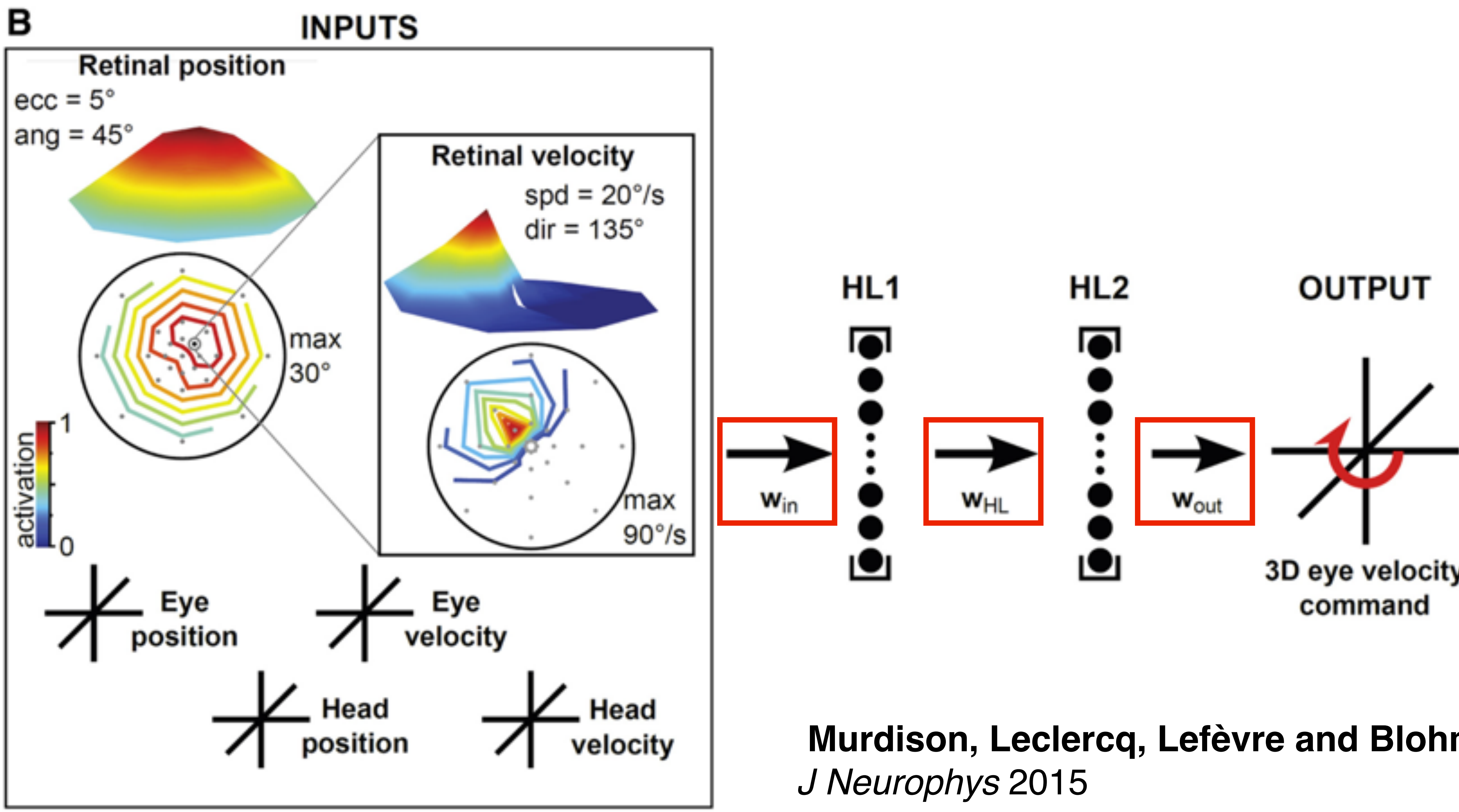
Rate-based artificial neural networks and error backpropagation learning

Scott Murdison
Machine learning journal club
May 16, 2016



Murdison, Leclercq, Lefèvre and Blohm
J Neurophys 2015

Neural networks??? Why would we use those?!

Some problems are just too complicated (i.e. nonlinear) to solve with simple programming…

Computer vision/audition applications:

- digit recognition
- facial recognition
- reading
- speech recognition
- object recognition

Others:

- detection of medical disorders
- industrial process control
- stock market predictions
- the list goes on…

Could theoretically write a program to solve each of these problems, but the rules governing such a program would be incredibly complicated!

Geoff Hinton sweet intro vid

Neural networks??? Why would we use those?!

Some problems are just too complicated to solve with simple programming…

Computer vision/audition applications:

- facial recognition

> In the Fall of 1992, for a class project in Artificial Intelligence, I designed a neural network to locate facial features in images. The one hundred images I used came from the underclassmen section of the 1987 University High School yearbook. They were scanned in at 96 by 128 resolution. I set four of the images aside to comprise the testing set, and for the remaining ninety-six I manually specified the coordinates of the left eye, right eye, nose, and mouth.

Paul Debevec. *A Neural Network for Facial Feature Location*. UC Berkeley CS283 Project Report, December 1992. http://www.debevec.org/FaceRecognition/

Neural networks??? Why would we use those?!

Some problems are just too complicated to solve with simple programming…

Computer vision/audition applications:

- facial recognition

manually located left eye, nose and mouth

polar-transformed images around each feature

subsampled images to feed to neural network **…Why?**

A. Avoids local minima

B. 4**M**B RAM in 1992 = $150 USD

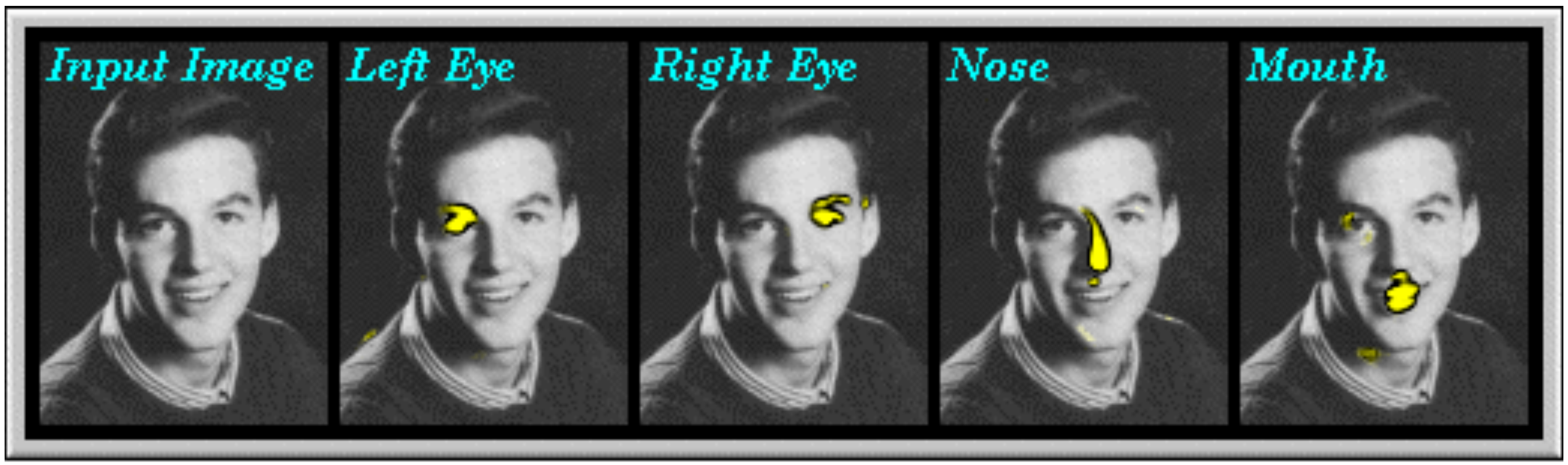
Paul Debevec. *A Neural Network for Facial Feature Location*. UC Berkeley CS283 Project Report, December 1992. http://www.debevec.org/FaceRecognition/

Neural networks??? Why would we use those?!

Some problems are just too complicated to solve with simple programming…

Computer vision/audition applications:

- facial recognition



Neural network outputs for a previously unseen face

After training a simple, feedforward network with backprop using yearbook photos it could successfully detect each eye, nose and mouth in previously unseen photo!

Paul Debevec. *A Neural Network for Facial Feature Location*. UC Berkeley CS283 Project Report, December 1992. http://www.debevec.org/FaceRecognition/

The choice for rate-based over spiking for machine learning

spiking

Pros

neuron-level resolution
time-resolved
  (spiking dynamics, variability)
several levels of abstraction available
  (H-H, leaky integrate-and-fire, Izhikevich)

Cons

computationally expensive for large populations of neurons
intractable for simulations of whole brain areas
not obviously useful for machine learning

rate-based

Pros

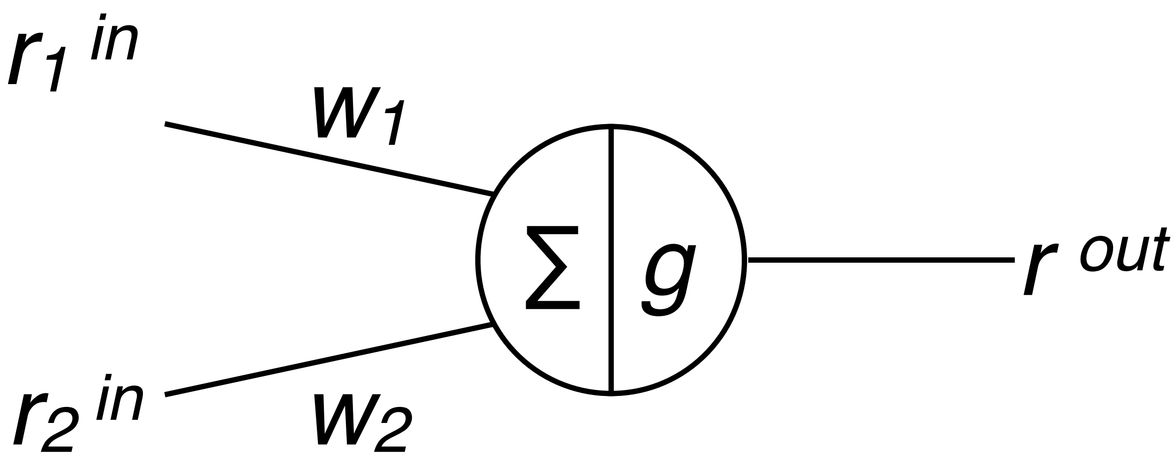
describes average firing of functional populations of neurons
computationally cheap
can address network structure/function
mathematical simplicity
Universal function approximator

Cons

lose benefits of spiking spatiotemporal resolution
  averages over timing/dynamics/variability/etc.
biological analogs not always obvious

General architecture

Simplest network is the perceptron



Transfer function *g*(∑)
general engineering term used to describe output of some processing unit as a function of input

if $r_i^{in} = x_i$ and $r^{out} = y$

then $y = g(w_1 \cdot x_1 + w_2 \cdot x_2)$

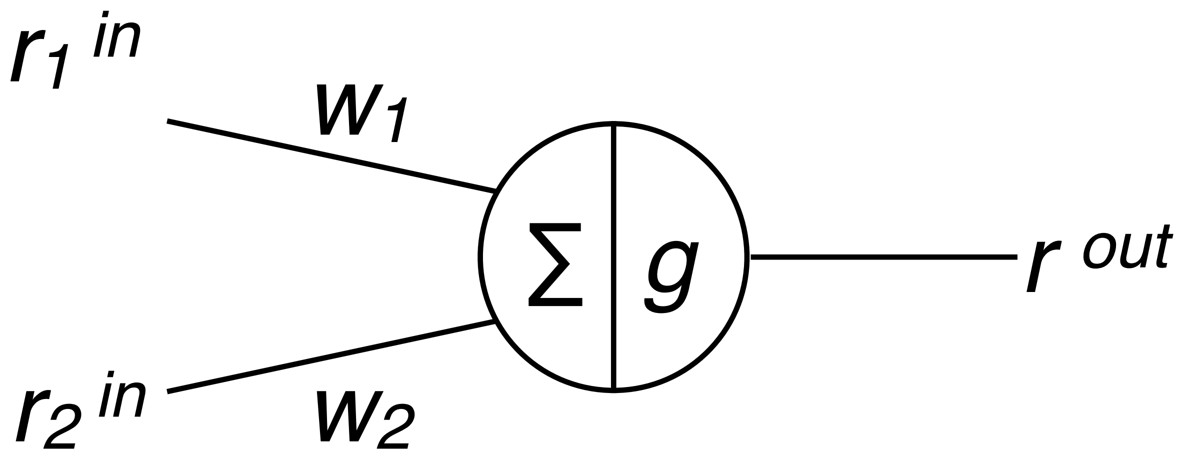
and if $g(\sum) = \sum$ (*i.e. g* is purely linear)

then $y = w_1 \cdot x_1 + w_2 \cdot x_2$

General single-layer mapping

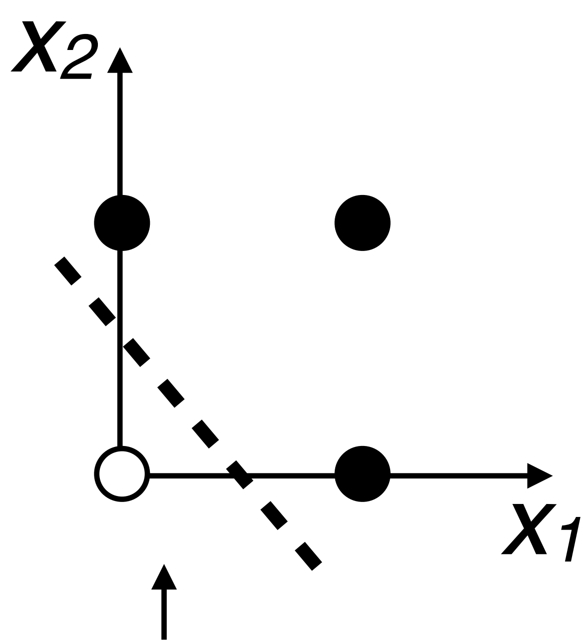
$$r_i^{out} = g(\sum_i w_{ij} r_j^{in}) \Leftrightarrow \mathbf{r}^{out} = g(\mathbf{w}\mathbf{r}^{in})$$

The XOR problem



Boolean OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

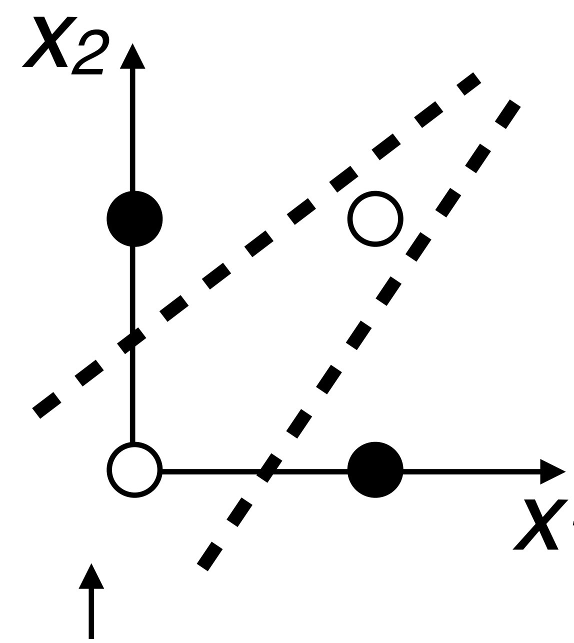


linearly separable: can be easily separated with single threshold!

$$g(x) = \begin{cases} 1 & \text{if } x > \Theta \\ 0 & \text{else} \end{cases}$$

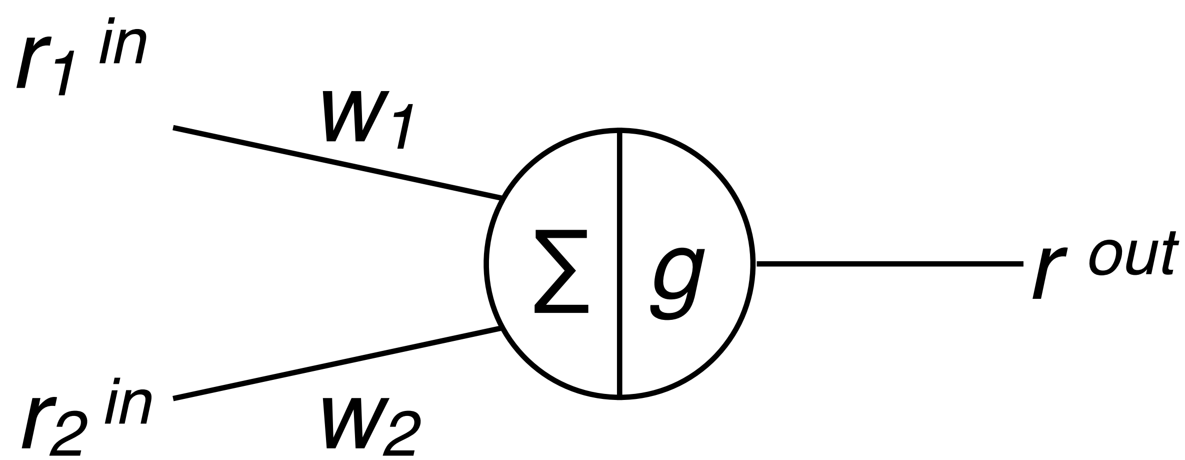
XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	**0**



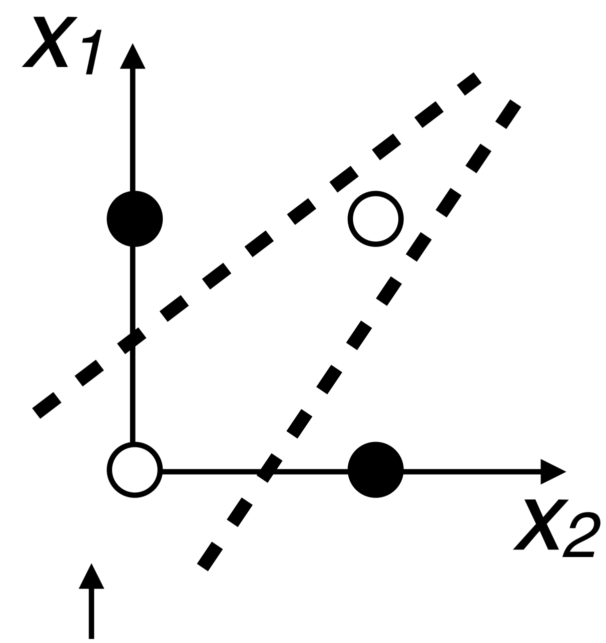
not linearly separable: more complicated!

The XOR problem



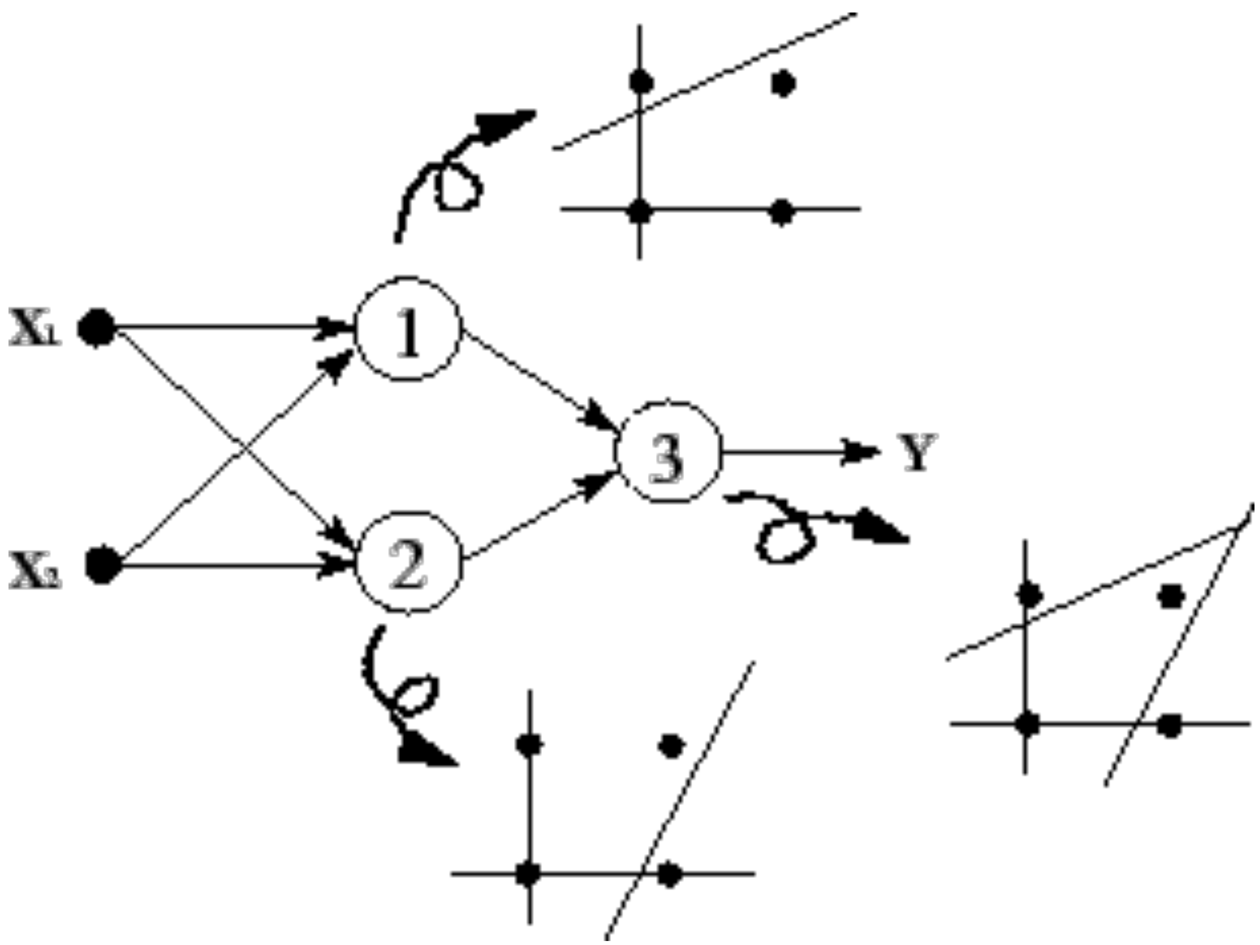
XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	**0**



not linearly separable: more complicated!

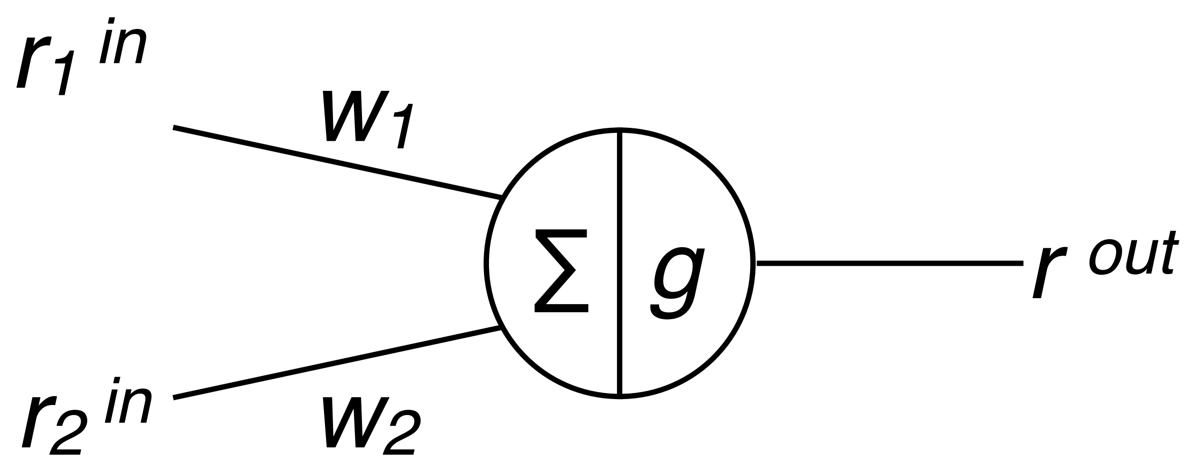
becomes



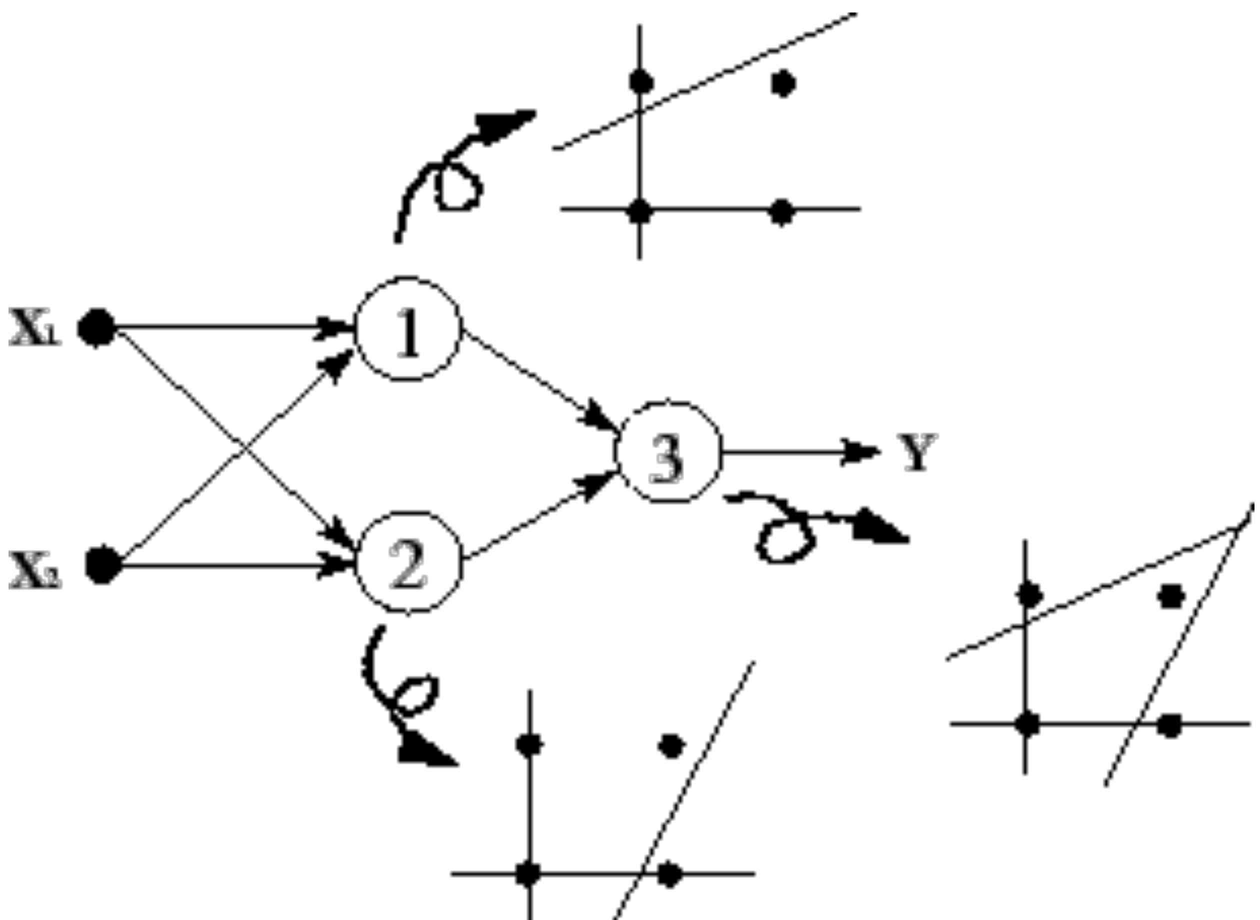
With enough hidden units, multilayer perceptron is the universal function approximator!

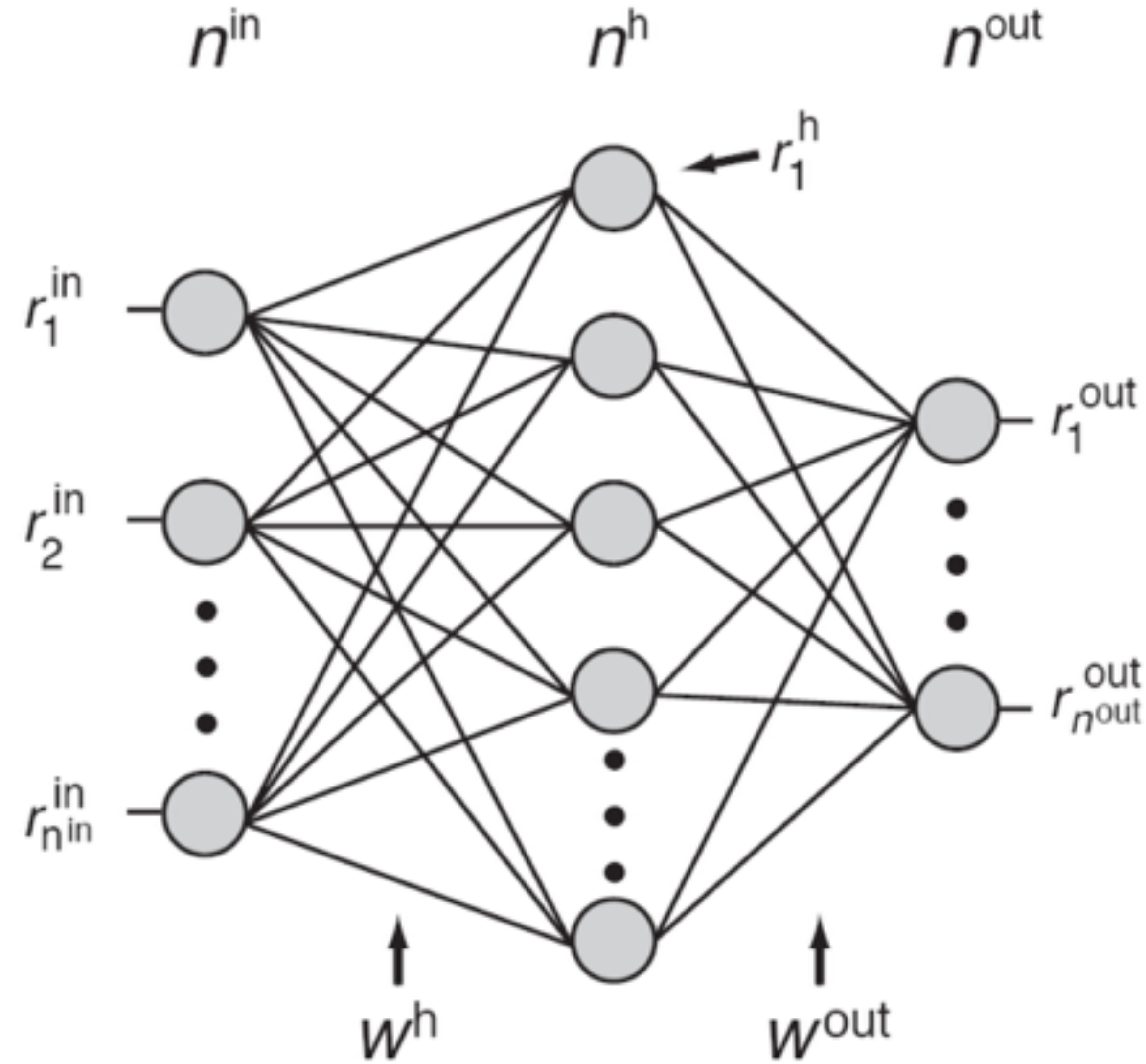
http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node20.html

The XOR problem



becomes



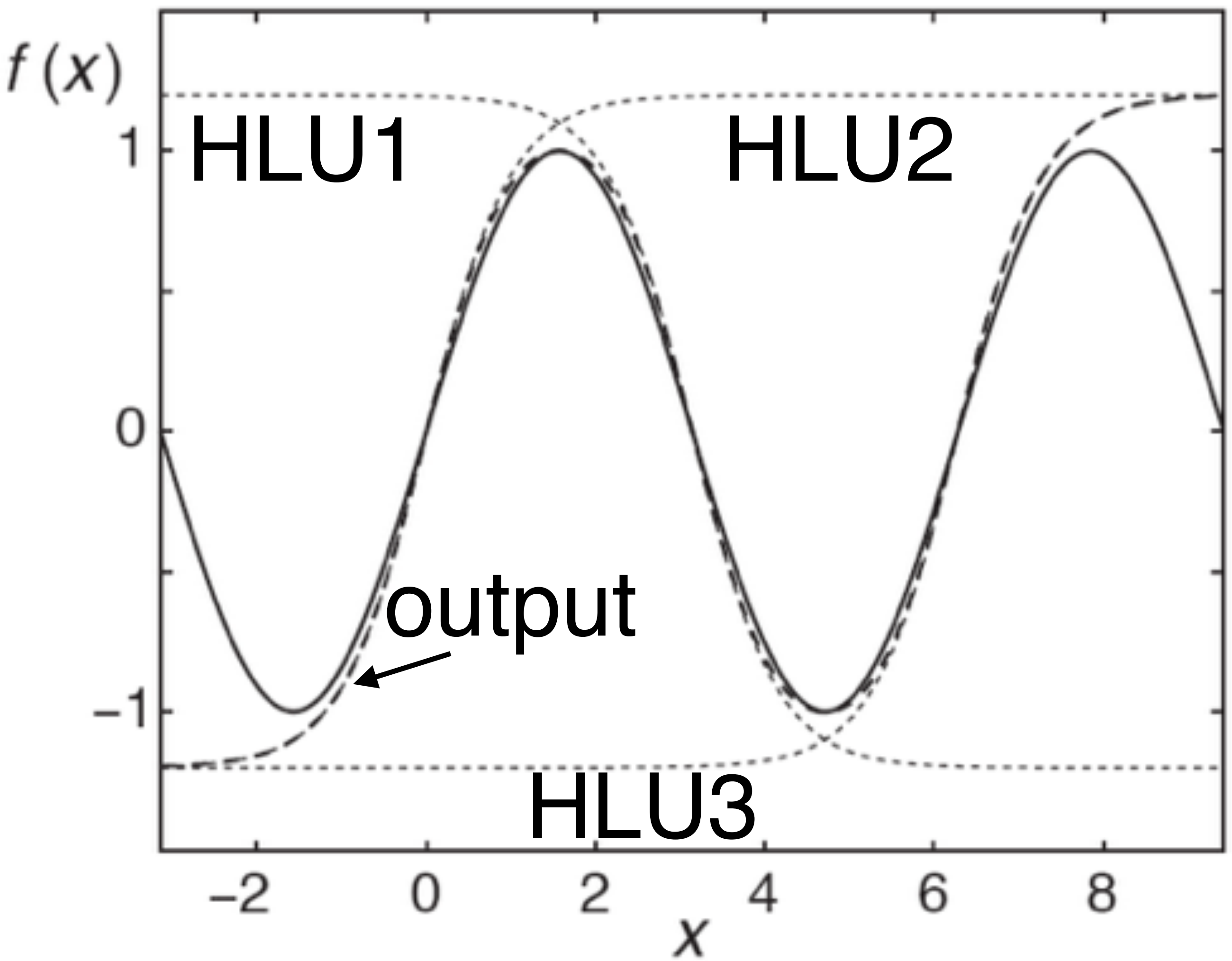


Trappenberg 2010

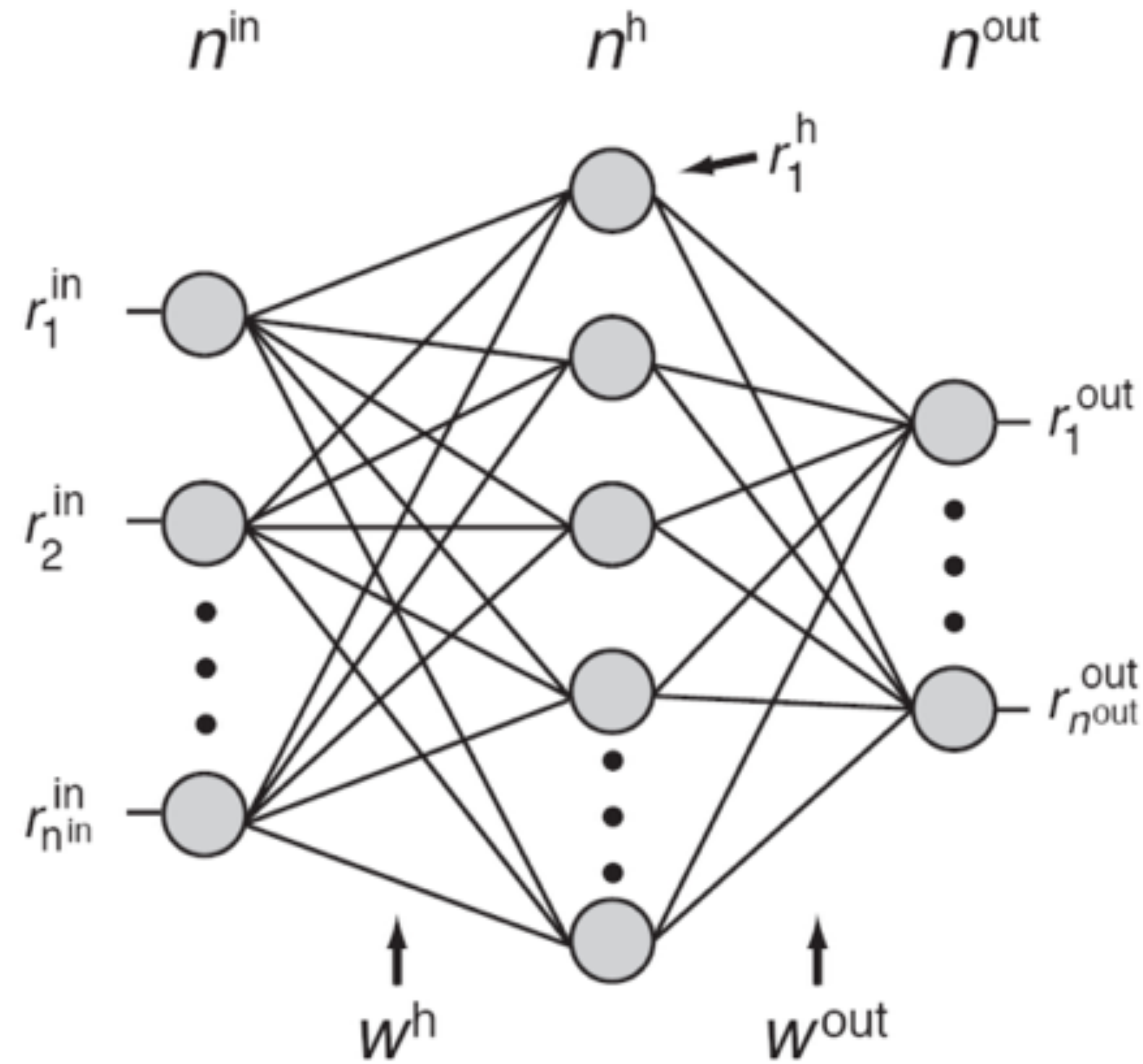
With enough hidden units, multilayer perceptron is the universal function approximator!

http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node20.html

Multi-layer perceptron



E.g. sine wave approximation using logistic transfer function



Trappenberg 2010

With enough hidden units, multilayer perceptron is the universal function approximator!

http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node20.html

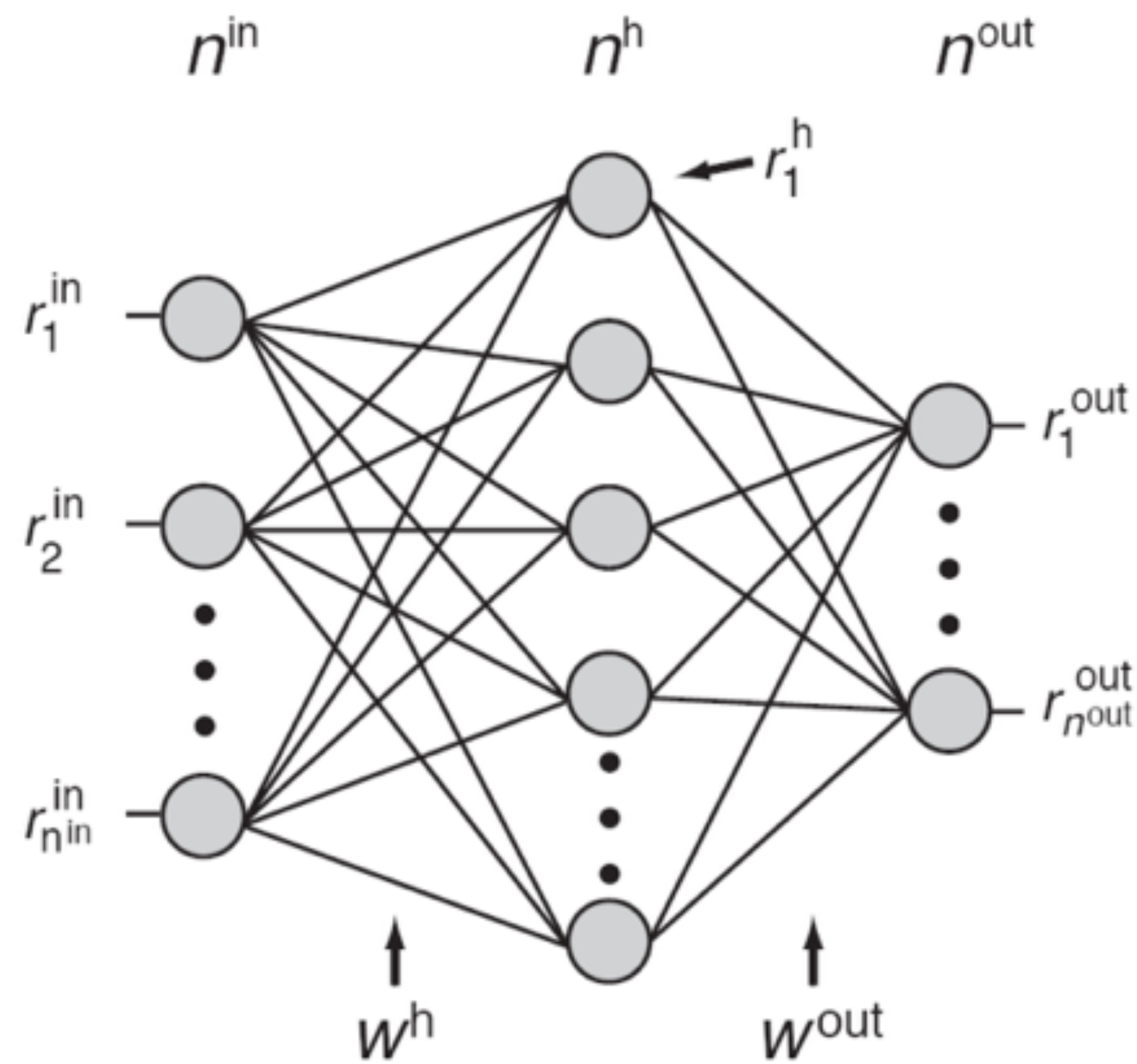
Multi-layer perceptron

Limitations

- Brain-like performance doesn't equate with actual performance
- Training rules are non-physiologic

Strengths

- Hidden layer activity might resemble brain function (with appropriate inputs/outputs)
- Brain = mapping network
- Self-Organization, like the brain
- High flexibility in possible computations

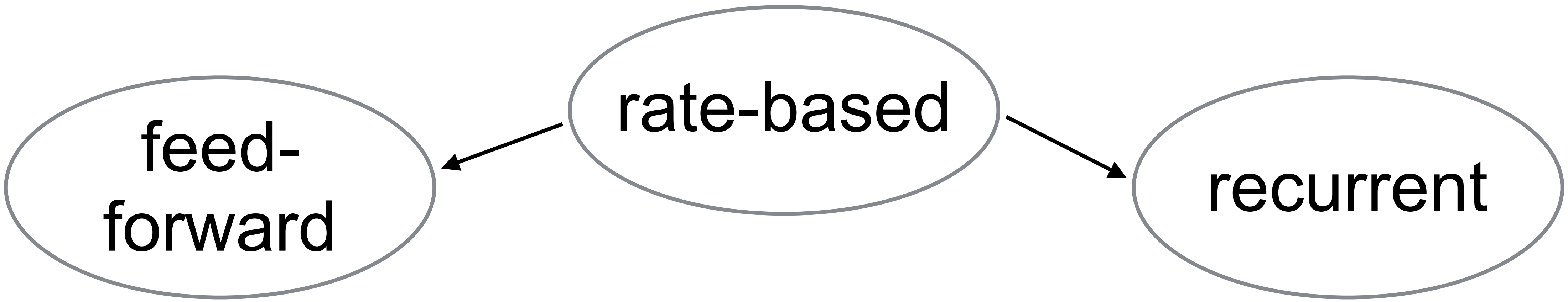


Trappenberg 2010

Point: MLP is usually good for machine learning purposes, but is not necessarily good for neuroscience theory all the time!

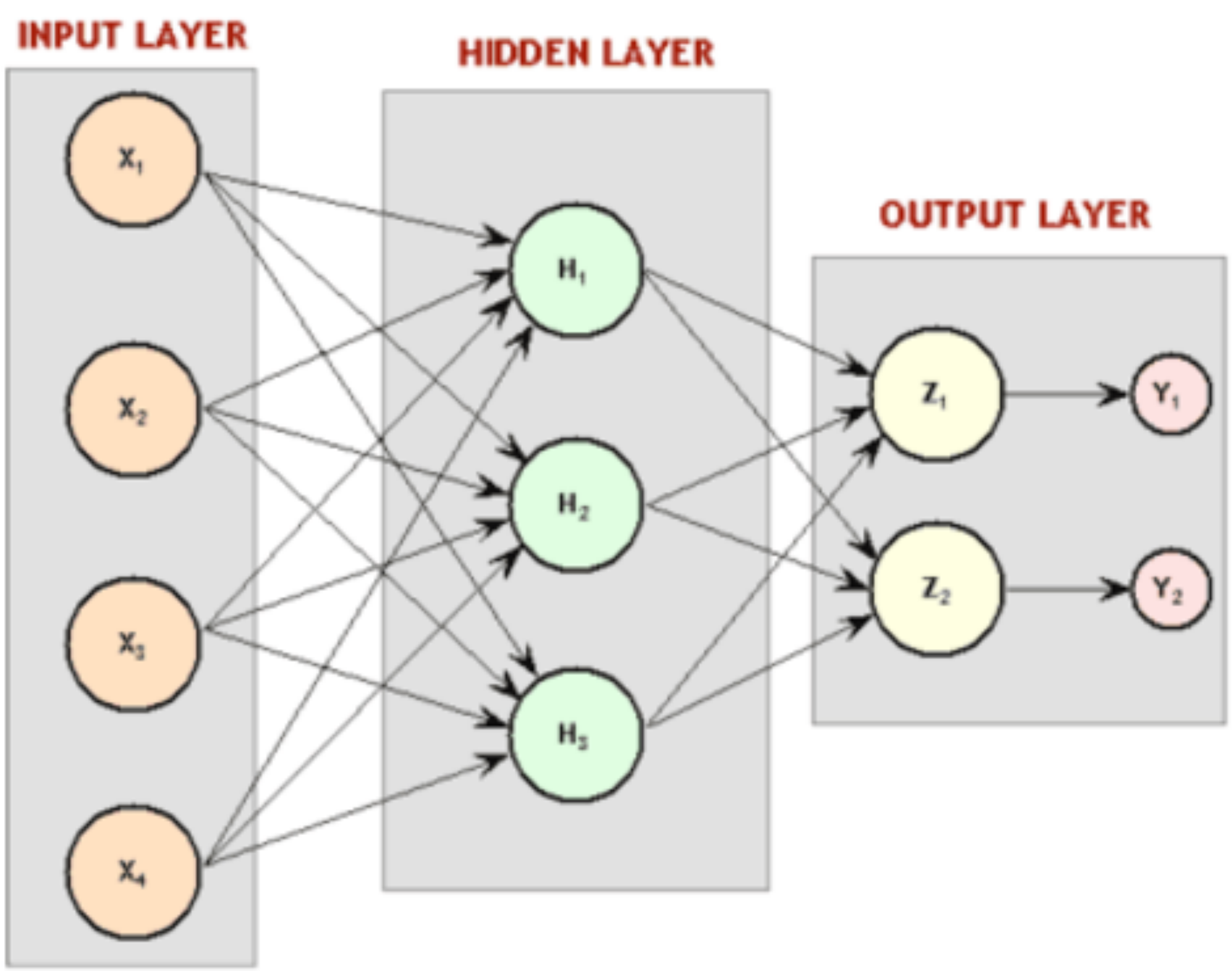
http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node20.html

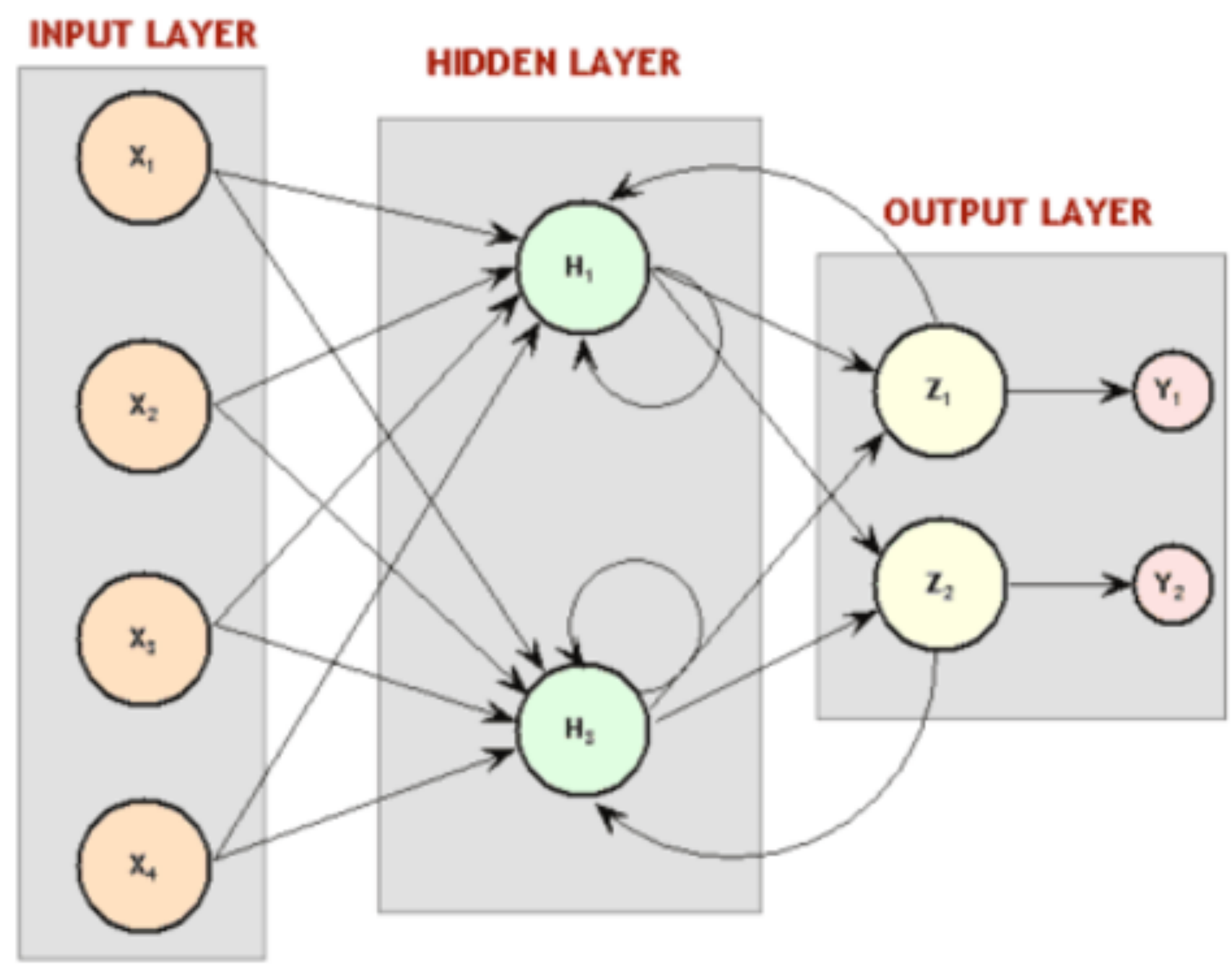
Two types of NNets



Feed-forward: information *only* flows forward, simplest connectivity

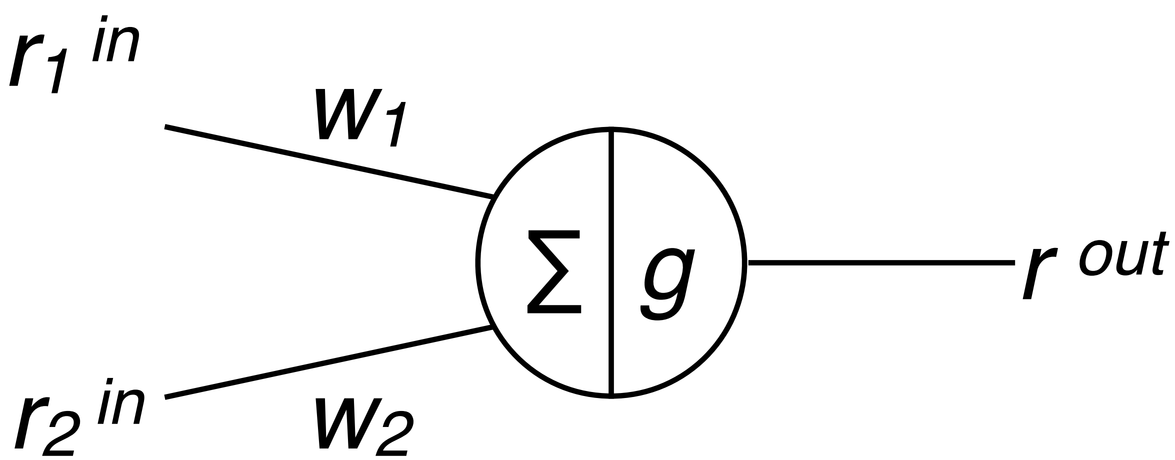
Recurrent: information can flow forward and backward, useful for time-resolved problems





How do we train a neural network?

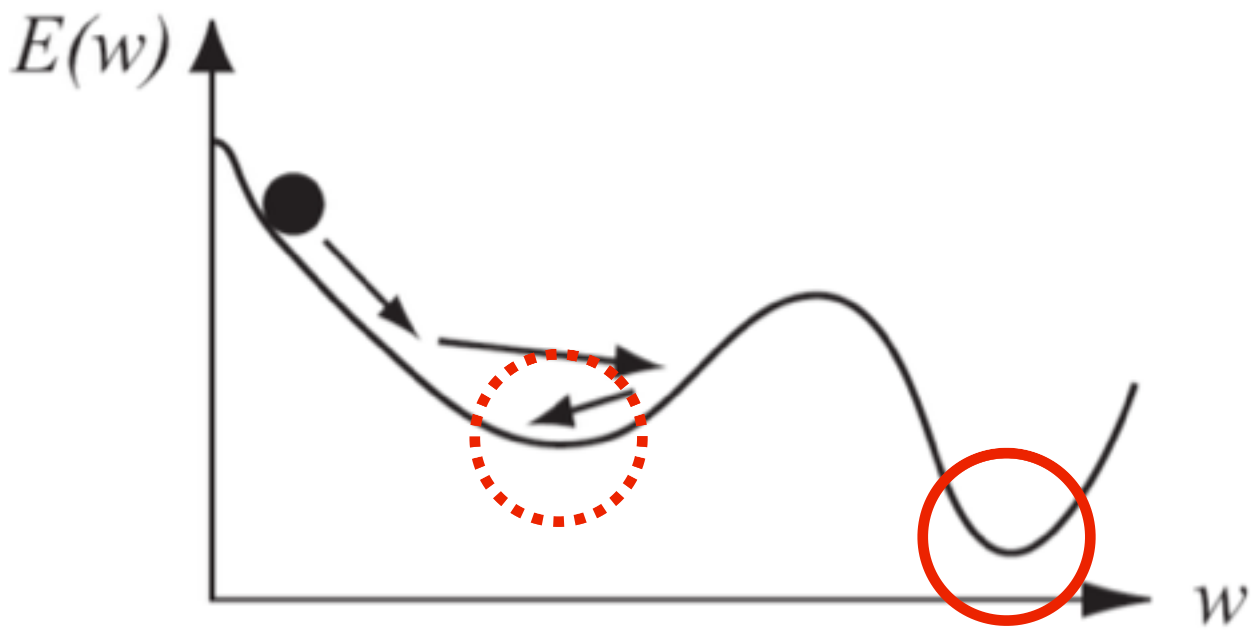
Minimize some cost function… gradient descent!



MSE

$$E = \frac{1}{2}\sum_i \left(r_i^{out} - y_i\right)^2$$

desired output (data, training set)



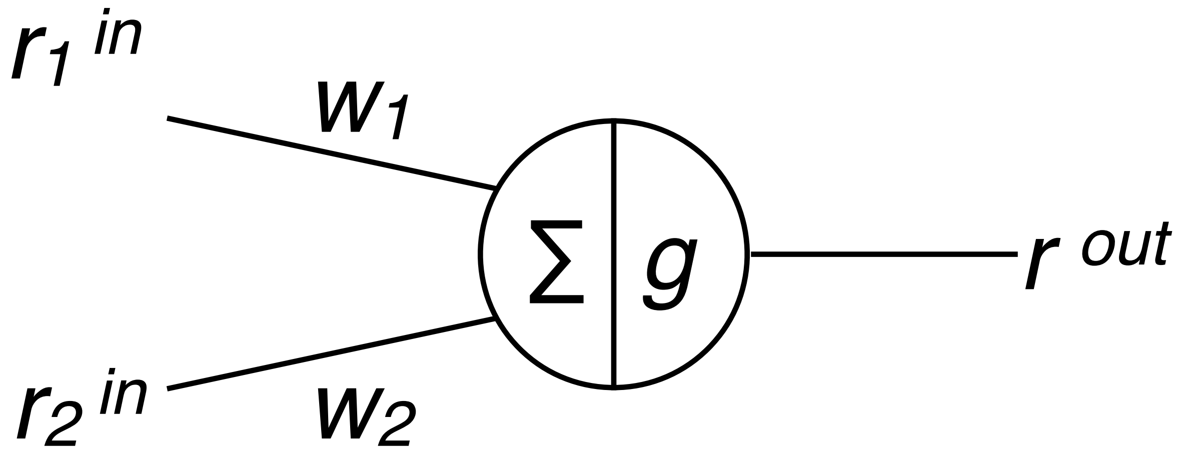
$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = -\varepsilon \left(\frac{\partial E}{\partial w_{ij}}\right)$$

learning rate

gradient of MSE wrt weights

How do we train a neural network?



$$E = \frac{1}{2}\sum_i \left(r_i^{out} - y_i\right)^2$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = -\varepsilon \left(\frac{\partial E}{\partial w_{ij}}\right)$$

change in weight i,j depends on learning rate and dependence of error on weight change at i,j

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{2}\frac{\partial}{\partial w_{ij}} \underbrace{\sum_i \left(g\left(\underbrace{\sum_j w_{ij} r_j^{in}}_{h_i}\right) - y_i\right)^2}_{f}$$

error's dependence on weight i,j, rewritten using MSE equation

$$\frac{\partial f}{\partial w_{ij}} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial h}\frac{\partial h}{\partial w_{ij}}$$

(using chain rule)

$$\Delta w_{ij} = \varepsilon \left(g'(h_i)\left(y_i - r_i^{out}\right) r_j^{in}\right)$$ **learning rule** (derivation?)

Adding layers…

Start by finding the output rates…

2-layer (1 hidden) perceptron

$$\mathbf{r}^{out} = g\left(\mathbf{w}^{out}\mathbf{r}^{h}\right)$$

$$r_i^{out} = g\left(\sum_j w_{ij}^{out} r_j^{h}\right)$$

$$E = \frac{1}{2}\sum_i \left(r_i^{out} - y_i\right)^2$$

$$\Delta w_{ij} = \varepsilon\left(g'\left(h_i\right)\left(y_i - r_i^{out}\right) r_j^{in}\right)$$

$$\mathbf{r}^{out} = g^{out}\left(\mathbf{w}^{out} g^{h}\left(\mathbf{w}^{h}\mathbf{r}^{in}\right)\right)$$

3-layer (2 hidden) perceptron

$$\mathbf{r}^{out} = g^{out}\left(\mathbf{w}^{out} g^{h_{out-1}}\left(\mathbf{w}^{h_{out-1}} g^{h_{out-2}}\left(\mathbf{w}^{h_{out-2}}\mathbf{r}^{in}\right)\right)\right)$$

n-layer (n-1 hidden) perceptron

$$\mathbf{r}^{out} = g^{out}\left(\mathbf{w}^{out} g^{h_{out-1}}\left(\mathbf{w}^{h_{out-1}} g^{h_{out-2}}\left(\mathbf{w}^{h_{out-2}} \ldots g^{h_{out-n+1}}\left(\mathbf{w}^{h_{out-n+1}} g^{h_{out-n}}\left(\mathbf{w}^{h_{out-n}}\mathbf{r}^{in}\right)\right)\right)\right)\right)$$

Idea is to nest each layer's output rates within the next…

Training through the layers…

$$E = \frac{1}{2}\sum_i \left(r_i^{out} - y_i\right)^2$$

$$\Delta w_{ij} = \varepsilon \left(g'\left(h_i\right)\left(y_i - r_i^{out}\right) r_j^{in}\right)$$

Generalized delta rule (output wts)

$$\frac{\partial E}{\partial w_{ij}^{out}} = \frac{1}{2}\frac{\partial}{\partial w_{ij}^{out}}\sum_i \left(r_i^{out} - y_i\right)^2$$
$$= \delta_i^{out} r_j^h$$

with

$$\delta_i^{out} = g^{out}{}'\left(h_i^h\right)\left(r_i^{out} - y_i\right)$$ **delta rule for output weights**

Hidden layer weights

Error propagates BACKWARDS through the layers!

$$\frac{\partial E}{\partial w_{ij}^h} = \frac{1}{2}\frac{\partial}{\partial w_{ij}^h}\sum_i \left(r_i^{out} - y_i\right)^2$$

$$\frac{\partial E}{\partial w_{ij}^h} = \frac{1}{2}\frac{\partial}{\partial w_{ij}^h}\sum_i \left(g^{out}\left(\sum_j w_{ij}^{out} g^h\left(\sum_k w_{jk}^h r_k^{in}\right)\right) - y_i\right)^2$$
$$= \delta_i^h r_j^{in}$$

with

$$\delta_i^h = g^h{}'\left(h_i^{in}\right)\sum_k w_{ik}^{out}\boxed{\delta_k^{out}}$$

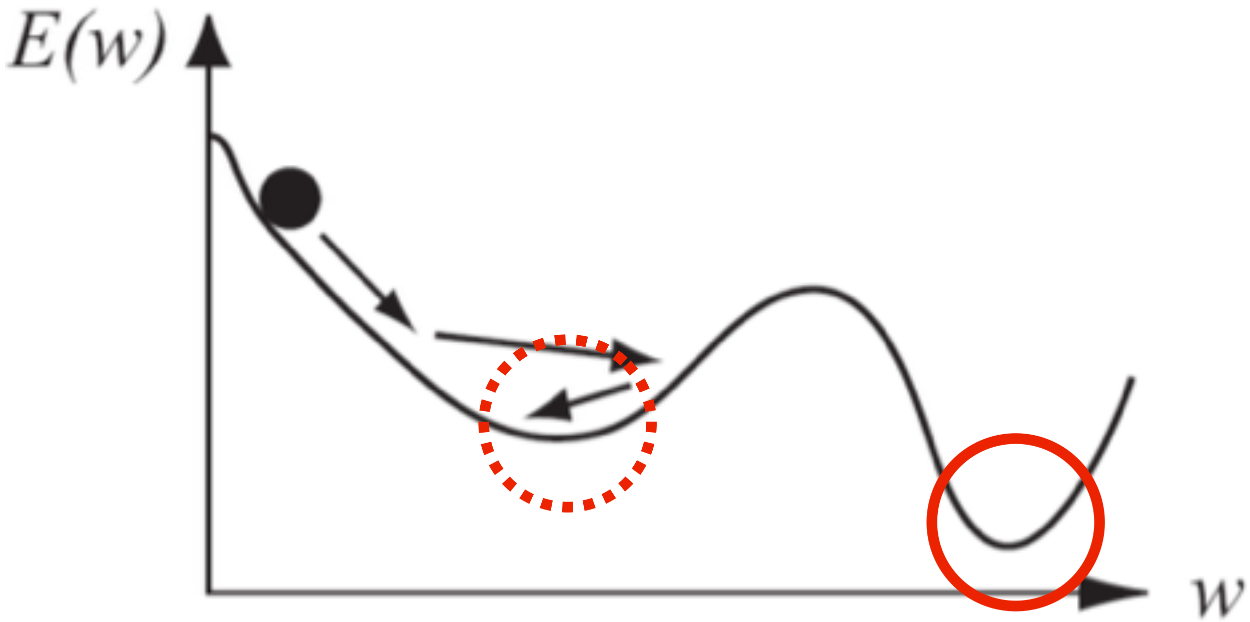
delta rule for hidden wts
(depends on delta rule for output wts)

Training protocol (gradient descent)

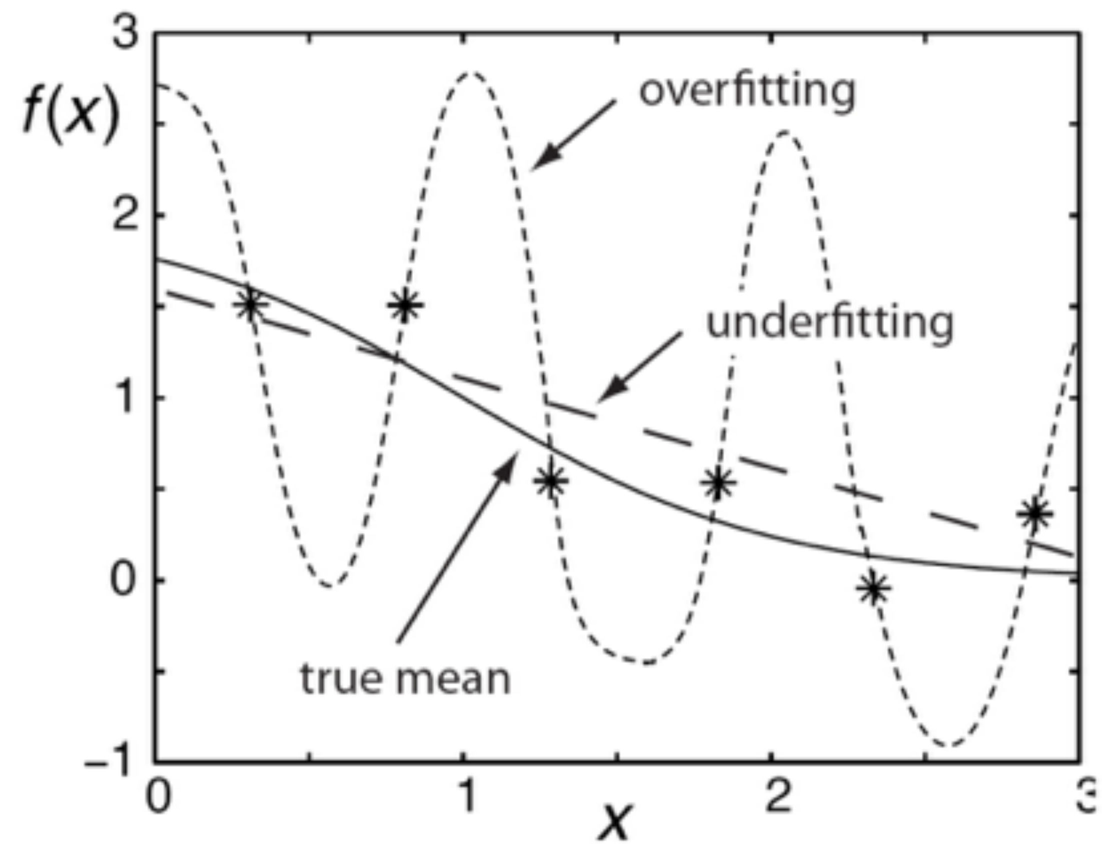
1. Train until gradient of error function reaches minimum.
   - **A. Batch:** use full training set with each iteration (smooth convergence, but more prone to local minima)
   - **B. Online:** use different sample of training set with each iteration (more memory efficient, but messy convergence)

2. Test generalization of network using previously unseen test data set.

Caveats



Local minima
Can use momentum term in weight update to incorporate history of weight changes.

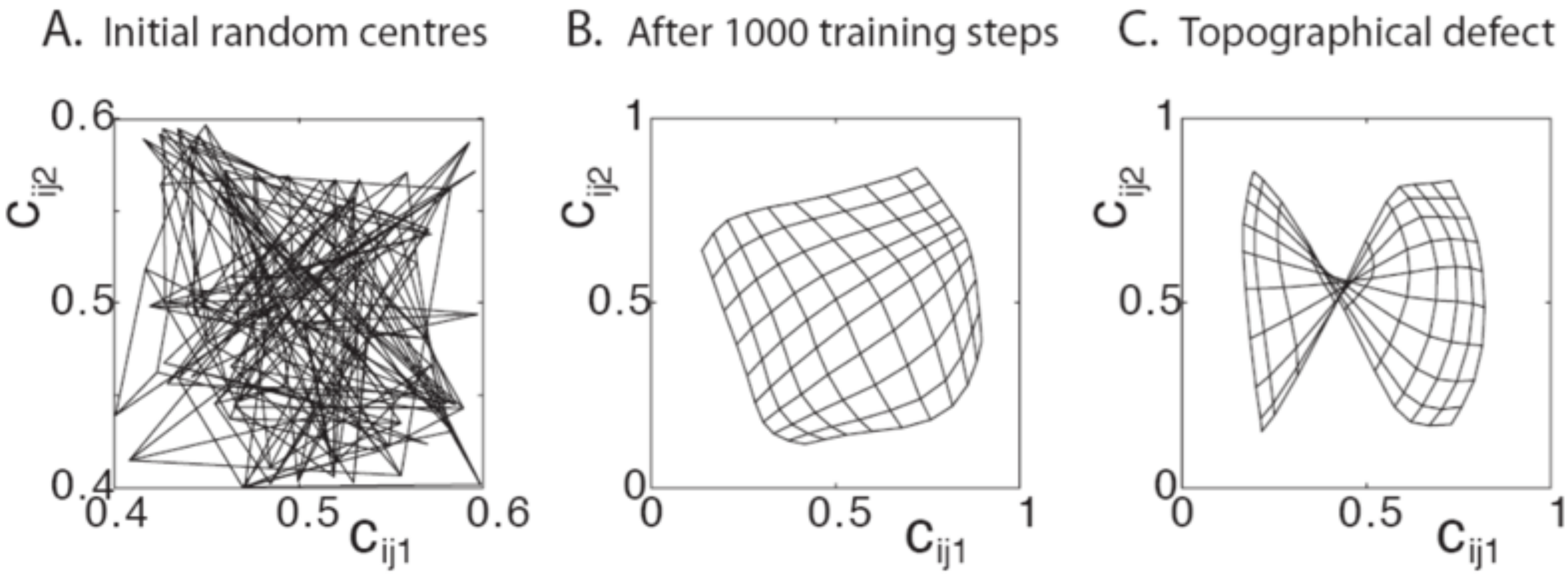


Trappenberg 2010

Overfitting
Use heuristics to determine appropriate number of nodes for solving particular problem. Can usually use 2*number of nodes for training set.

Recurrent networks are a whole new game!

but I'll spare you



Trappenberg 2010

Backprop. through time (BPTT)