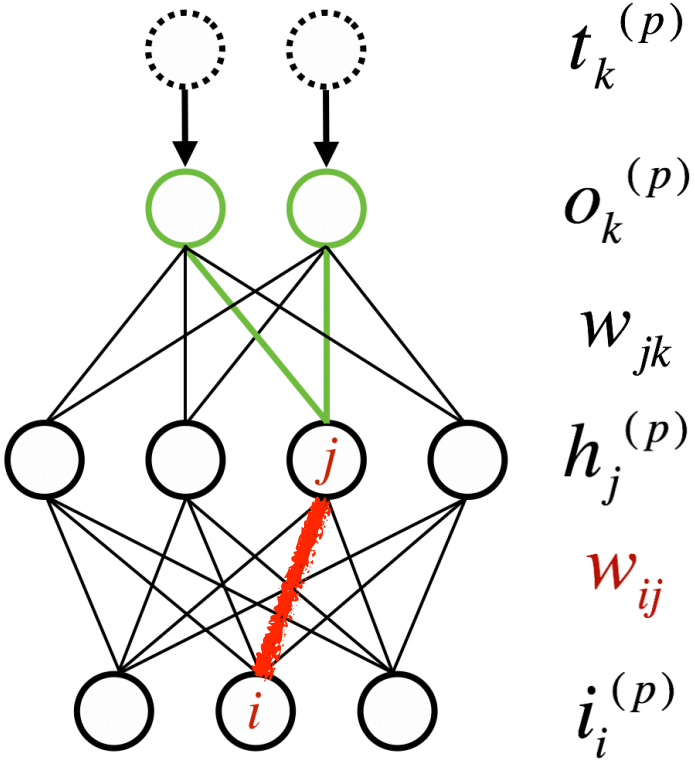
tanh
on hidden nodes

w_{ij}

$i_i^{(p)}$

let's step through the parts

$$\Delta w_{ij} = \epsilon \left[\sum_{k=1}^{K} \left(t_k - o_k\right) o_k \left(1 - o_k\right) w_{jk} \right] h_j \left(1 - h_j\right) i_i$$



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

w_{ij}

$i_i^{(p)}$

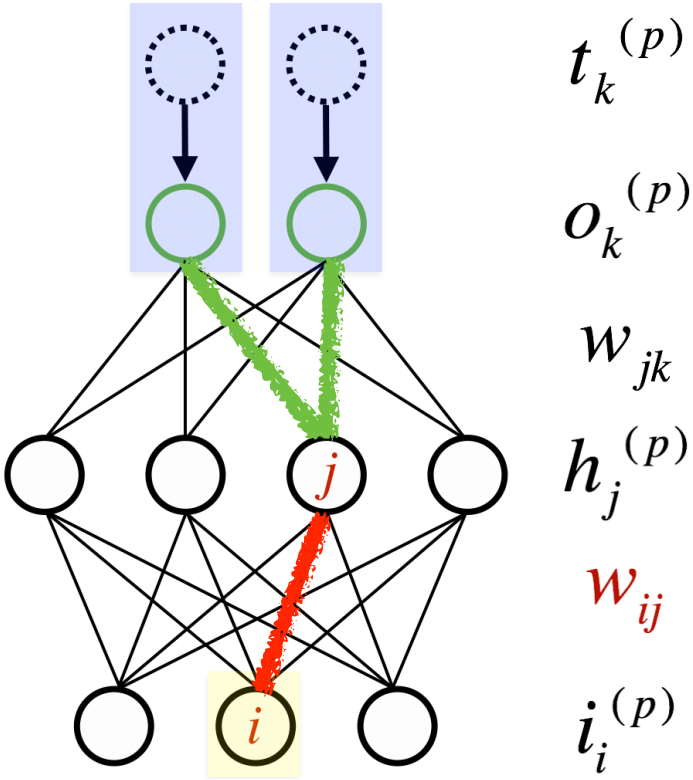
let's step through the parts

$$\Delta w_{ij} = \epsilon \left[\sum_{k=1}^{K} (t_k - o_k)\, o_k (1 - o_k)\, w_{jk} \right] h_j (1 - h_j)\, i_i$$

deviation from output to teacher (magnitude and sign)

scaled by the weight (magnitude and sign)

scaled by the input (magnitude and sign)



$t_k^{(p)}$

$o_k^{(p)}$

w_{jk}

$h_j^{(p)}$

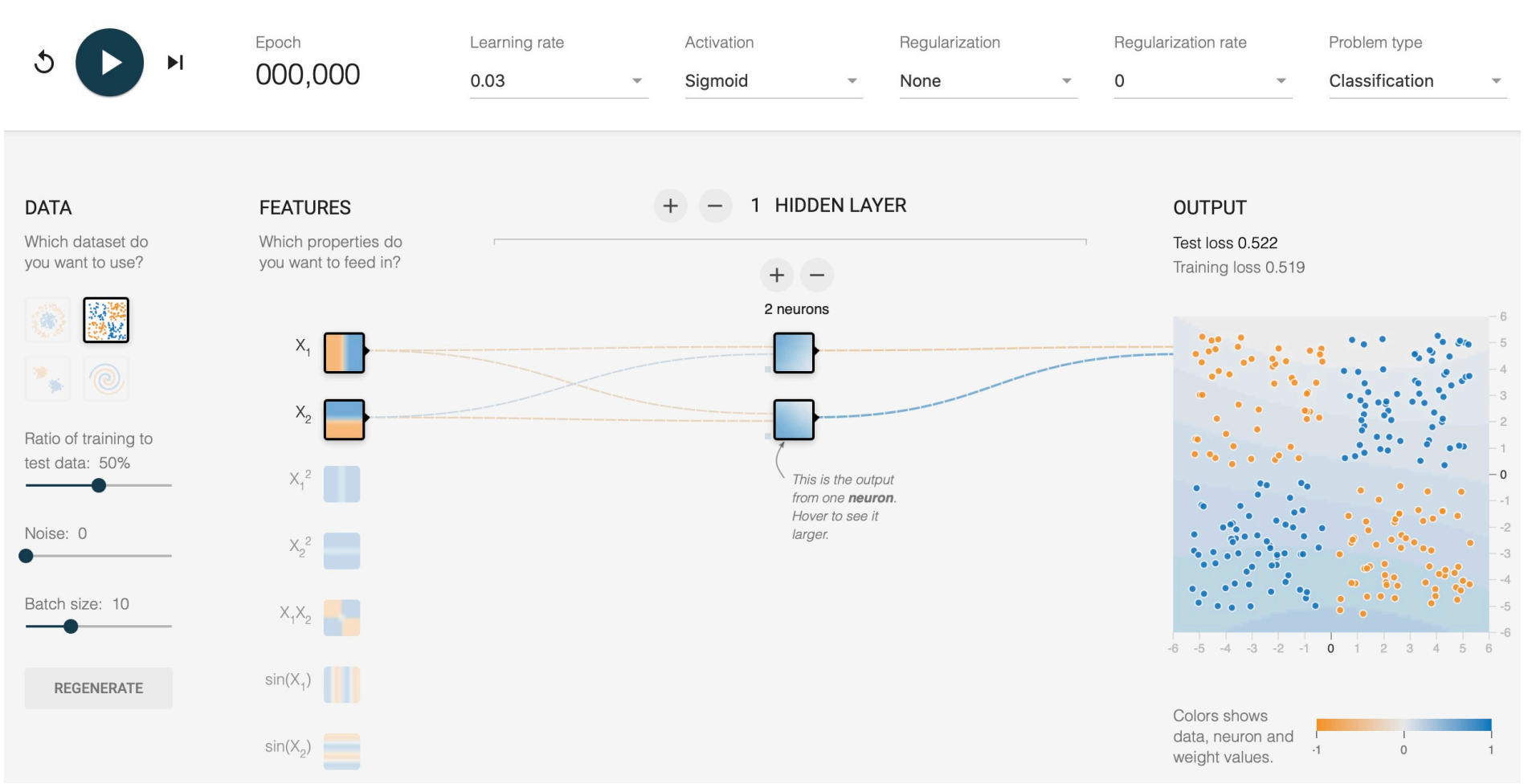
w_{ij}

$i_i^{(p)}$

backpropagation or "backprop" is
backward propagation of errors

https://playground.tensorflow.org/

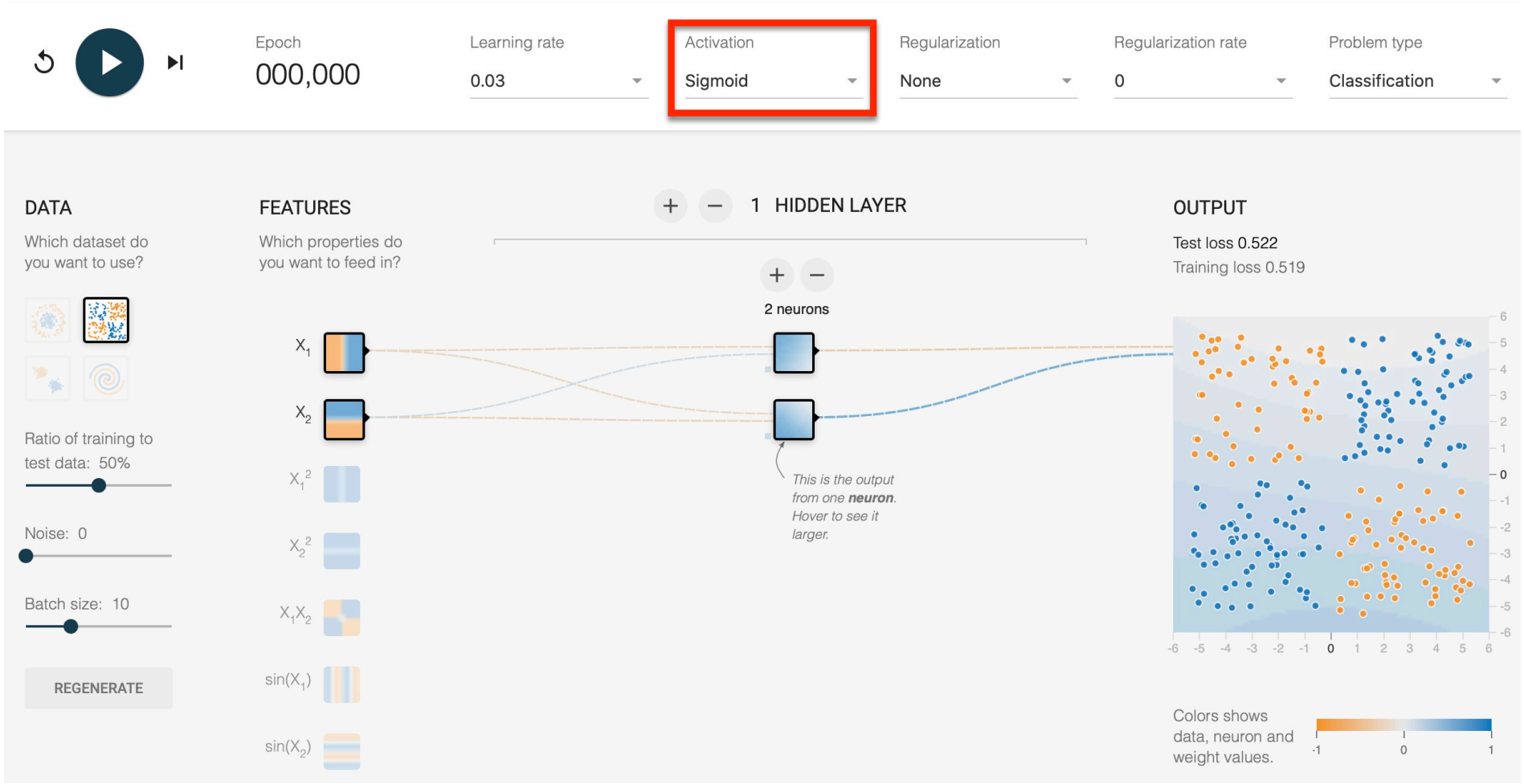
we saw before how a network with a hidden layer can learn the XOR problem



https://playground.tensorflow.org/

activation function needs to be non-linear*

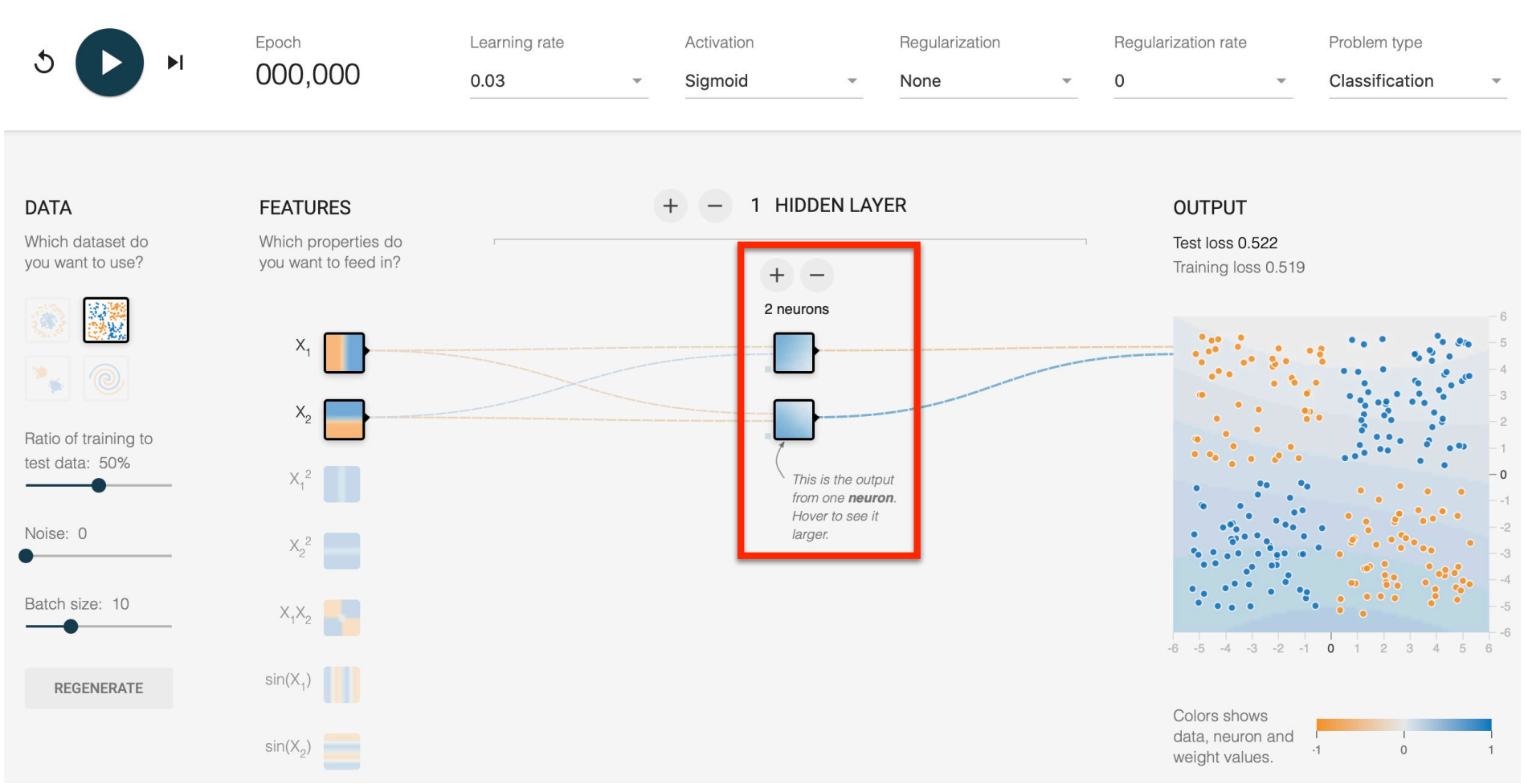
*note that ReLU can sometimes get "stuck" on a tiny network like this



https://playground.tensorflow.org/

with only two hidden layers, you can hit a long plateau in the error surface

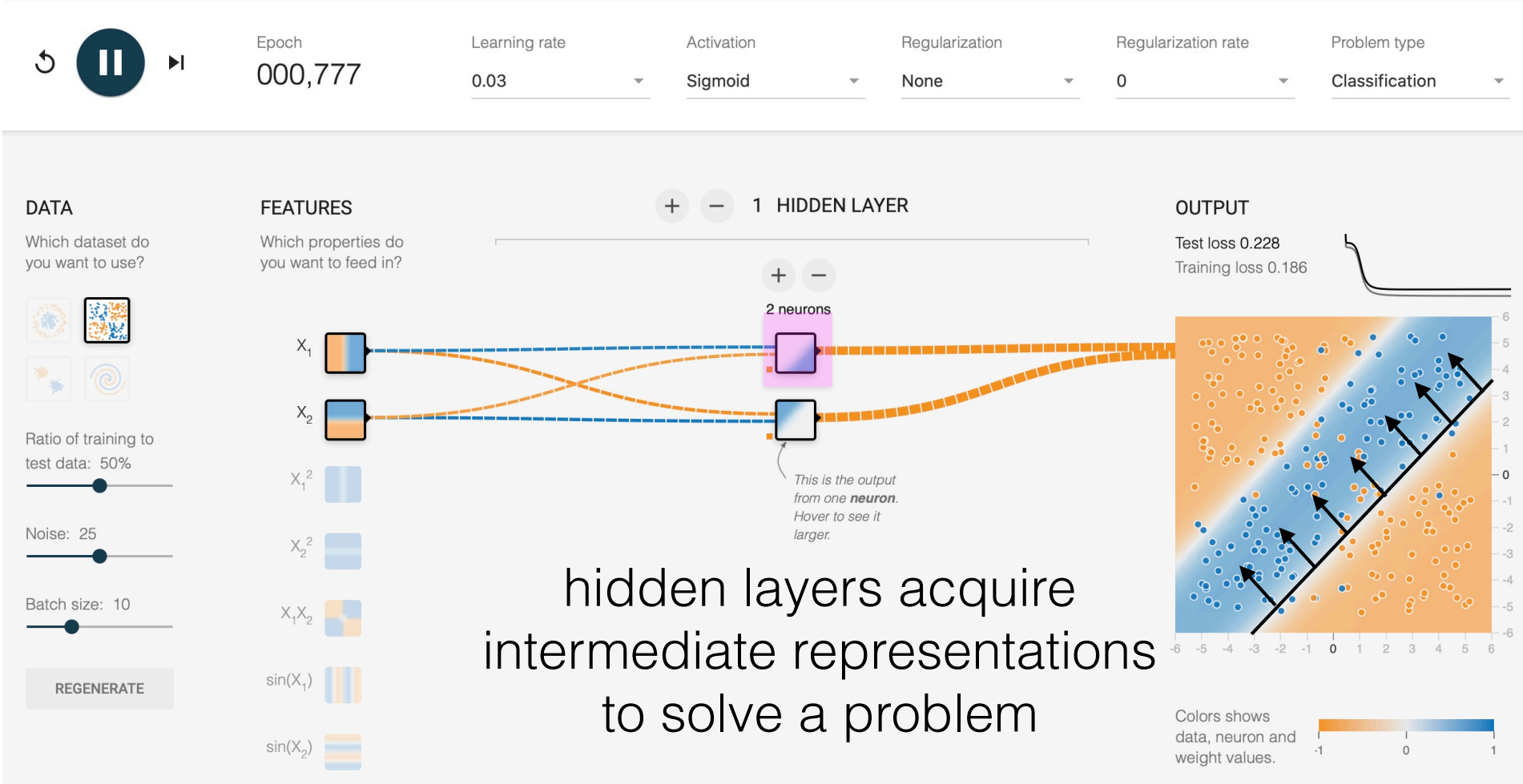
three hidden nodes converges more efficiently



https://playground.tensorflow.org/

with only two hidden layers, you can hit a long plateau in the error surface

three hidden nodes converges more efficiently

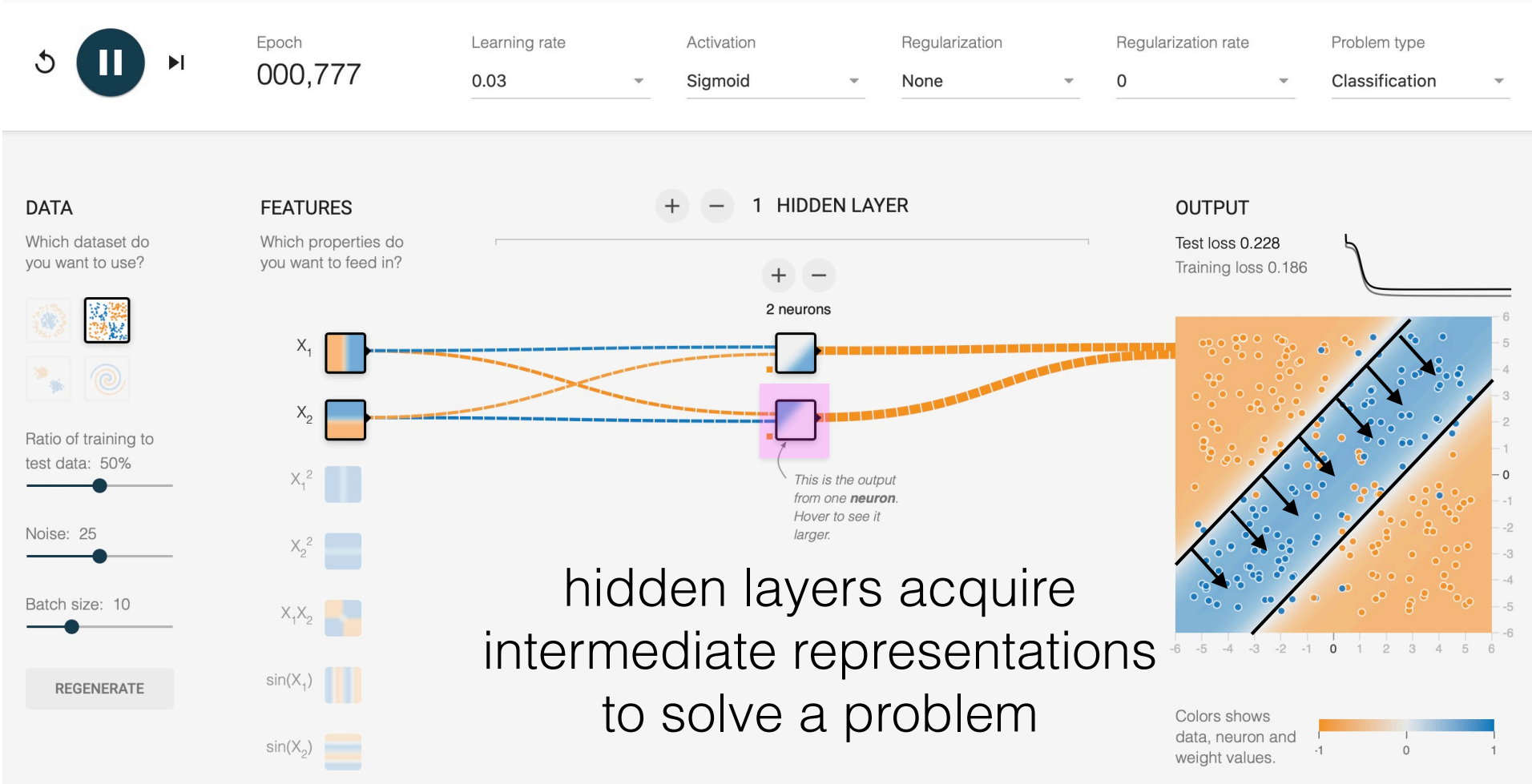


hidden layers acquire intermediate representations to solve a problem

https://playground.tensorflow.org/

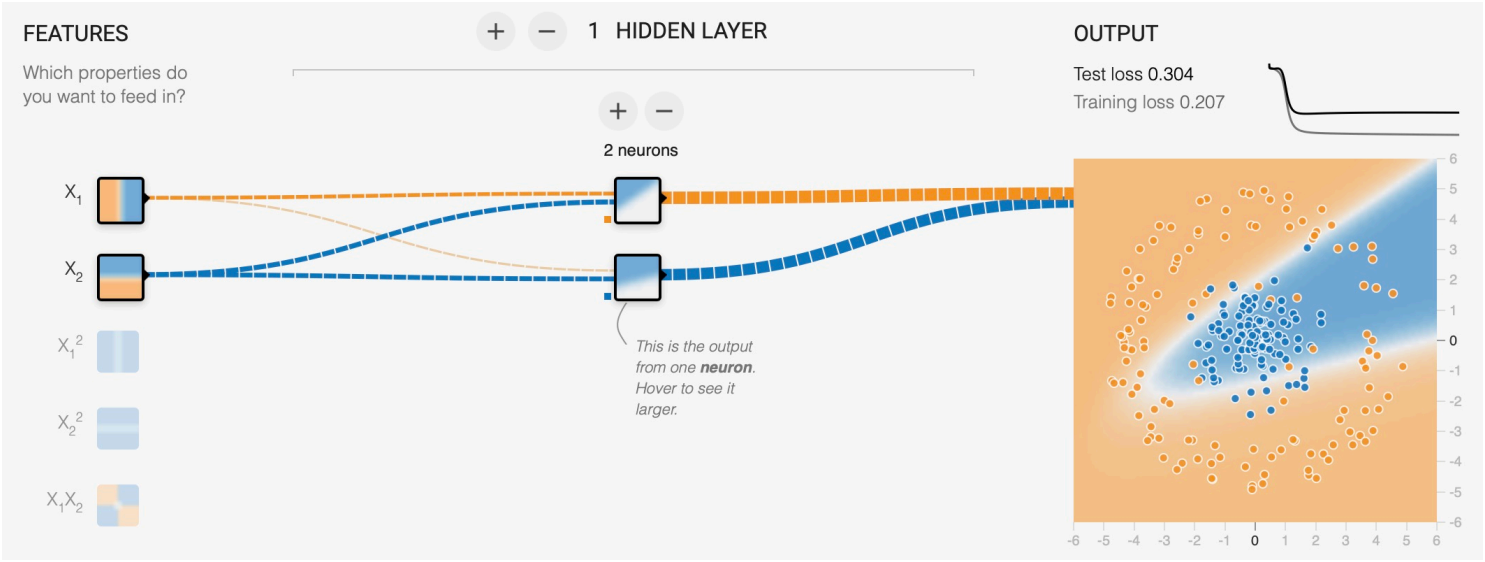
with only two hidden layers, you can hit a long plateau in the error surface

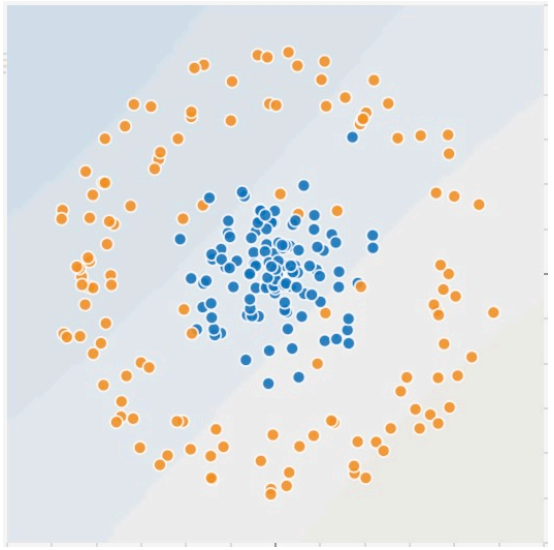
three hidden nodes converges more efficiently

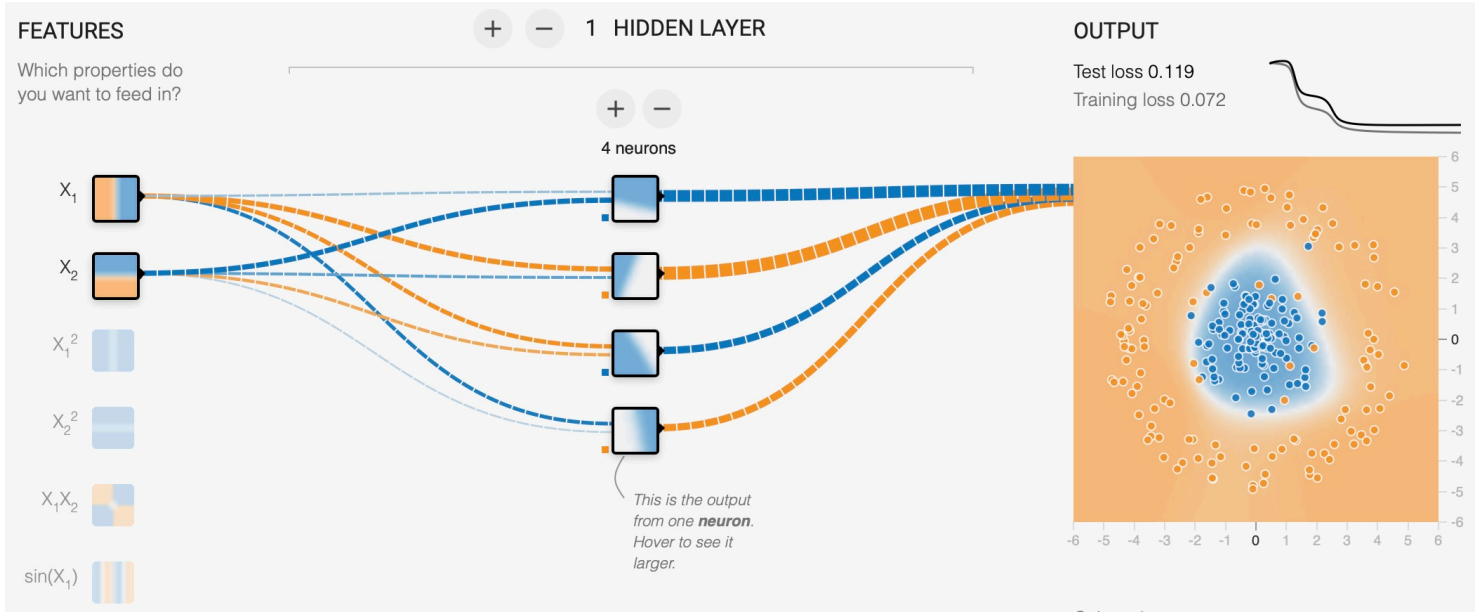


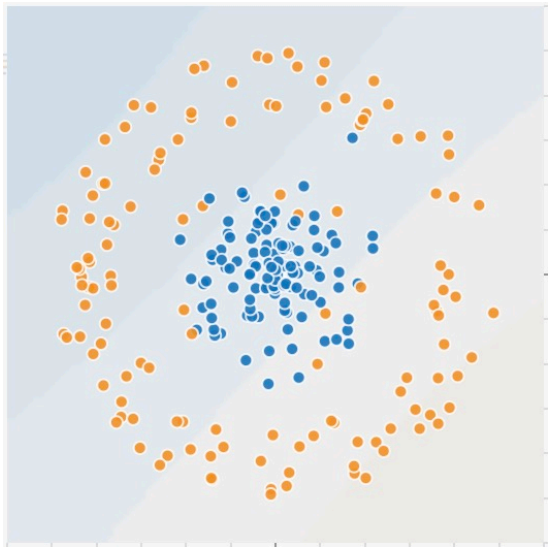
hidden layers acquire intermediate representations to solve a problem

hidden layers acquire intermediate representations to solve a problem

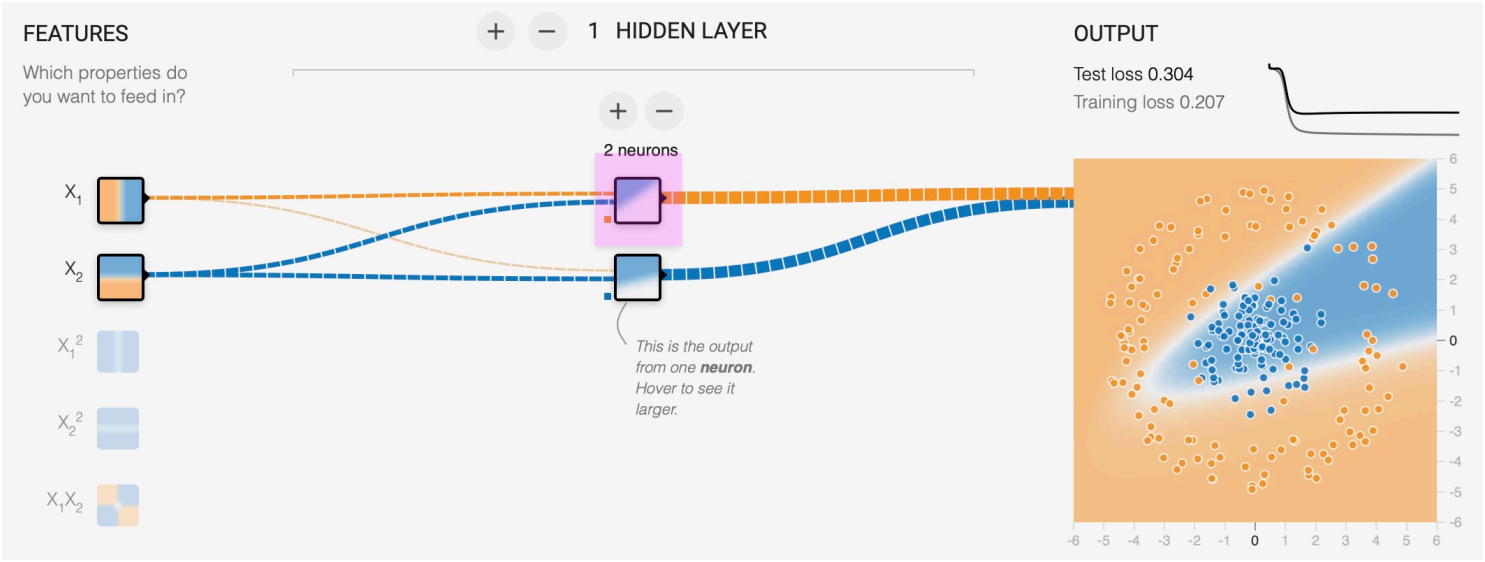


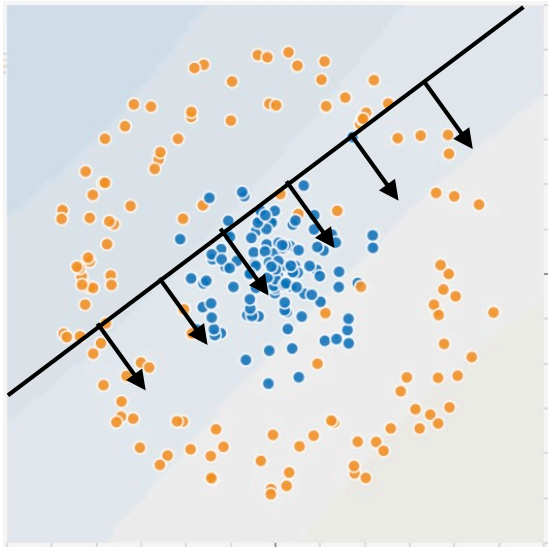


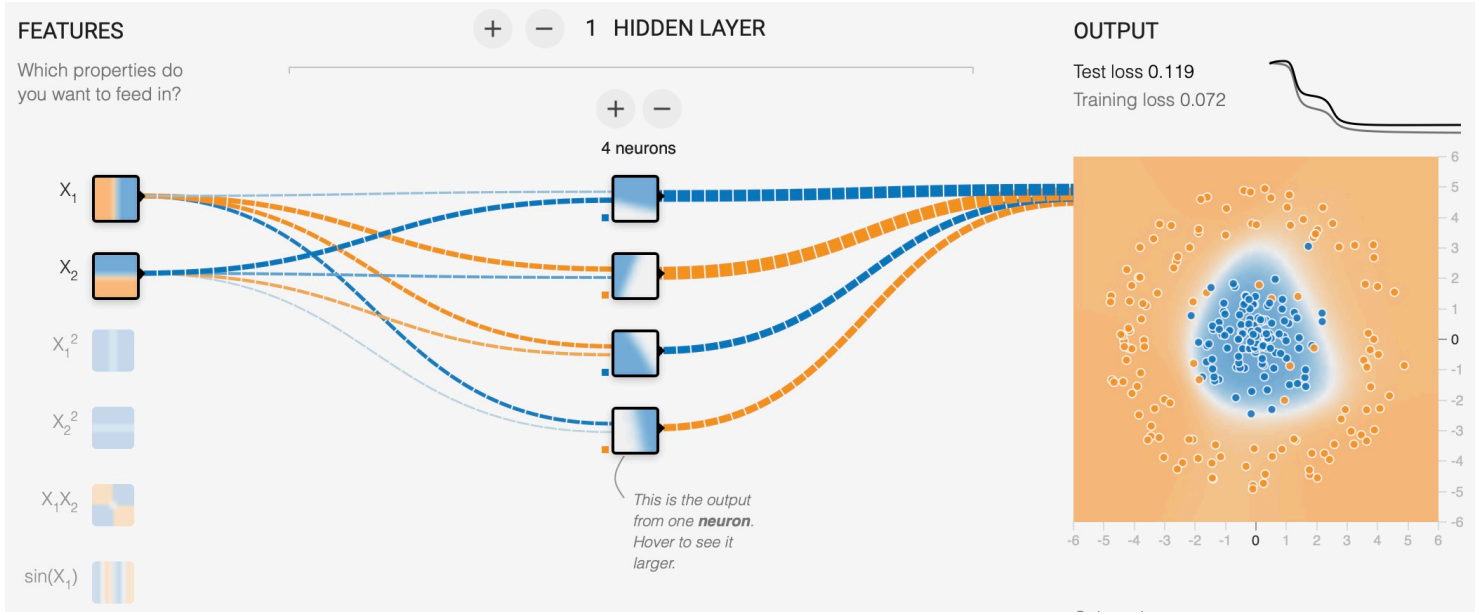


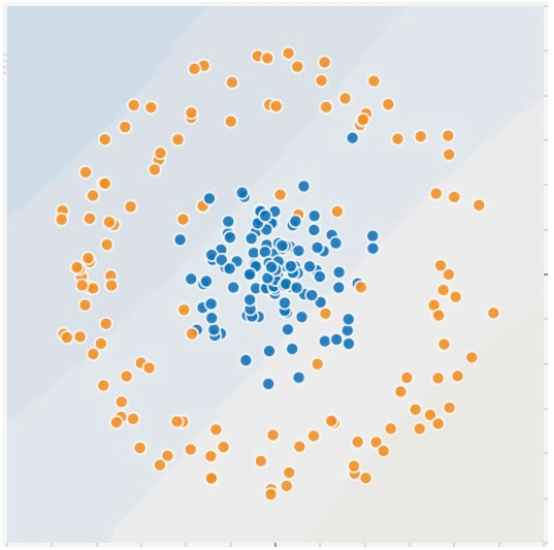


hidden layers acquire intermediate representations to solve a problem

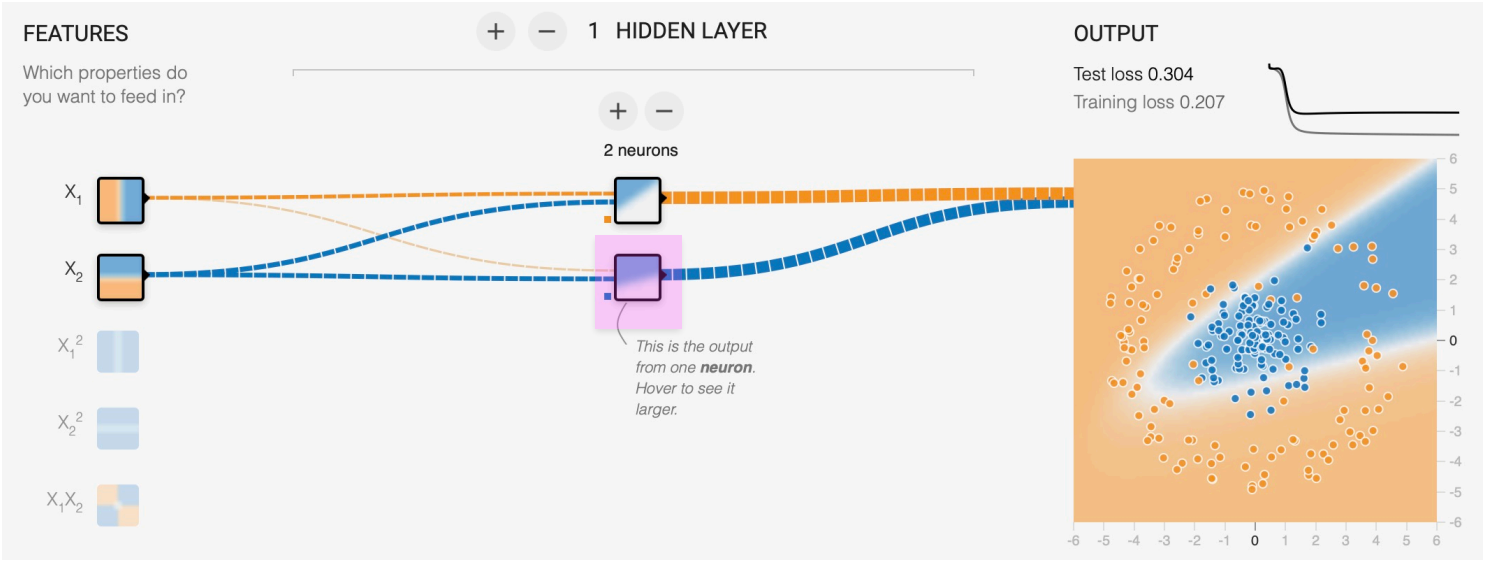


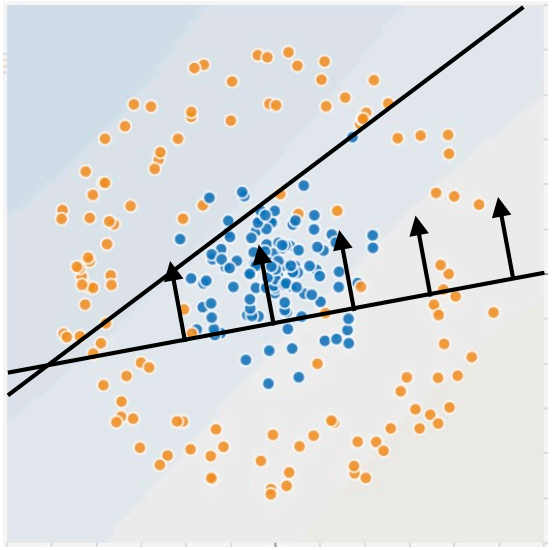


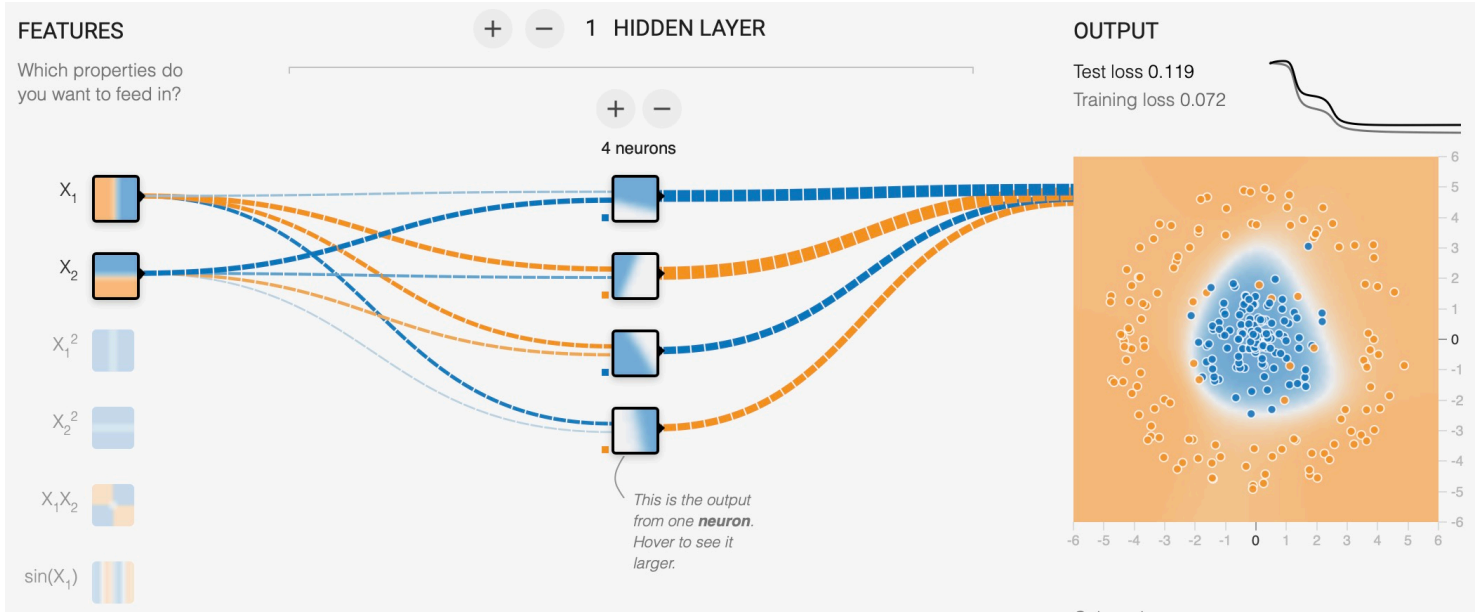


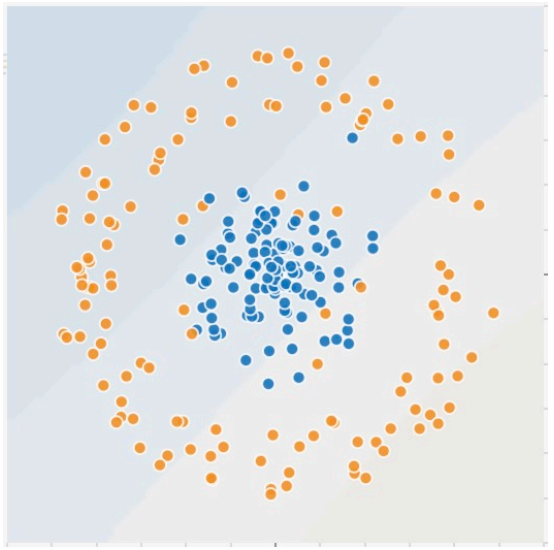


hidden layers acquire intermediate representations to solve a problem

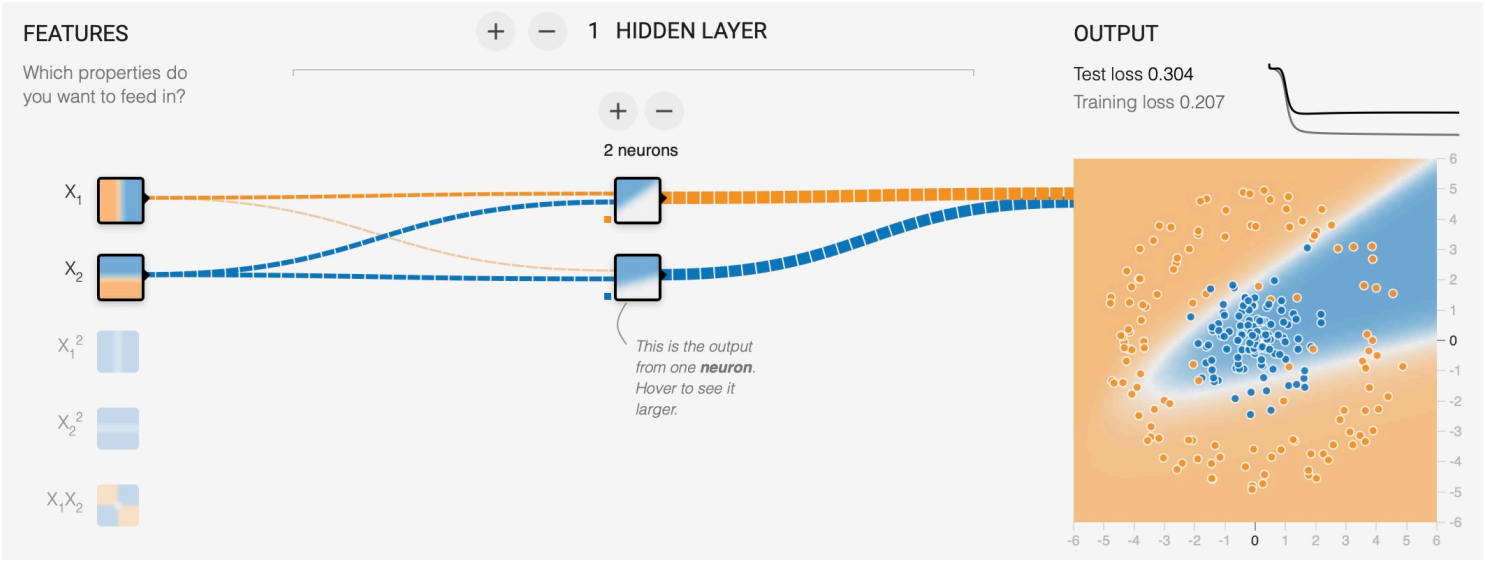


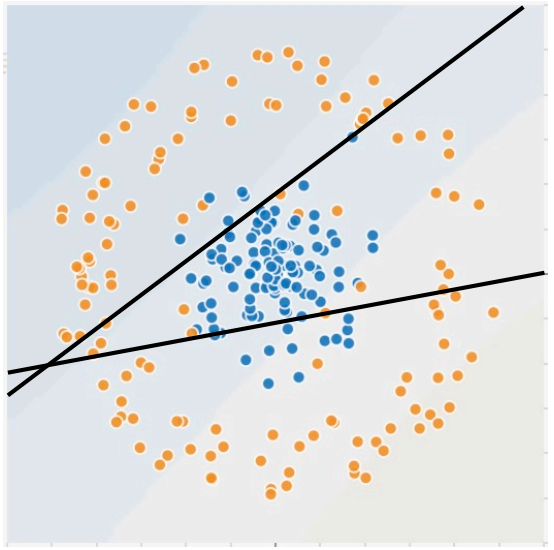


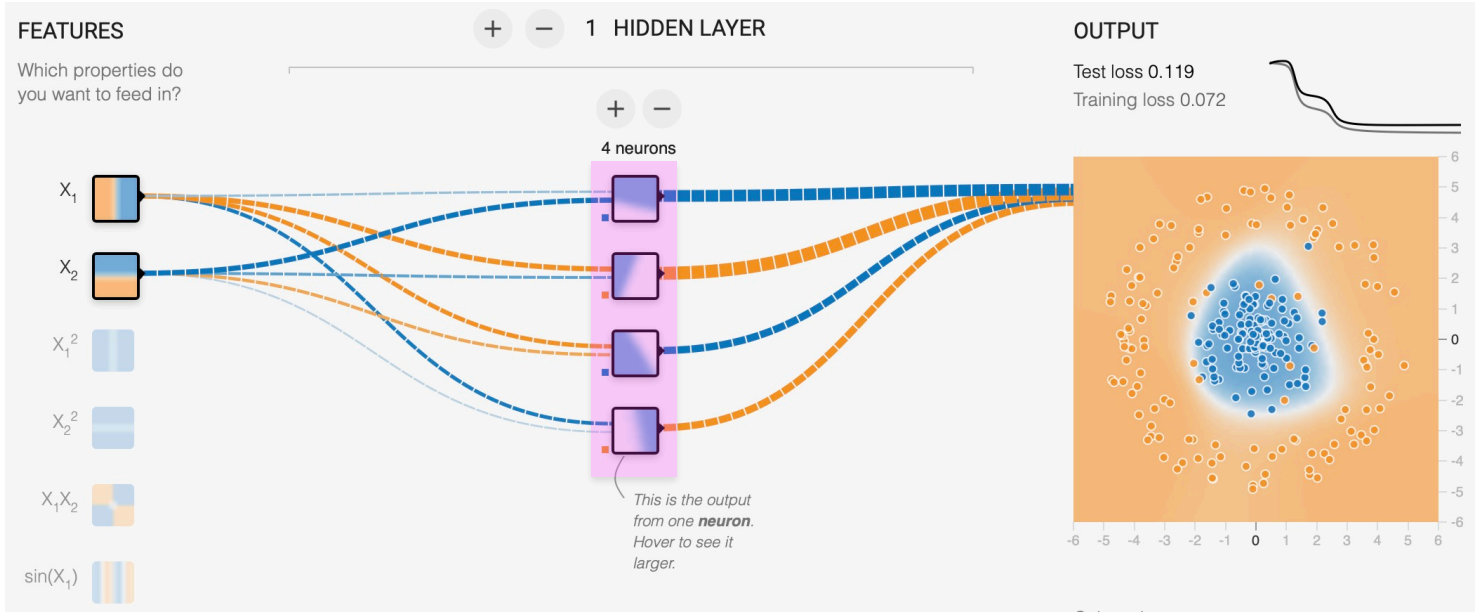


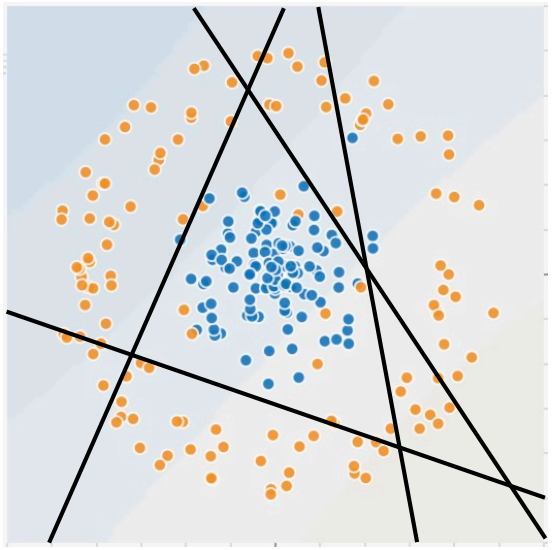


hidden layers acquire intermediate representations to solve a problem





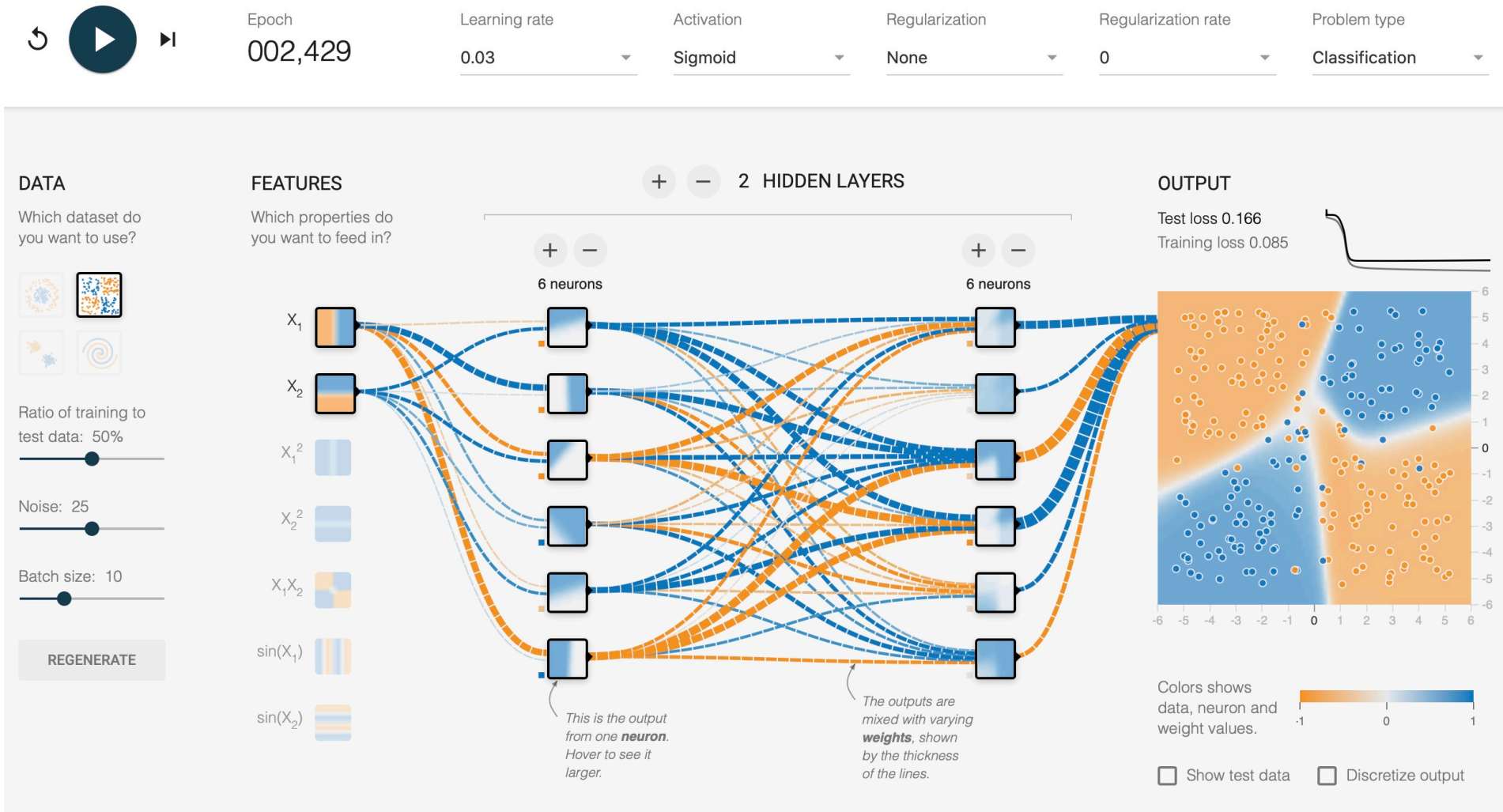




https://playground.tensorflow.org/

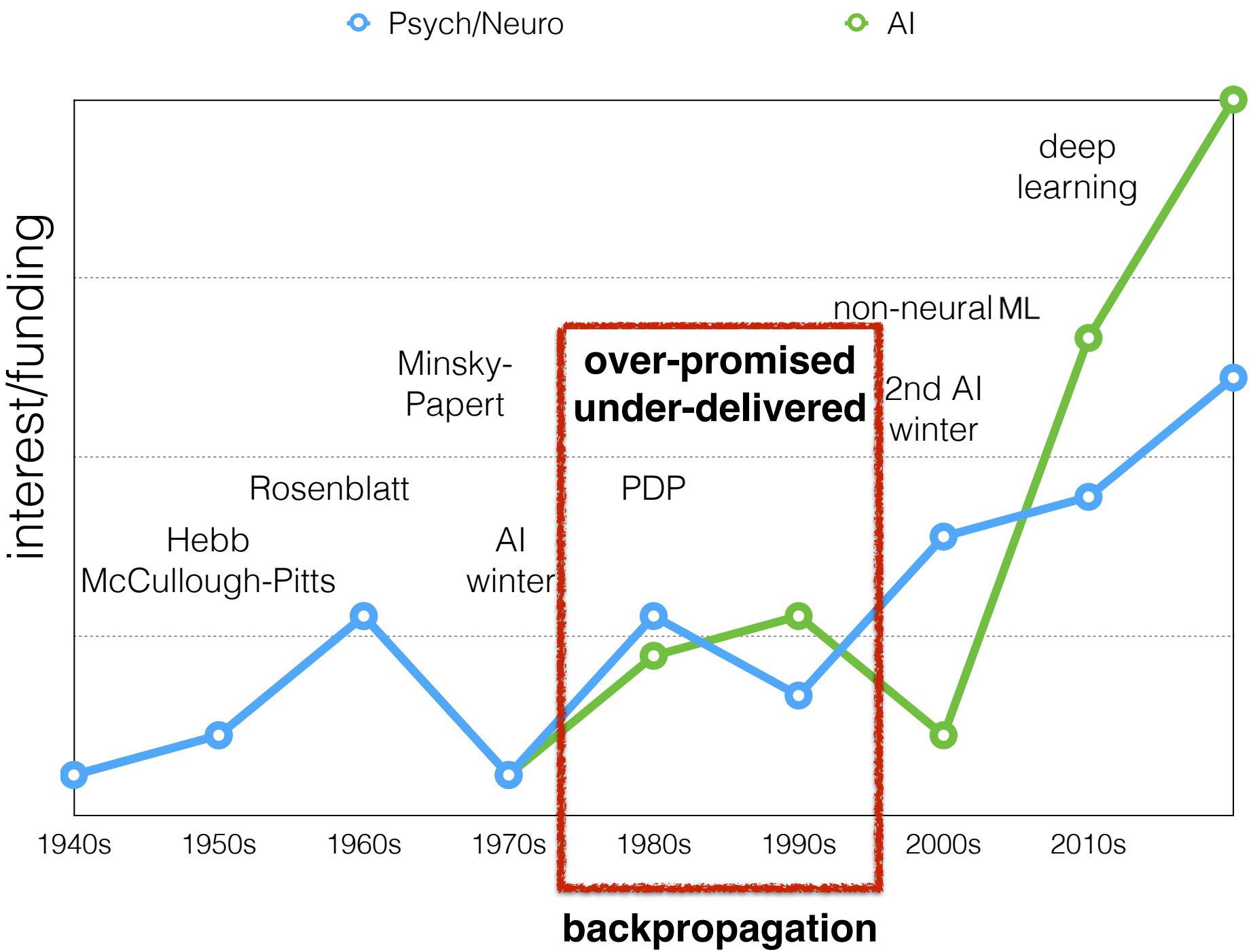
a network with too many layers and too many hidden nodes per layer

for a relatively simple problem like this



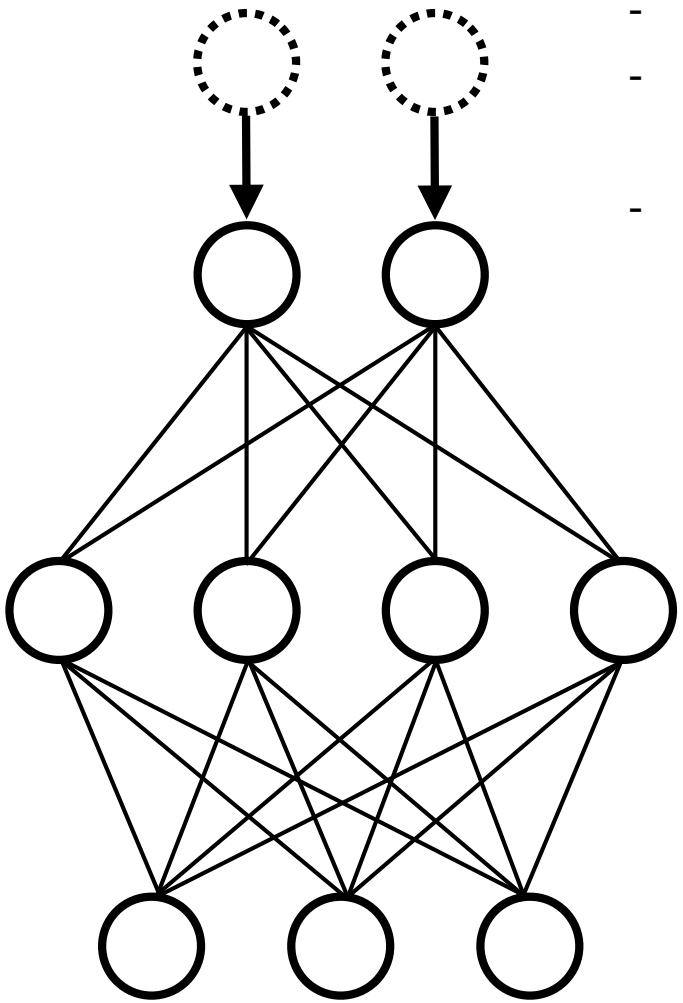
we'll talk about model complexity and over-fitting in coming lectures

a brief history of neural network models



Hidden Layers

Proven that with only a single layer of **nonlinear** hidden units, **given enough of them**, you can **approximate essentially any function*** to an arbitrary degree of precision (Hornik, Stinchcombe, & White, 1989).

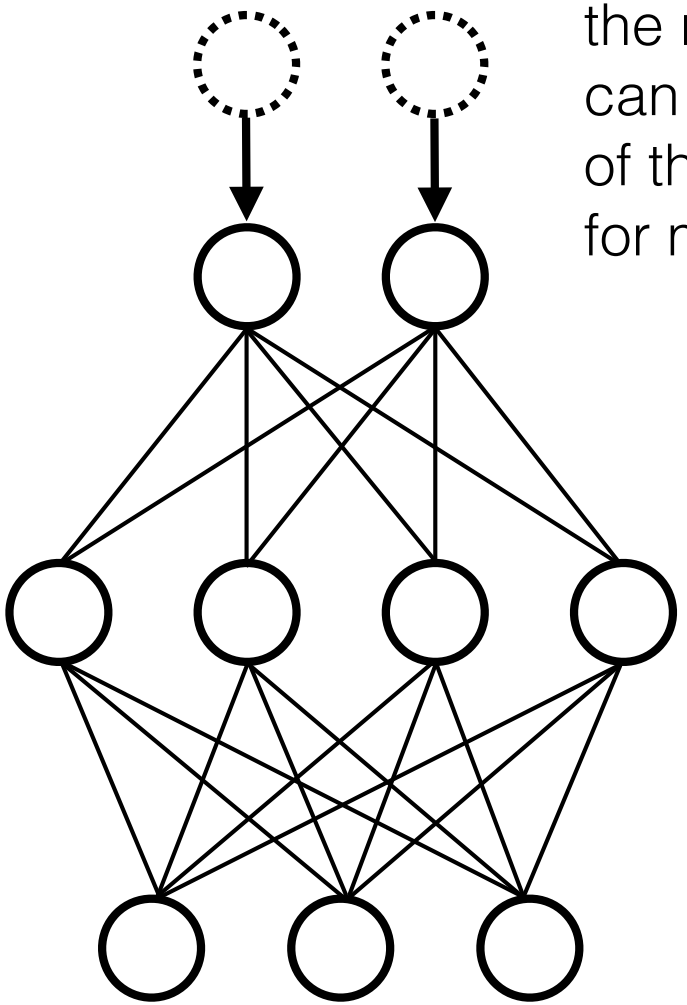


* **function** here is broad

- not just a mathematical function
- the mapping between images you see and what they are called is a function (vision)
- or between sounds you hear and words you recognize is a function (speech recognition)

Hidden Layers

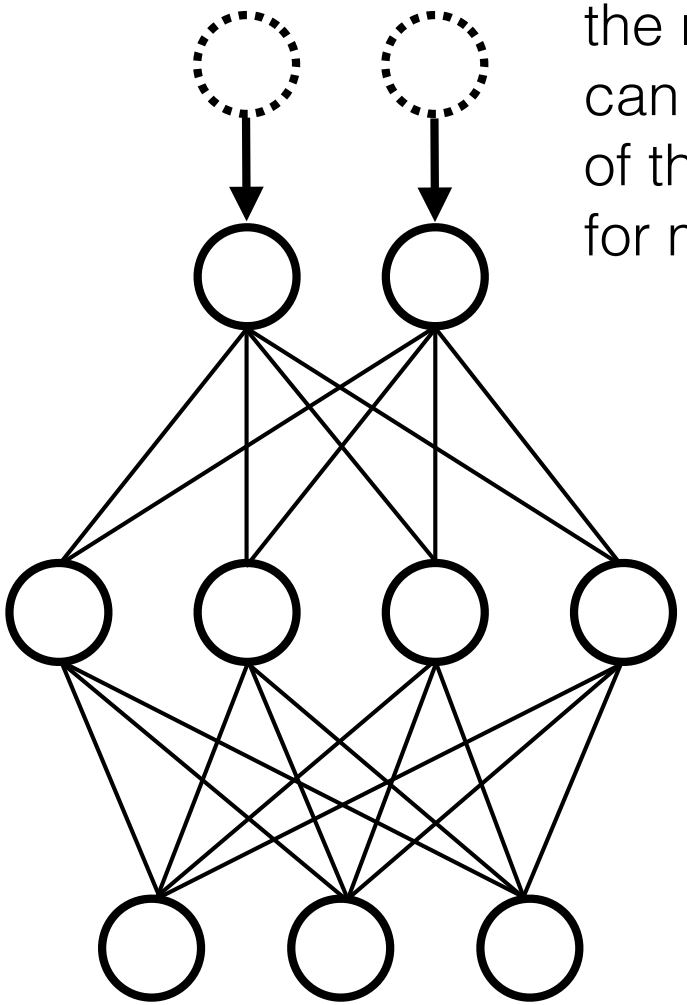
Proven that with only a single layer of **nonlinear** hidden units, **given enough of them**, you can **approximate essentially any function** to an arbitrary degree of precision (Hornik, Stinchcombe, & White, 1989).



The challenge is that with only a single hidden layer (a single layer of intermediate representations) the number of hidden nodes (and hence weights) can grow astronomically large and the optimization of those weights can become impossible to solve for most complex problems (like visual recognition).

Hidden Layers

Proven that with only a single layer of **nonlinear** hidden units, **given enough of them**, you can **approximate essentially any function** to an arbitrary degree of precision (Hornik, Stinchcombe, & White, 1989).

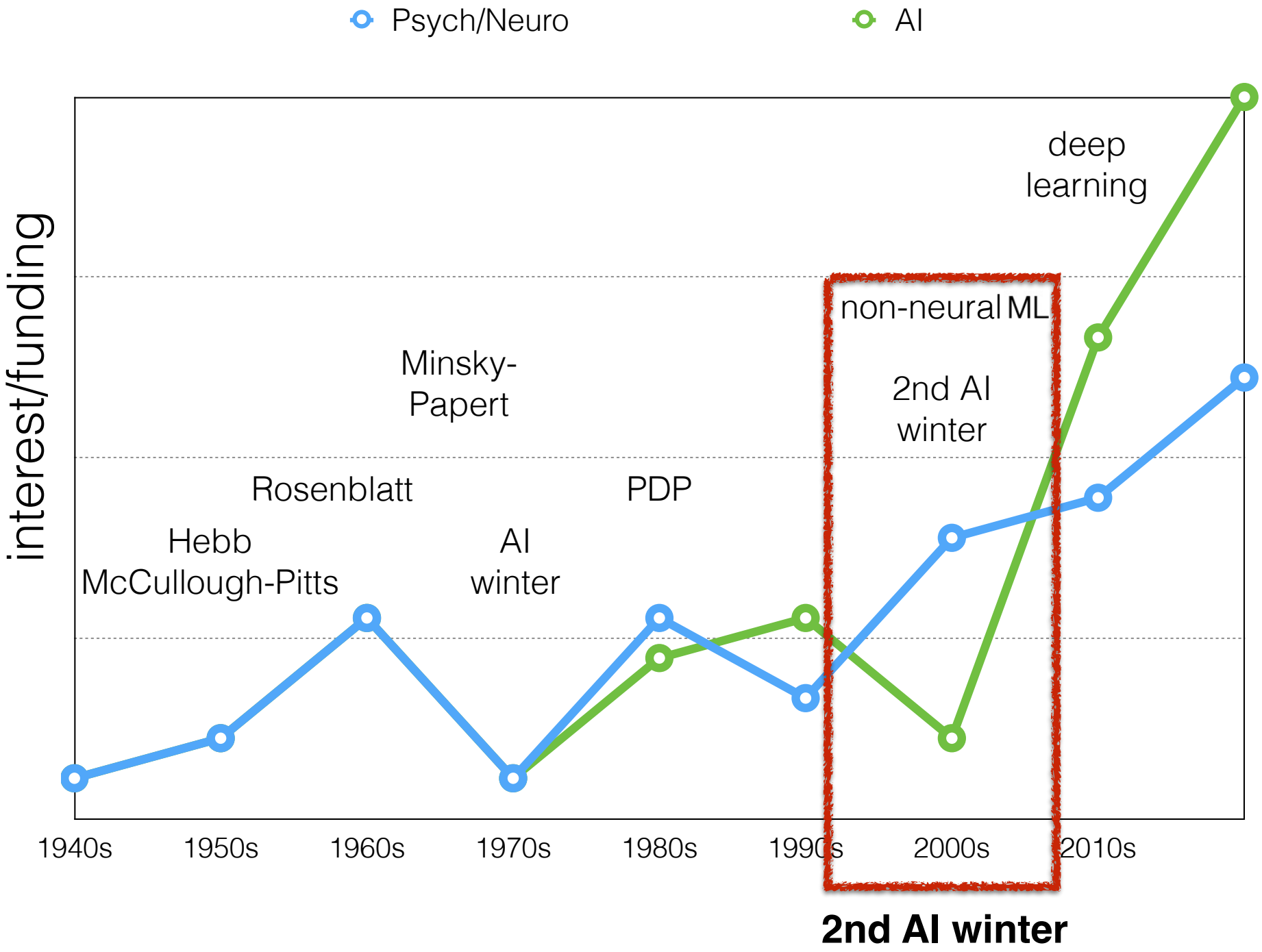


The challenge is that with only a single hidden layer (a single layer of intermediate representations) the number of hidden nodes (and hence weights) can grow astronomically large and the optimization of those weights can become impossible to solve for most complex problems (like visual recognition).

over-promised
under-delivered

And while a network with only a single hidden layer might learn that function, it might not generalize appropriately beyond the training data (we'll unpack this concept later).

a brief history of neural network models



The Economist, 7 June 2007: *"[Investors] were put off by the term 'voice recognition' which, like 'artificial intelligence', is associated with systems that have all too often failed to live up to their promises."*

Patty Tascarella in Pittsburgh Business Times, 2006: *"Some believe the word 'robotics' actually carries a stigma that hurts a company's chances at funding."*

John Markoff in the New York Times, 2005: *"At its low point, some computer scientists and software engineers avoided the term artificial intelligence for fear of being viewed as wild-eyed dreamers."*

- quotes from Wikipedia