

The Backpropagation Algorithm

Psych 209

Andrew Saxe

Backpropagation from 30,000ft

- Learning algorithm for arbitrary, deep, complicated neural networks
- You've used it!
 - Google/Microsoft/Yahoo voice recognition, image search, machine translation, ...
- The core idea behind many psychology & neuroscience models

Backpropagation from 30,000ft

- Can be written down in one line of math:

$$\Delta W = -\lambda \frac{\partial E}{\partial W}$$

- Sort of like Newton's laws
 - Three simple equations
 - But endless implications and consequences
- Need to understand it *intuitively* at *many levels*

Two meanings

- “Backprop” technically refers to a specific *algorithm*
- But often used as shorthand for a much broader *framework* with a galaxy of associated ideas and commitments
- Important not to confuse *backprop-the-algorithm* with *backprop-the-framework*—this has caused trouble in the past
- Your job is to learn backprop-the-framework

Levels of analysis

- Computational level
 - Learning as optimization
 - Gradient descent
- Algorithmic level
 - Backprop-the-algorithm
- Implementation level

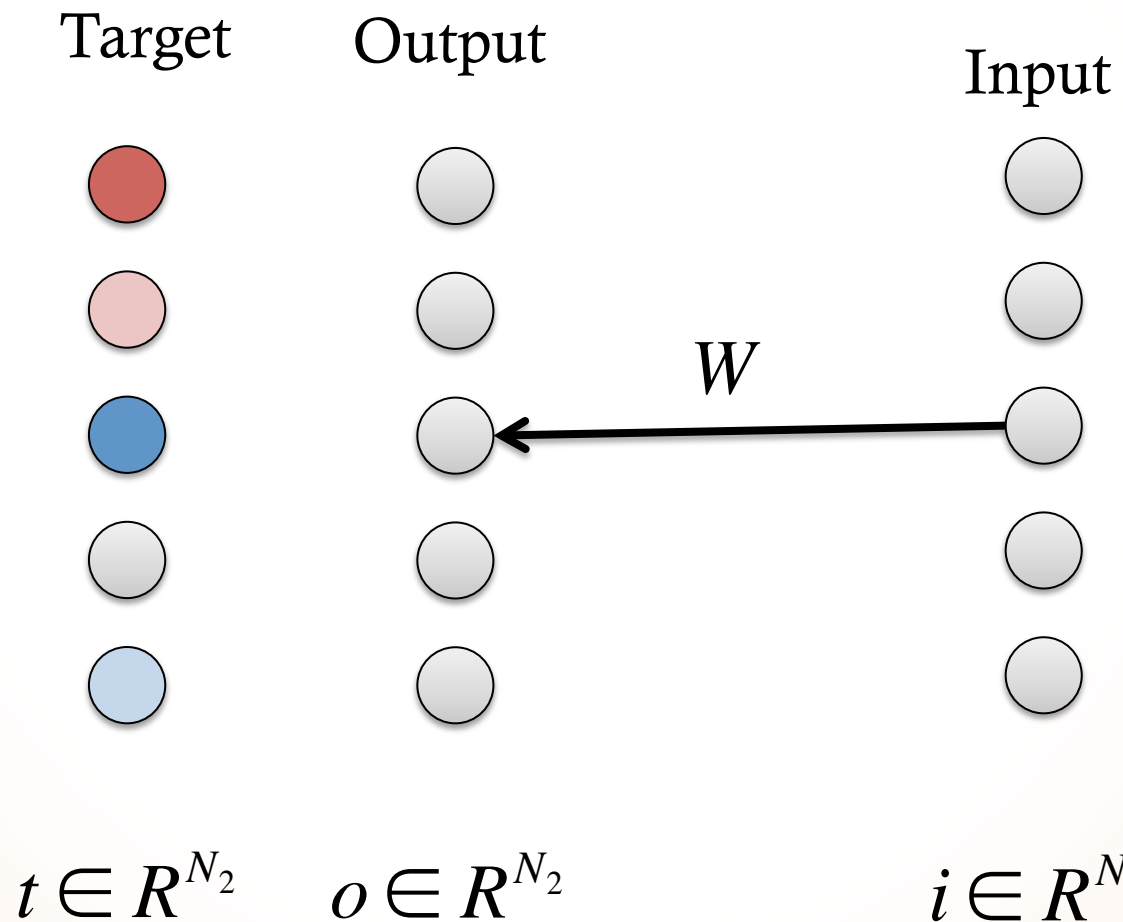
Psych/neuro conceptual implications

- Backprop-as-framework comes with a host of ideas about the brain and mind:
- E.g., learning and development are
 - Incremental
 - Experience-driven
 - Encoded in synaptic weights
- Computation is
 - Parallel
 - Distributed

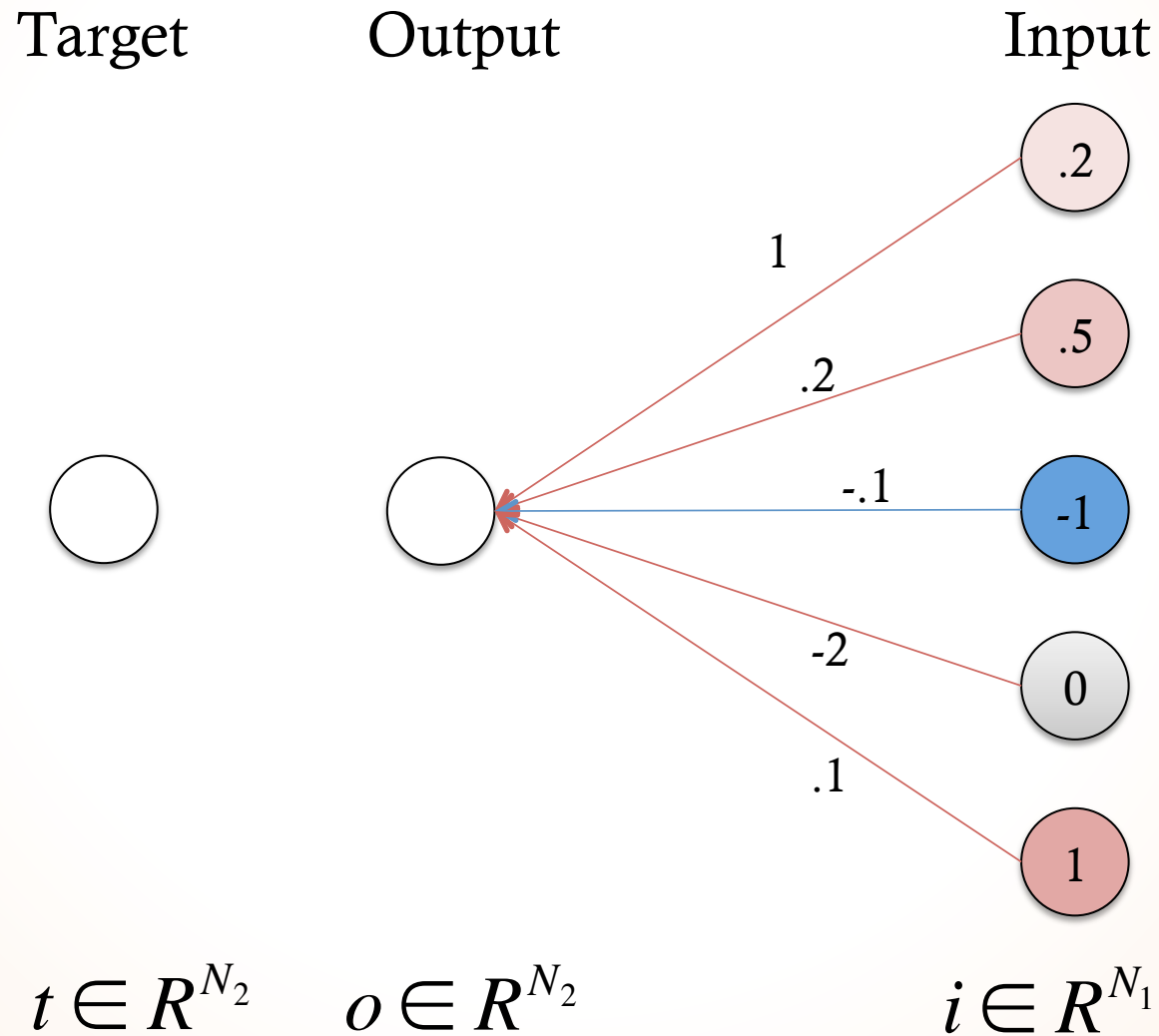
Today

- Build up to learning in arbitrary deep network
 1. One parallel layer
 2. Many serial layers
 3. Nonlinearities
 4. Pandemonium! The general case

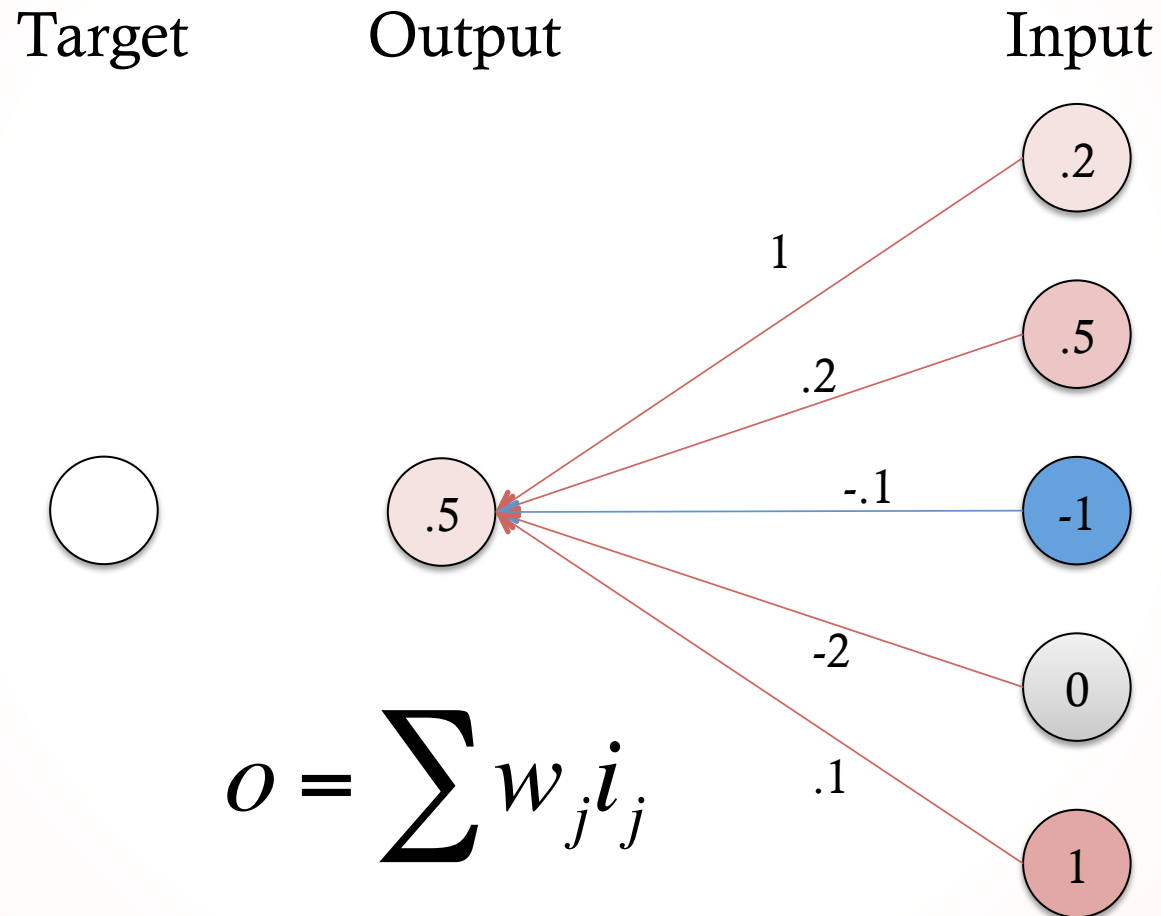
The pattern associator



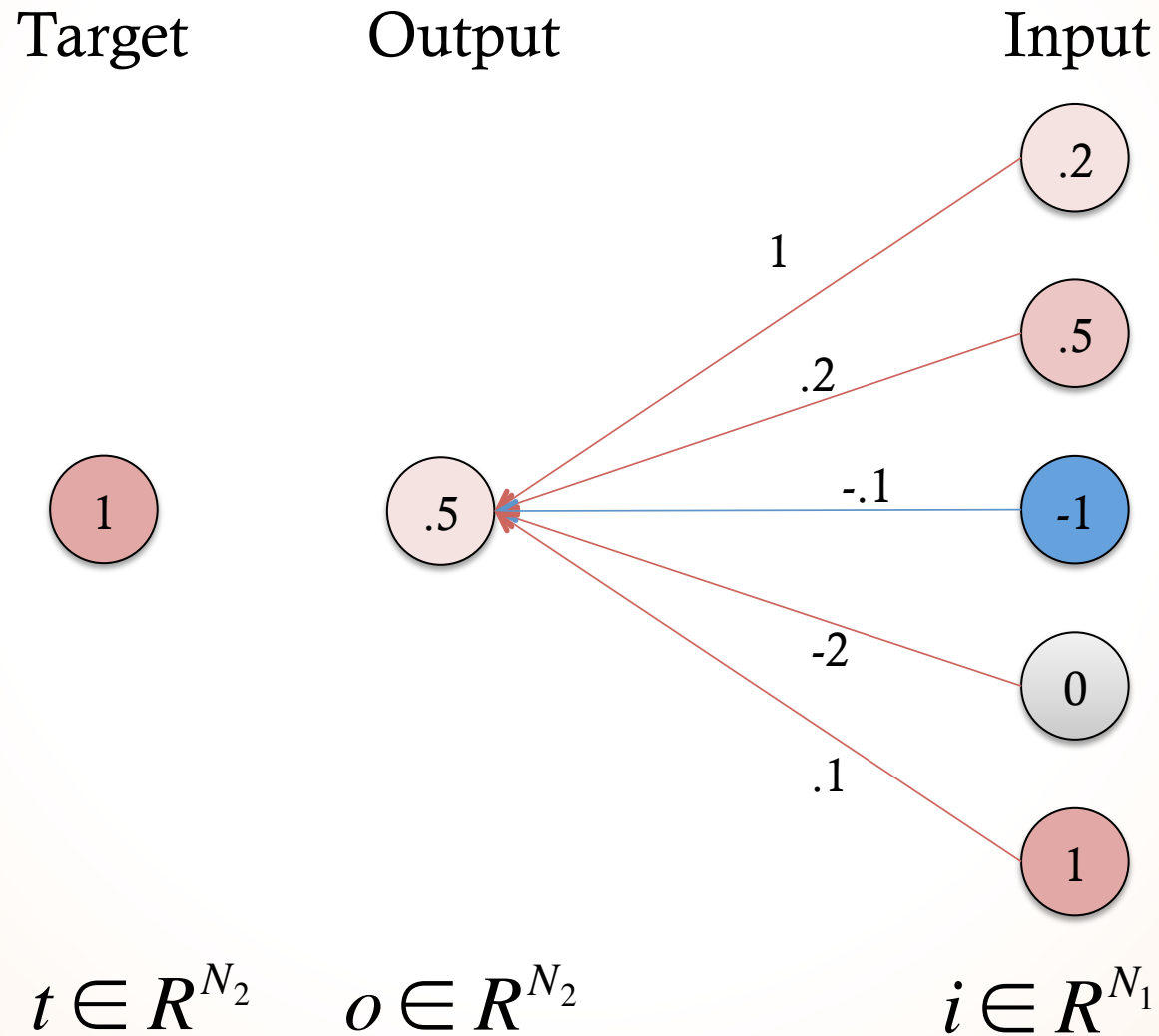
One layer learning



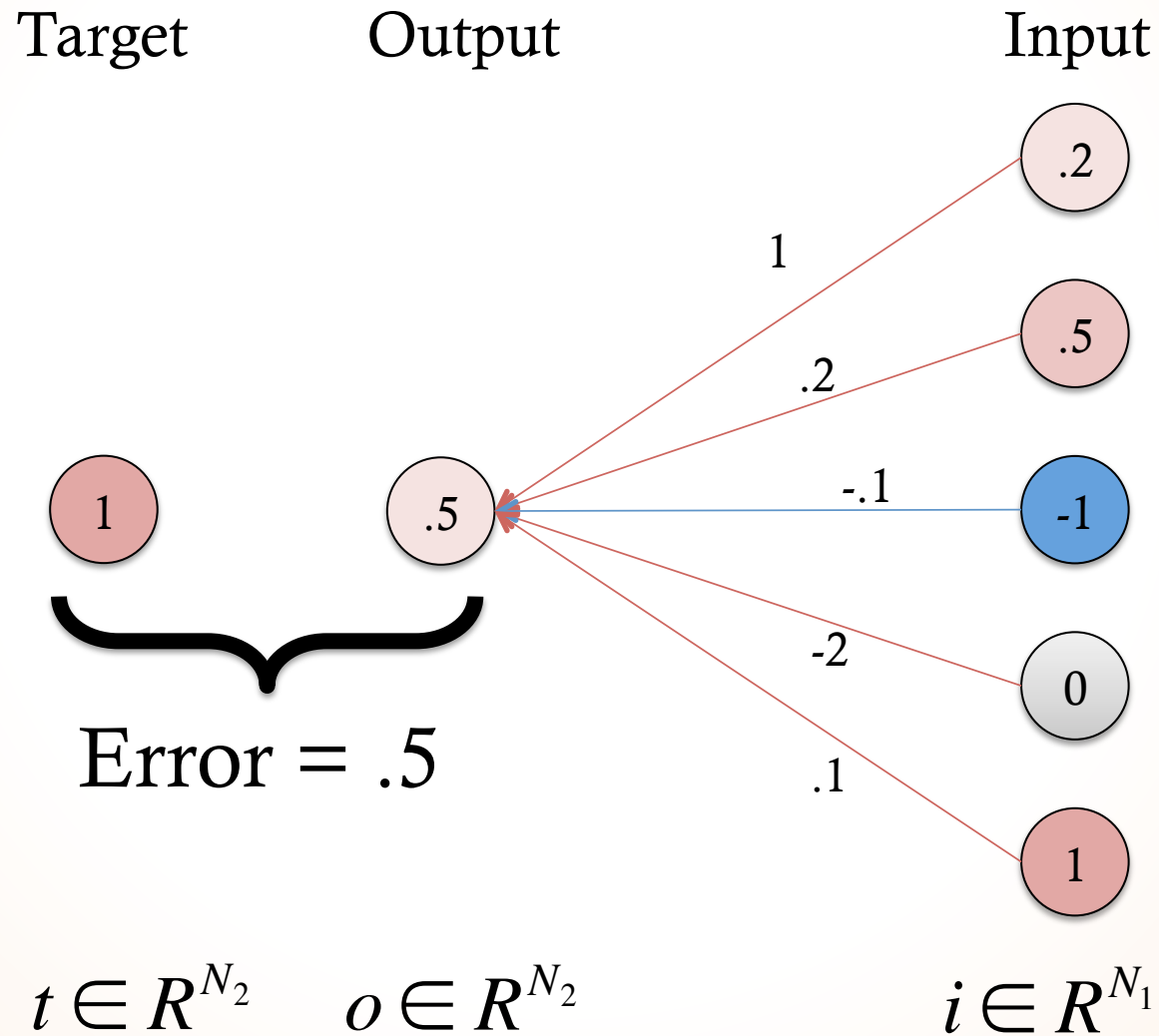
Forward propagation



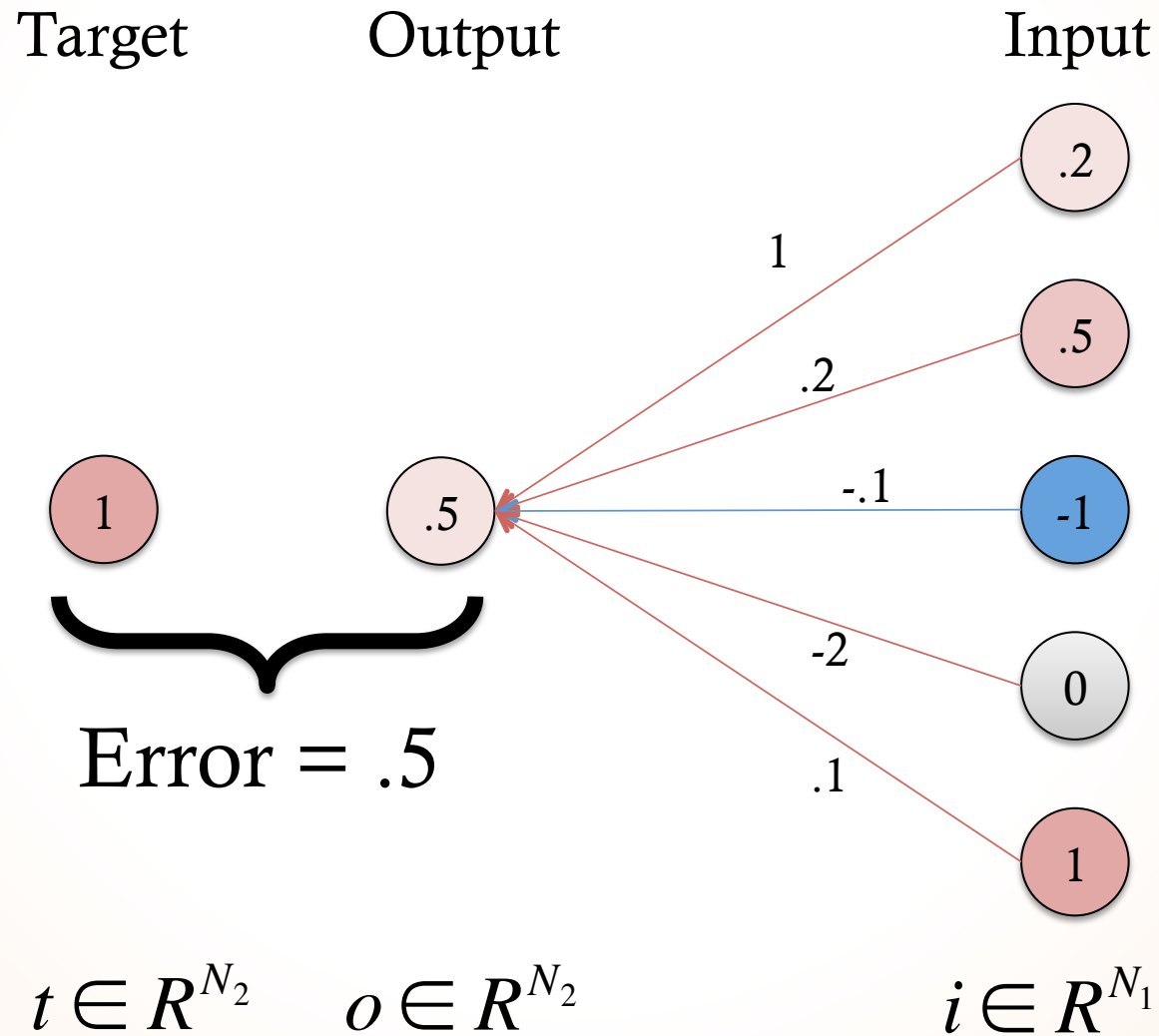
Target Feedback



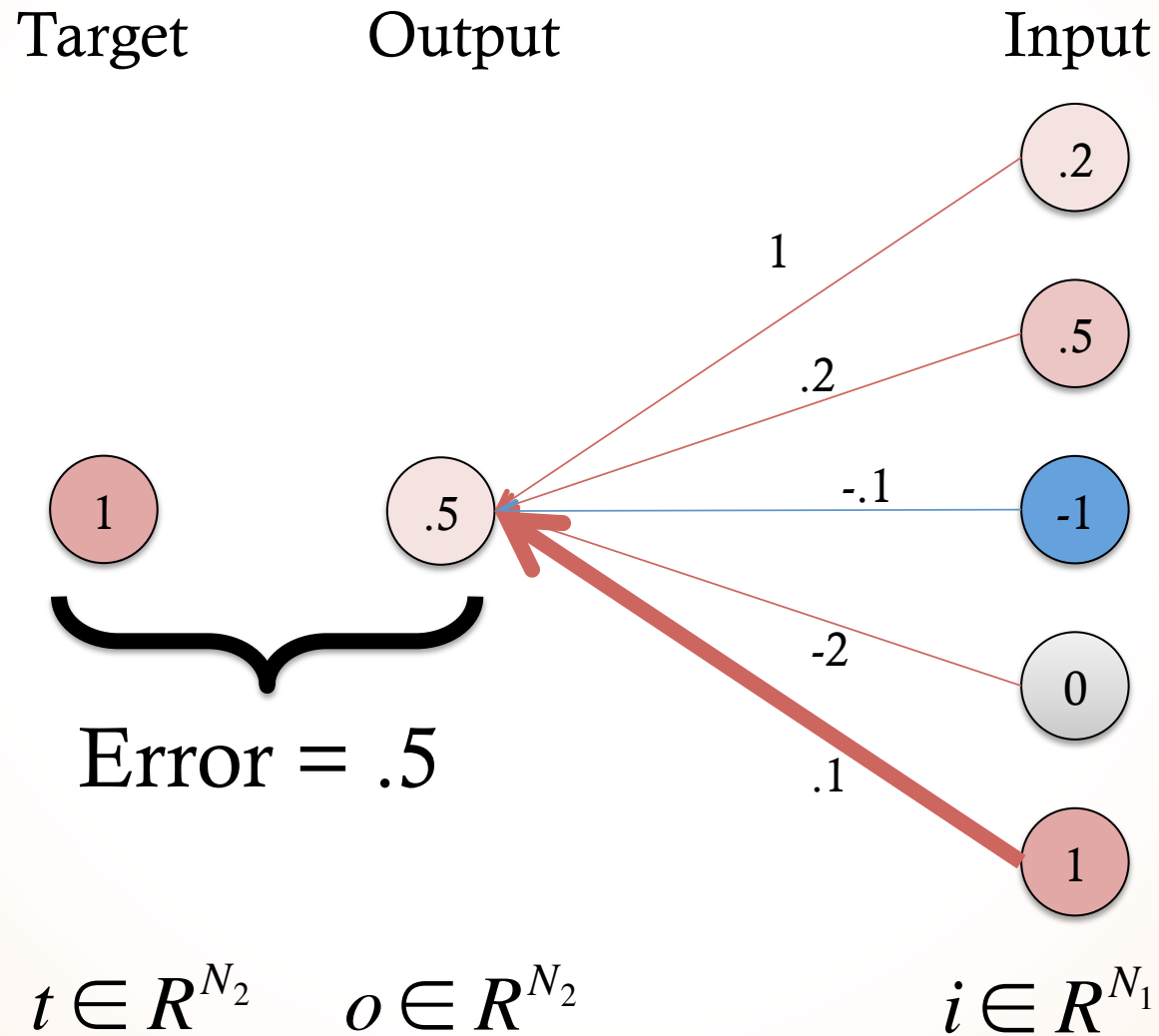
Error signal



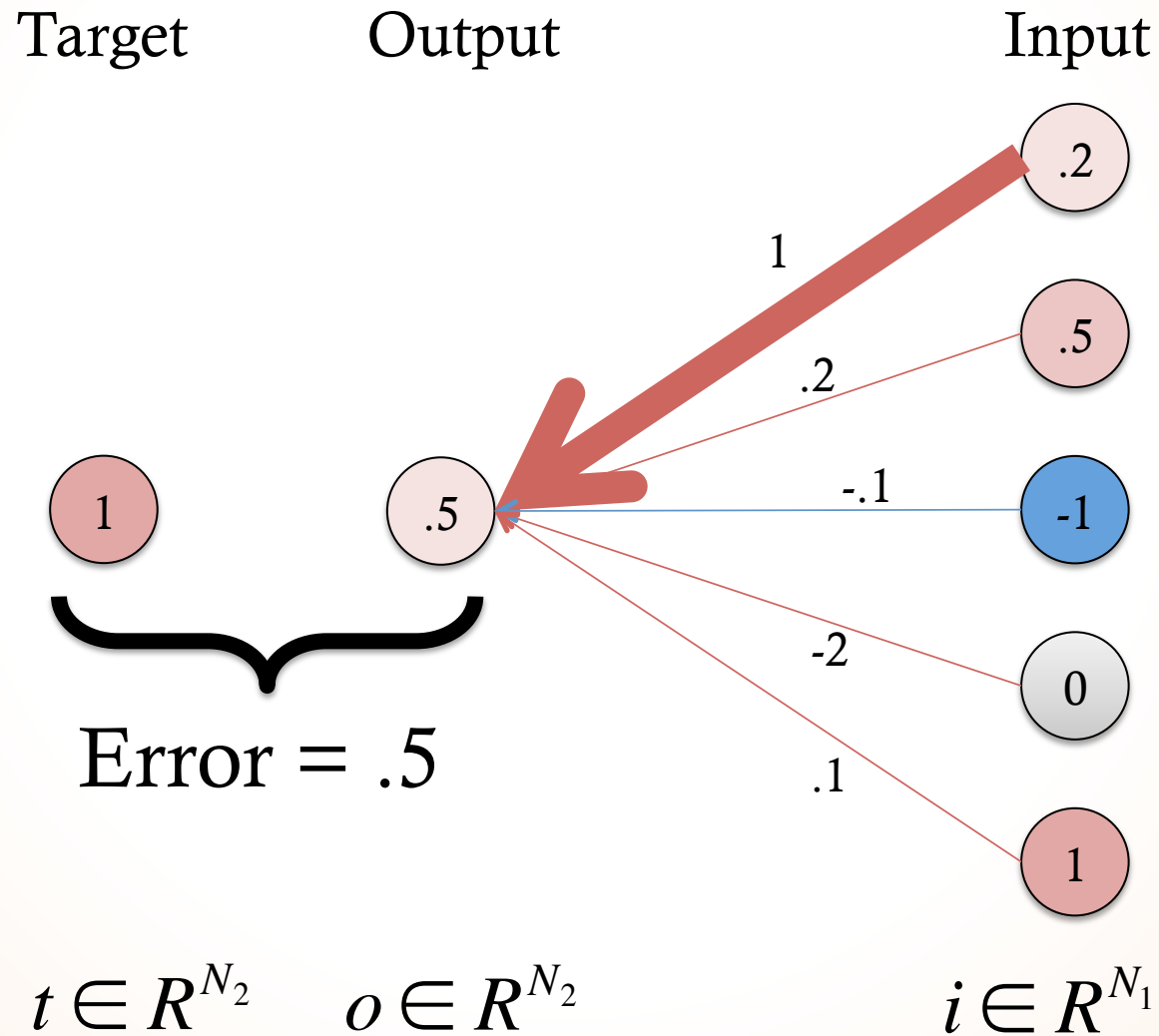
How to fix?



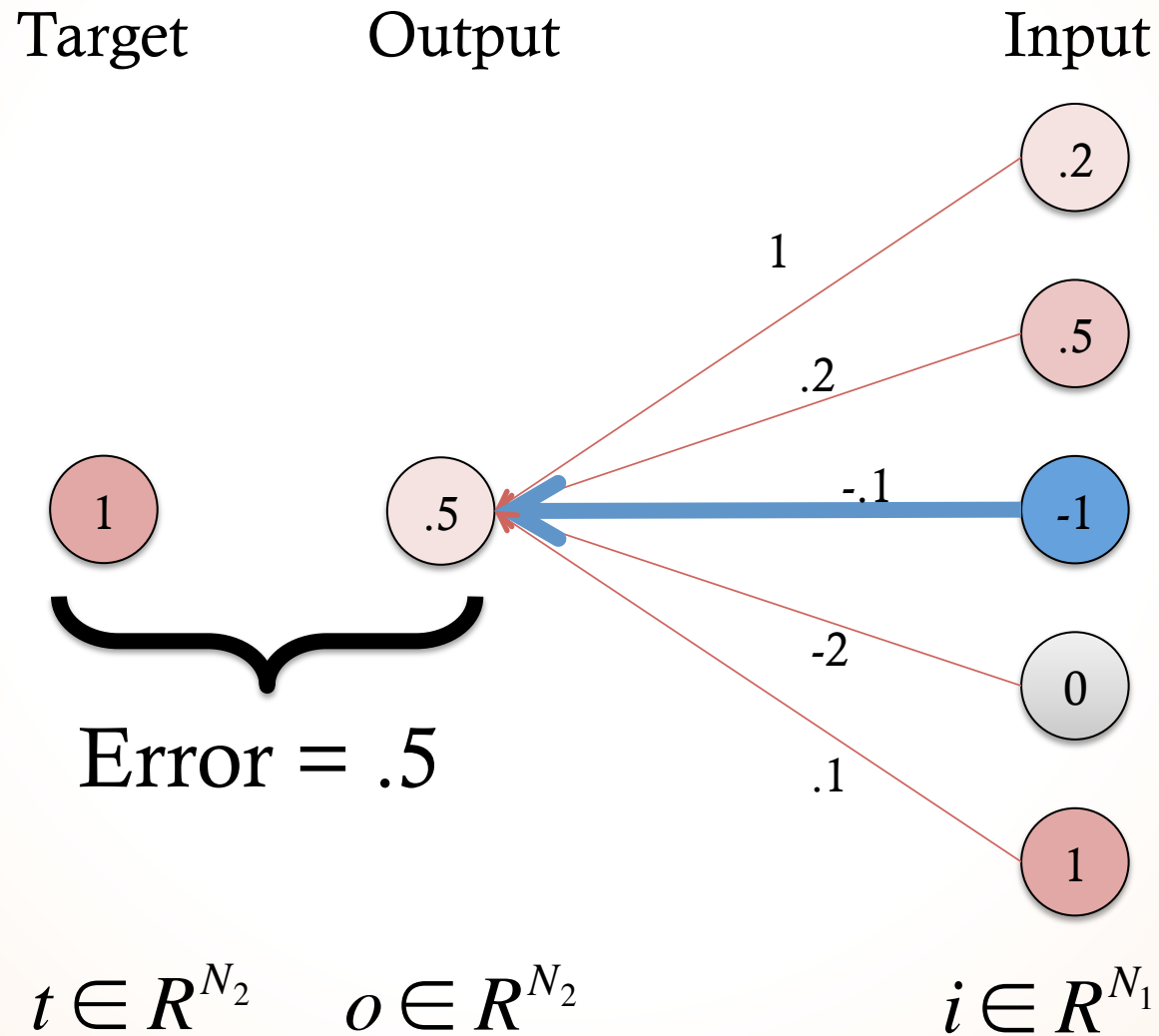
How to fix?



How to fix?



How to fix?



Lots of choices

- How can we choose?
- Maybe: make small change that *most rapidly* decreases the error

Lots of choices

- How can we choose?
- Maybe: make small change that *most rapidly* decreases the error
- This is called Gradient Descent

Gradient Descent Learning

- Make small change in weights that most rapidly improves task performance



- Change each weight in proportion to the gradient of the error $\Delta W = -\lambda \frac{\partial E}{\partial W}$

Optimization view of learning

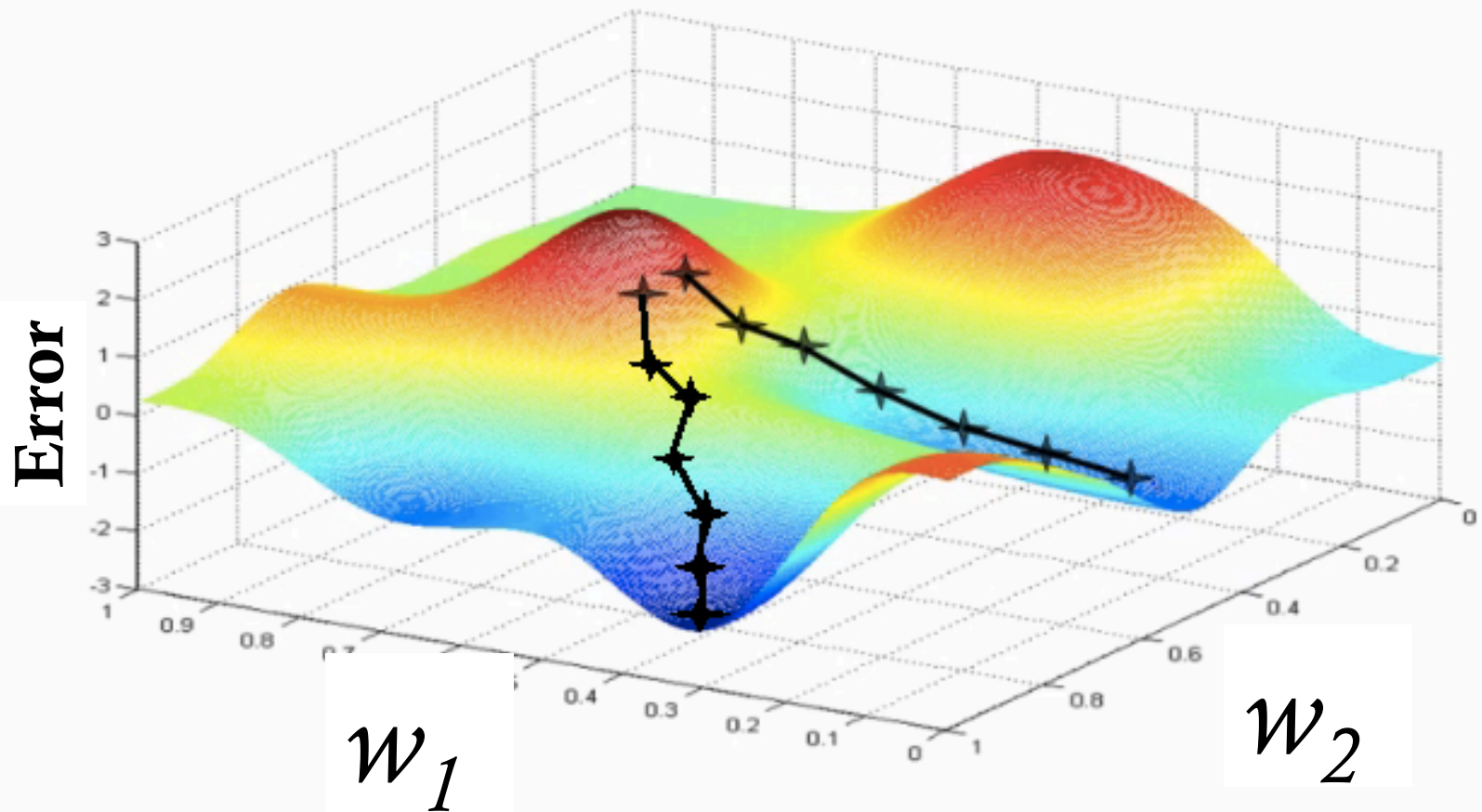
- The network and training data together define an *error function*, say, squared error

$$E(t, w, i) = (t - o)^2$$

- *Learning* is just minimizing this function w.r.t. w

$$w^* = \operatorname{argmin}_w E(t, w, i)$$

Gradient Descent $\Delta W = -\lambda \frac{\partial E}{\partial W}$



<http://blog.datumbox.com/wp-content/uploads/2013/10/gradient-descent.png>

Delta rule derivation

$$\begin{aligned} E &= (t - o)^2 \\ &= (t - \sum w_j i_j)^2 \\ &= (t - wi)^2 \end{aligned}$$

Delta rule derivation

$$\begin{aligned}E &= (t - o)^2 \\&= (t - \sum w_j i_j)^2 \\&= (t - wi)^2\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_k} &= -2(t - o) \frac{\partial E}{\partial w_k} o \\&= -2(t - o) \sum \frac{\partial E}{\partial w_k} w_j i_j \\&= -2(t - o) i_k\end{aligned}$$

Delta rule derivation

$$\begin{aligned}E &= (t - o)^2 \\&= (t - \sum w_j i_j)^2 \\&= (t - wi)^2\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial w_k} &= -2(t - o) \frac{\partial E}{\partial w_k} o \\&= -2(t - o) \sum \frac{\partial E}{\partial w_k} w_j i_j \\&= -2(t - o) i_k\end{aligned}$$

Delta rule: $\Delta w = \lambda(t - o)i^T = \lambda ei^T$

What about *deep* networks?

- Delta rule only makes sense for one layer!
- How can we learn in deep, multilayer networks?

What about *deep* networks?

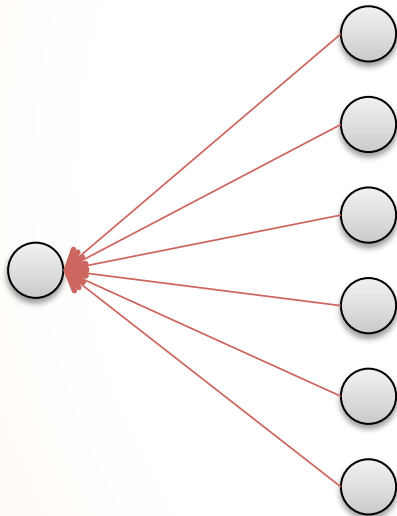
- Delta rule only makes sense for one layer!
- How can we learn in deep, multilayer networks?
- Why would we want deep, multilayer networks?

Why deep networks?

- Shallow networks can't represent any function, eg, XOR
- Anatomically and physiologically, the brain is deep
- Empirically, deep networks just *work better*

Two fundamental topologies

Parallel

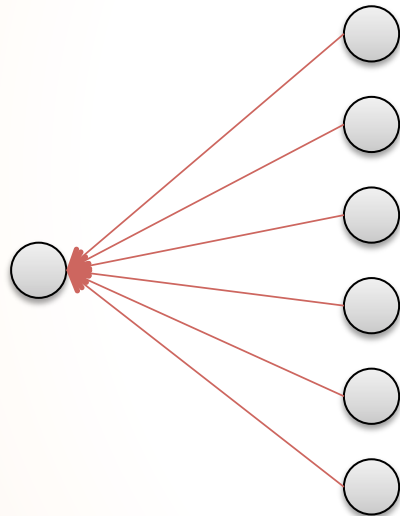


Serial

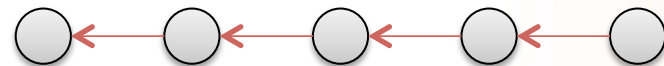


Two fundamental topologies

Parallel



Serial

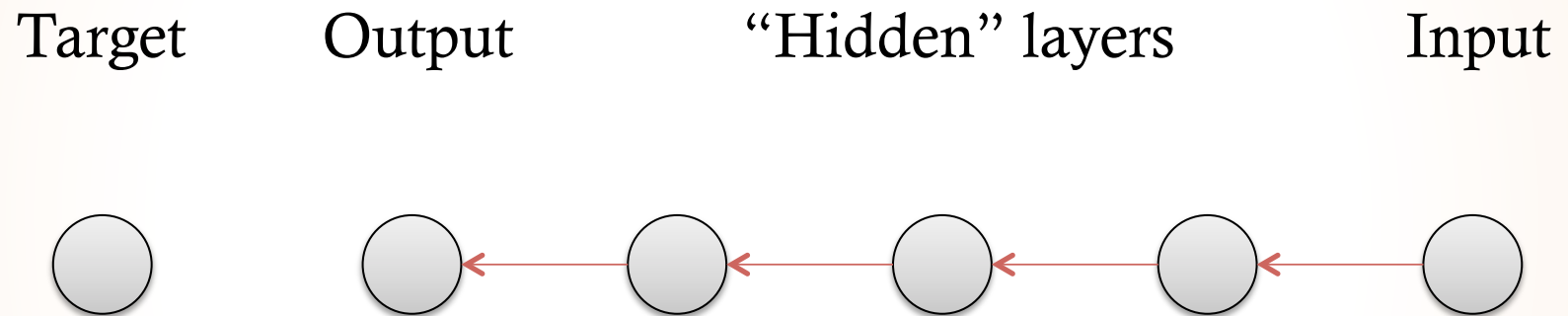


Gradient descent learning rule:

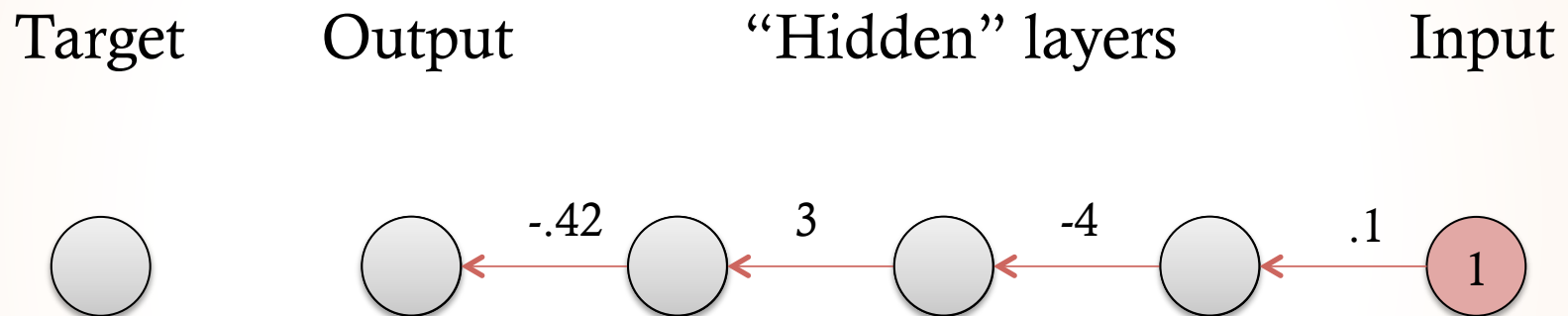
Delta rule

???

Many layer learning

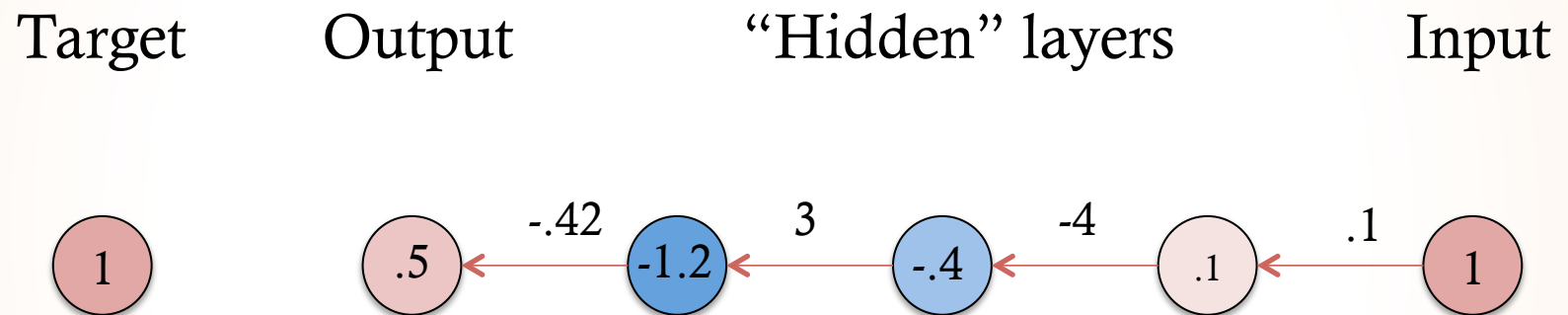


Forward propagation

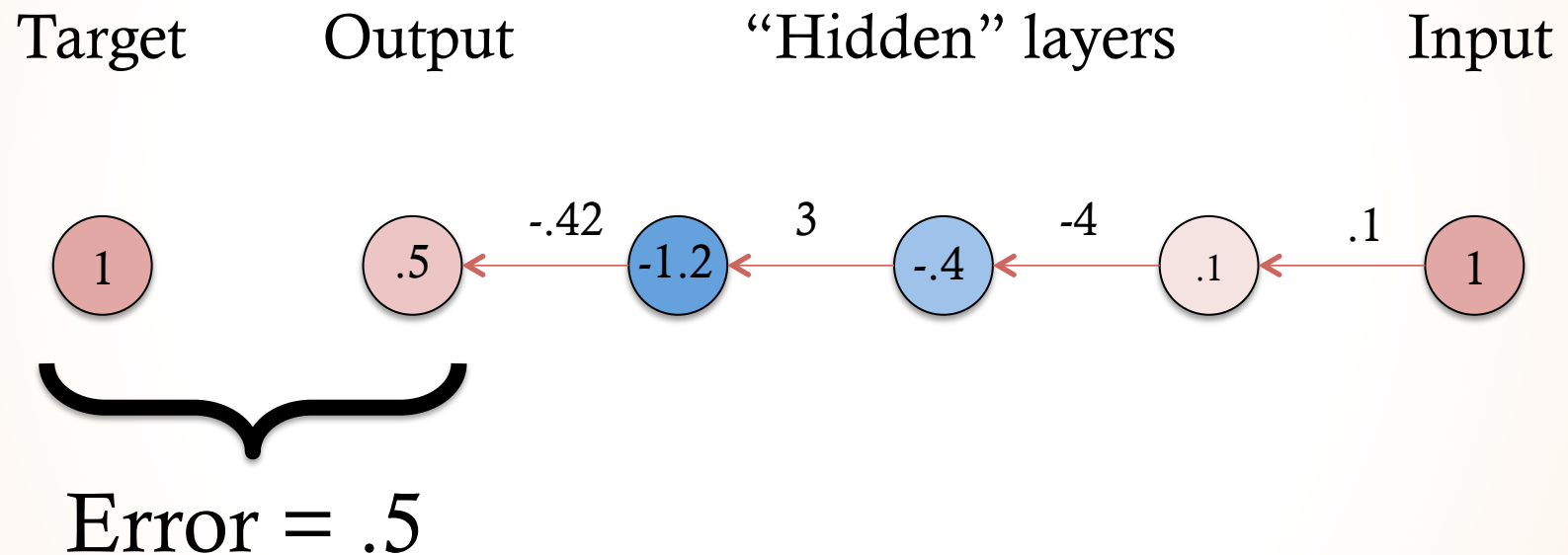


$$o = w_4 w_3 w_2 w_1 i = \left(\prod w_k \right) i$$

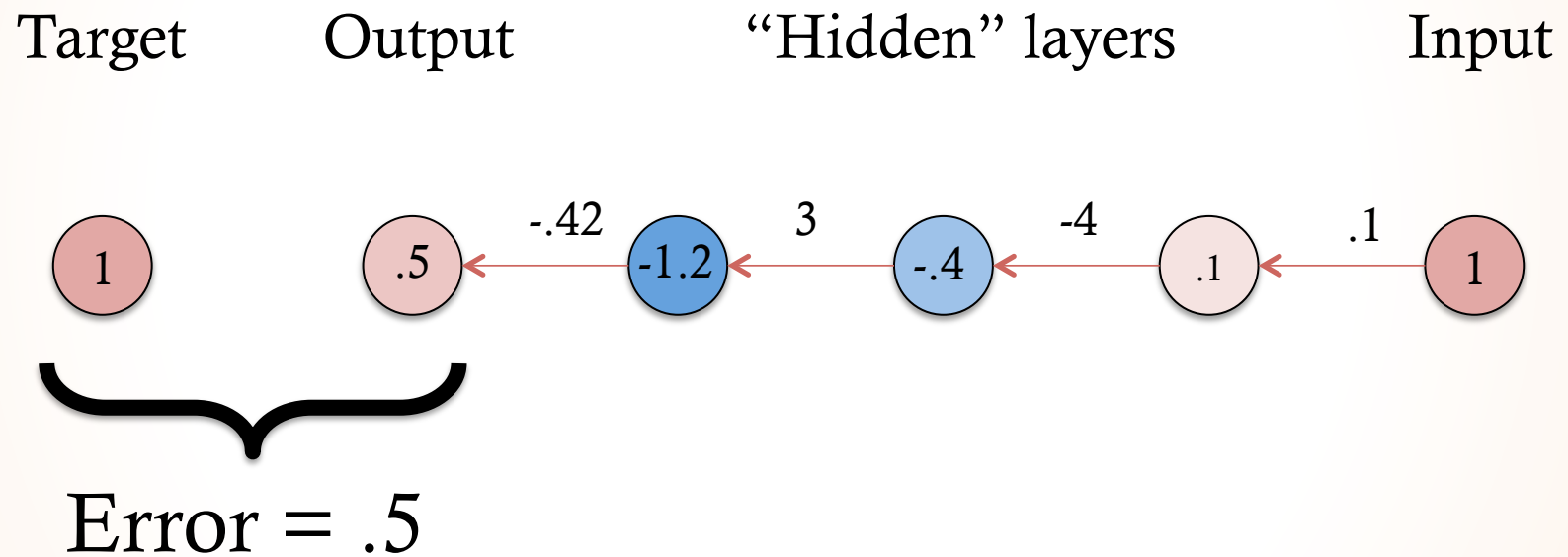
Target feedback



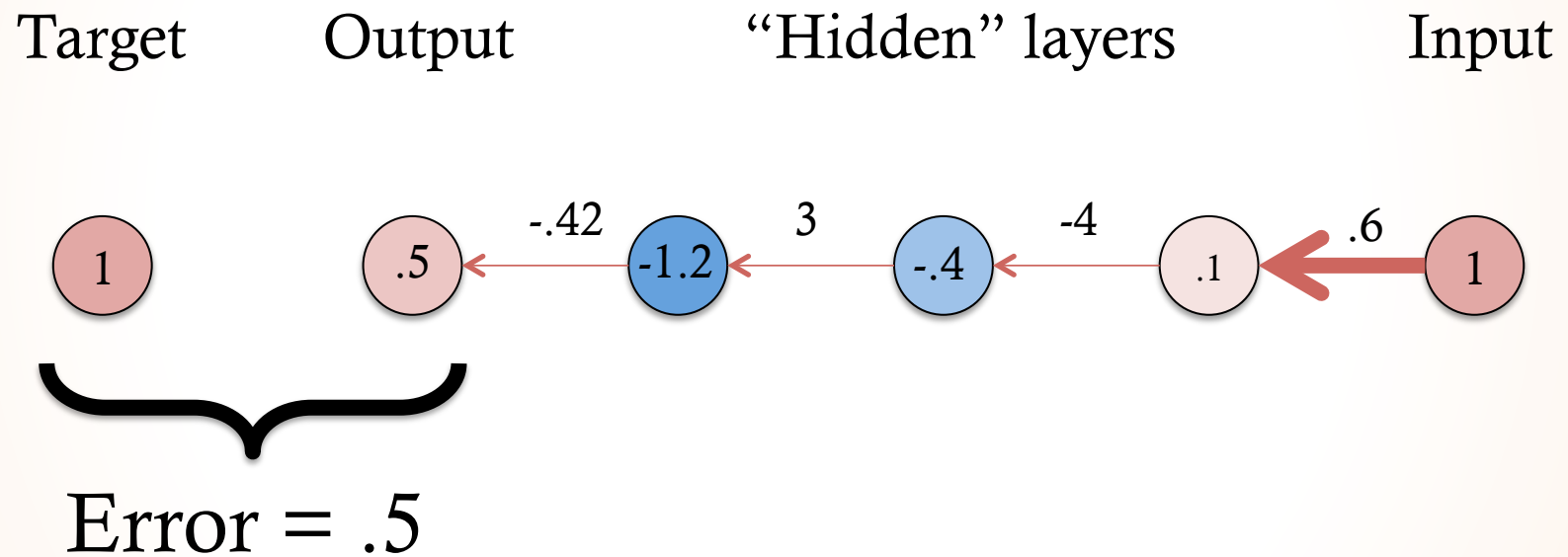
Error signal



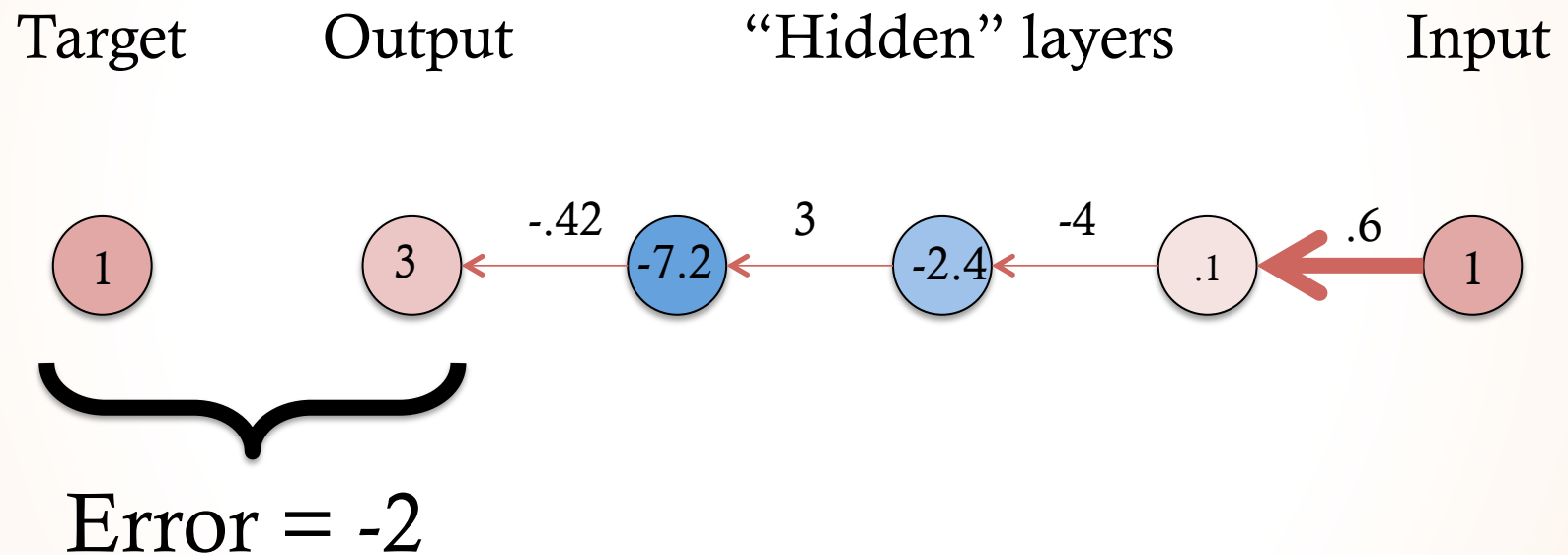
How to fix?



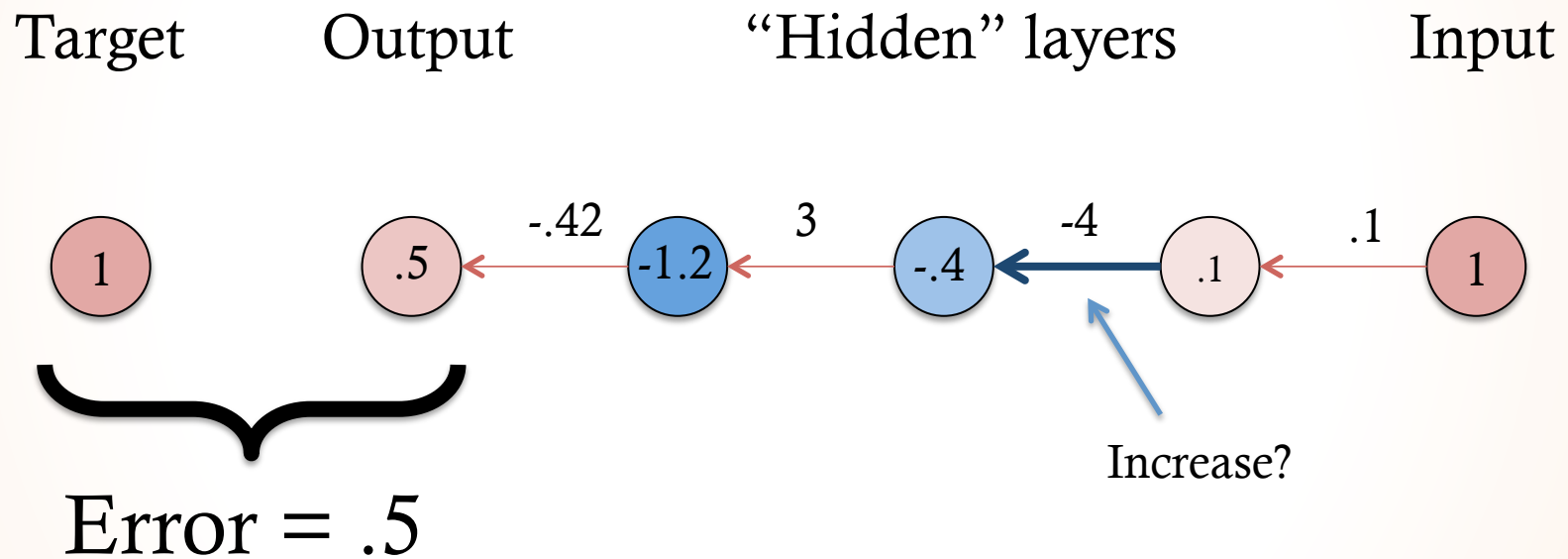
How to fix?



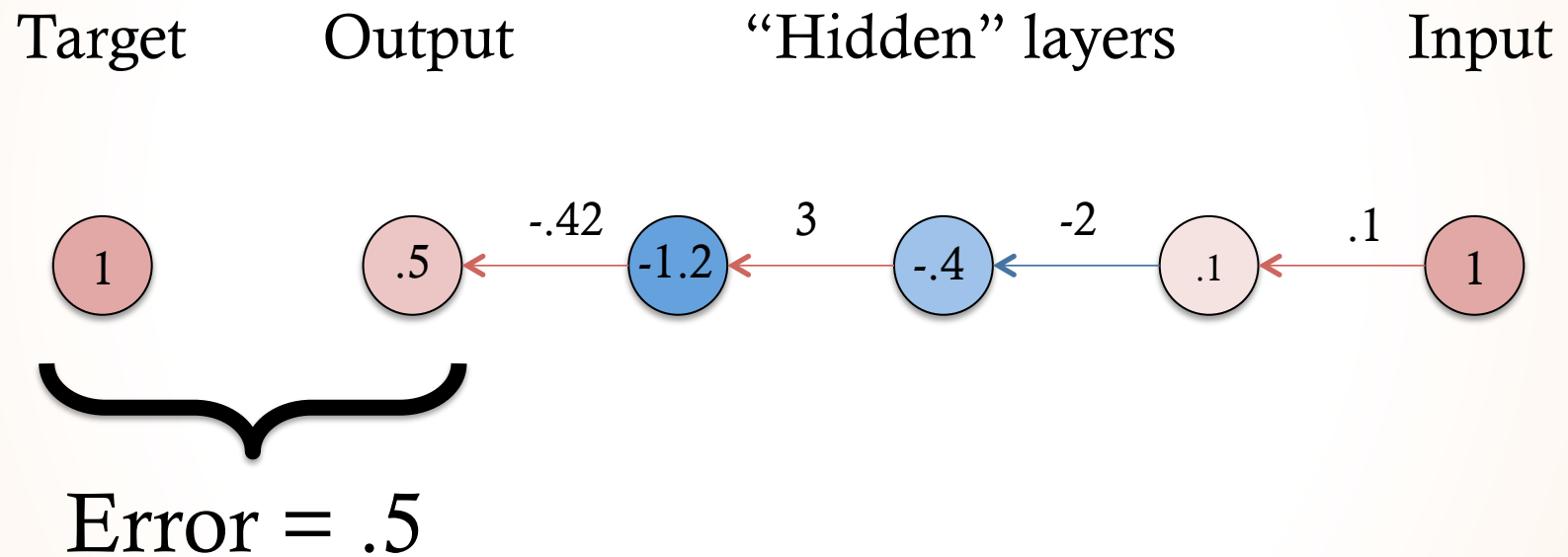
How to fix?



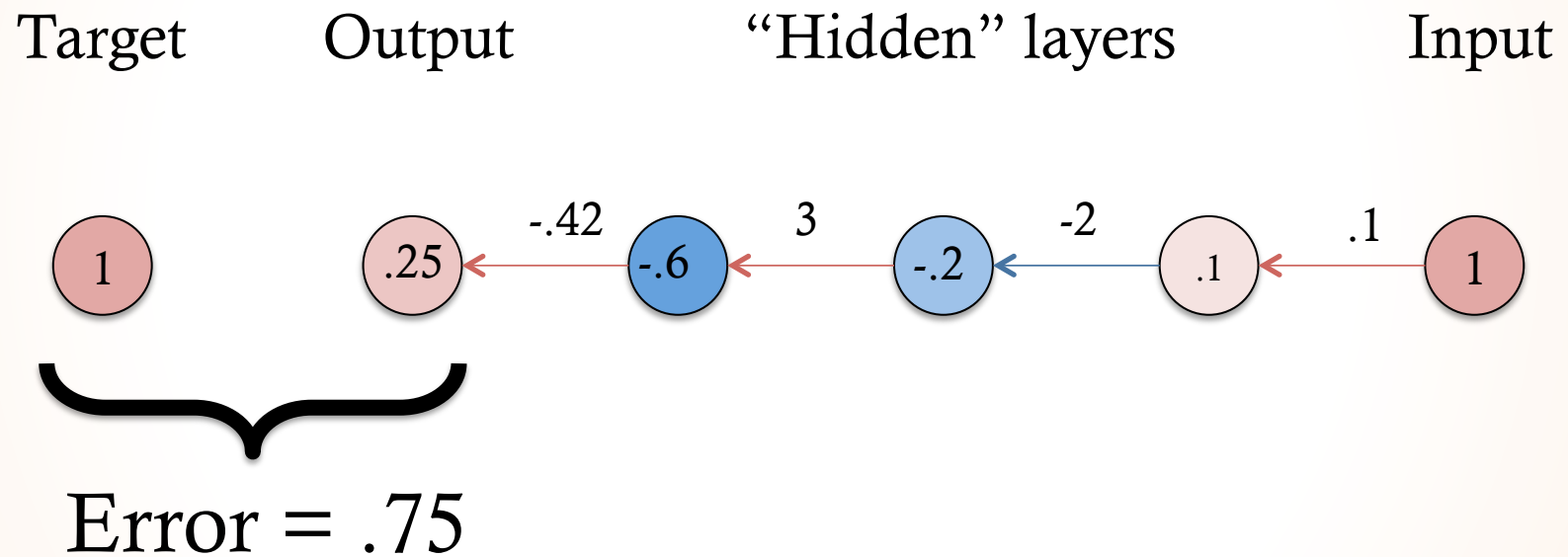
How to fix?



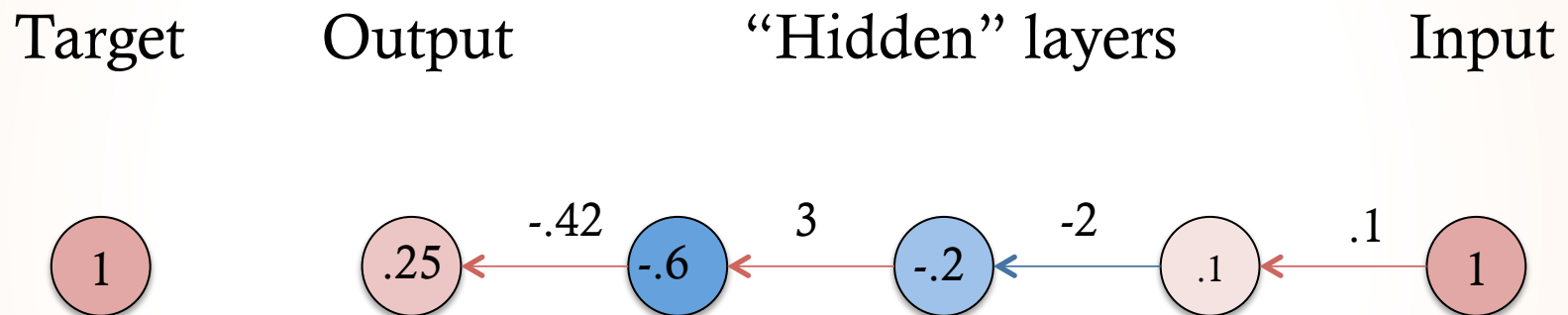
How to fix?



How to fix?



How to fix?

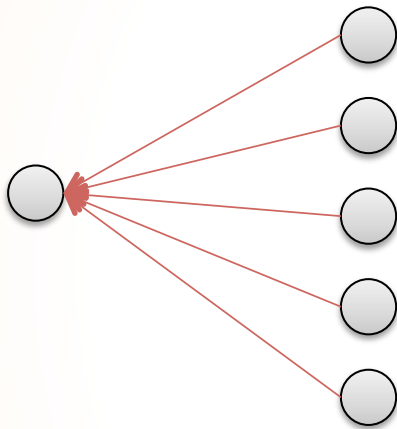


Must consider effect of all upstream weights to know how to change any given weight!

In a deep network, weights are *coupled*

Two fundamental topologies

Parallel

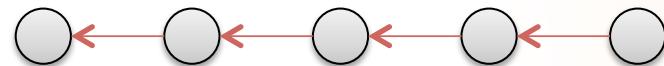


$$o = \sum w_j i_j$$

Sum of variables

Weights are **independent**

Serial



$$o = \left(\prod w_k \right) i$$

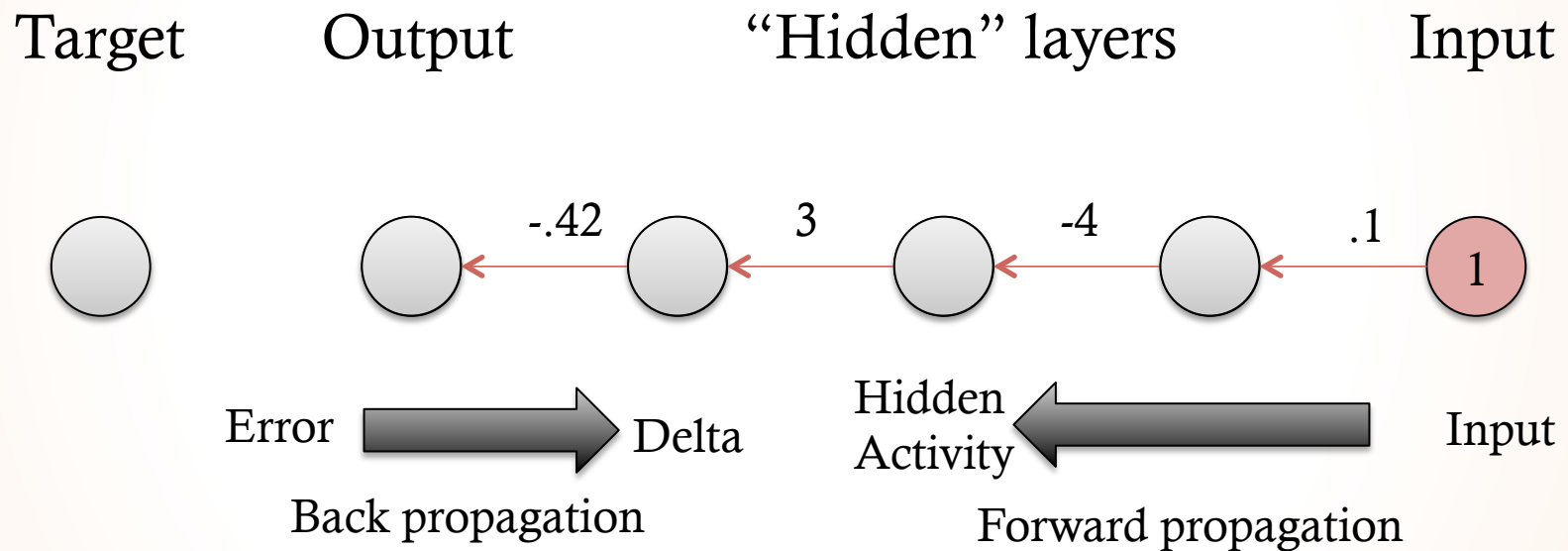
Product of variables

Weights are **coupled**

The solution: Backpropagation

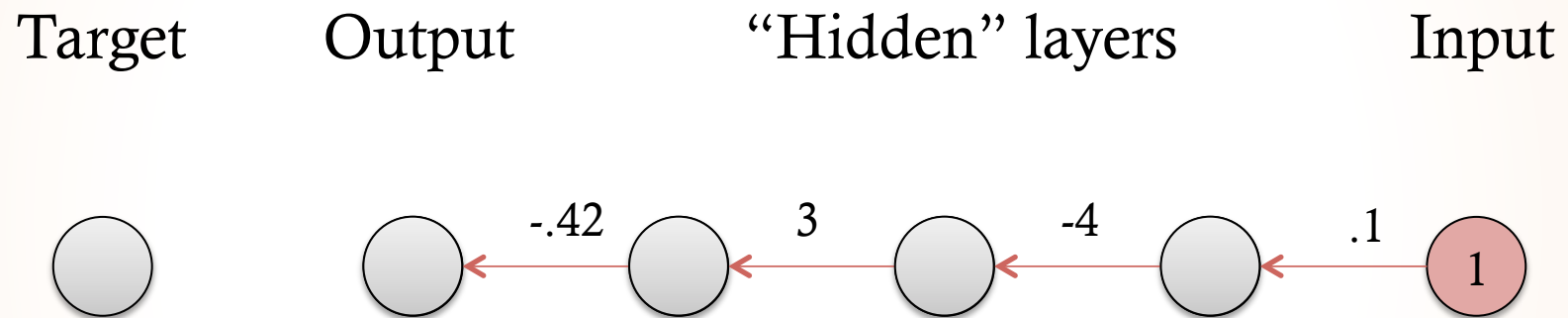
- Send a signal from the output of the network back down towards the input
- This signal will encode how changing a weight will propagate to the output
- Then use delta-like rule as usual

The solution: Backpropagation

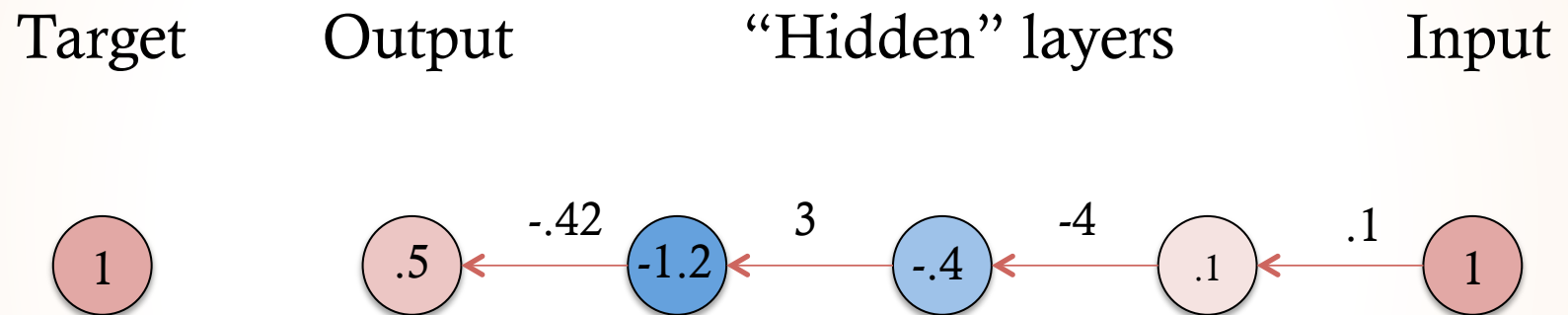


$$\Delta w_j = \lambda \delta_{j+1} h_j$$

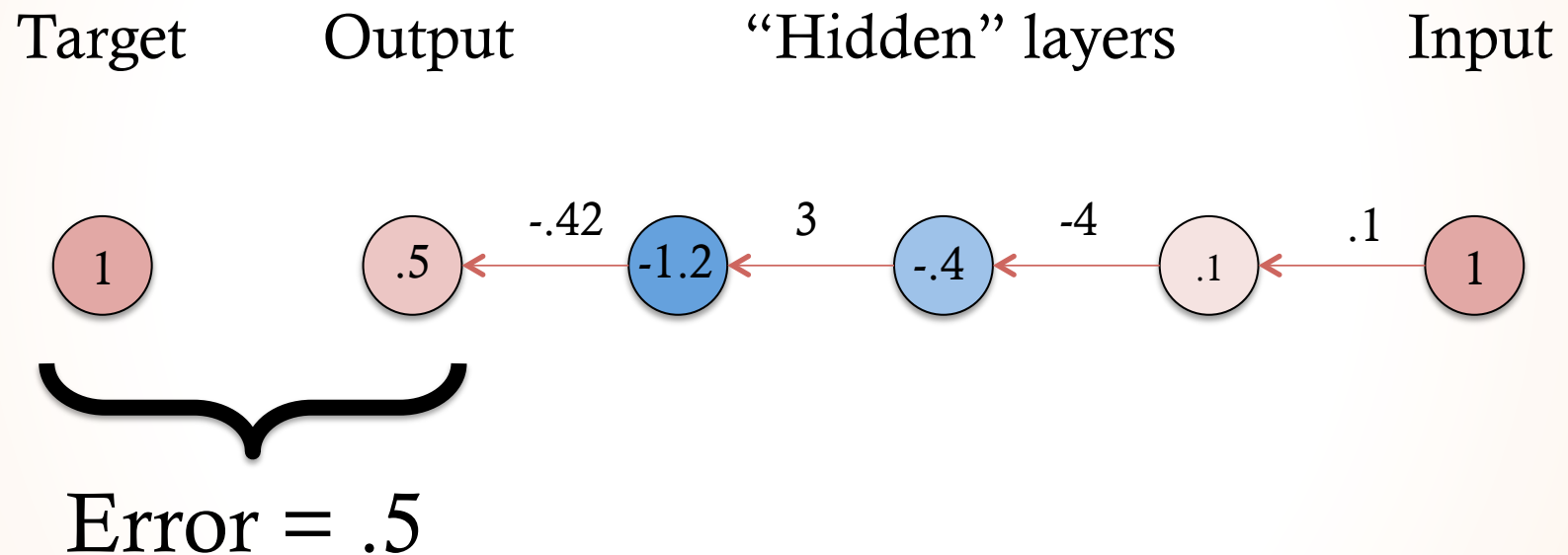
Forward propagation



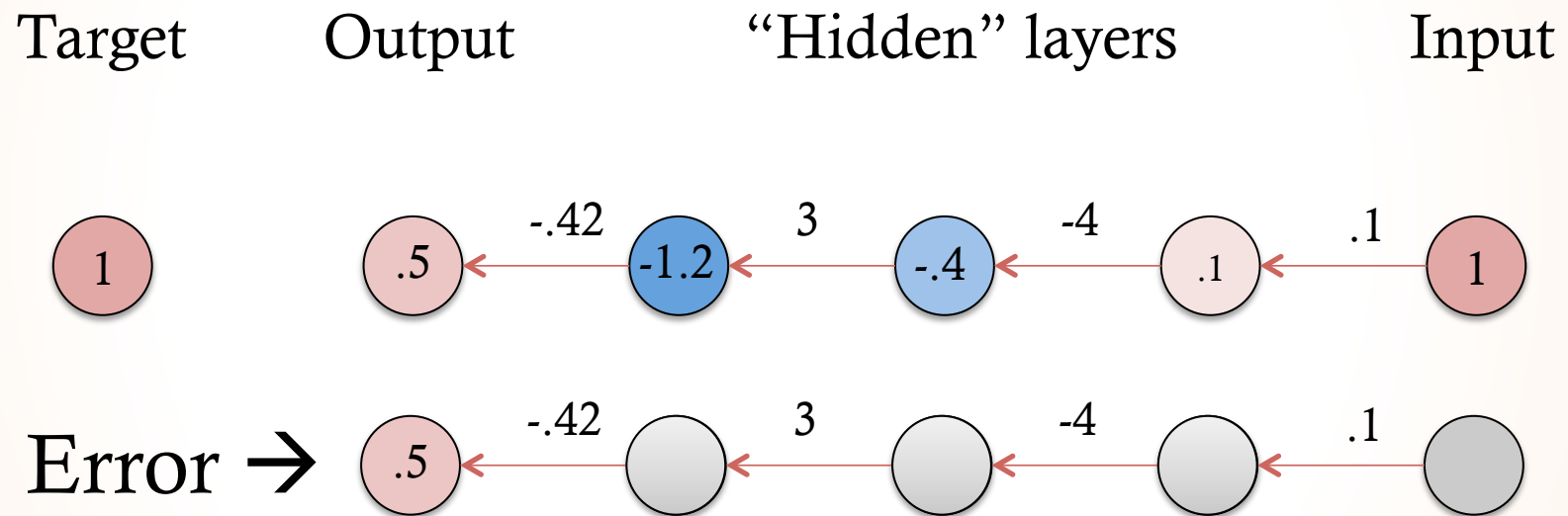
Target feedback



Error signal

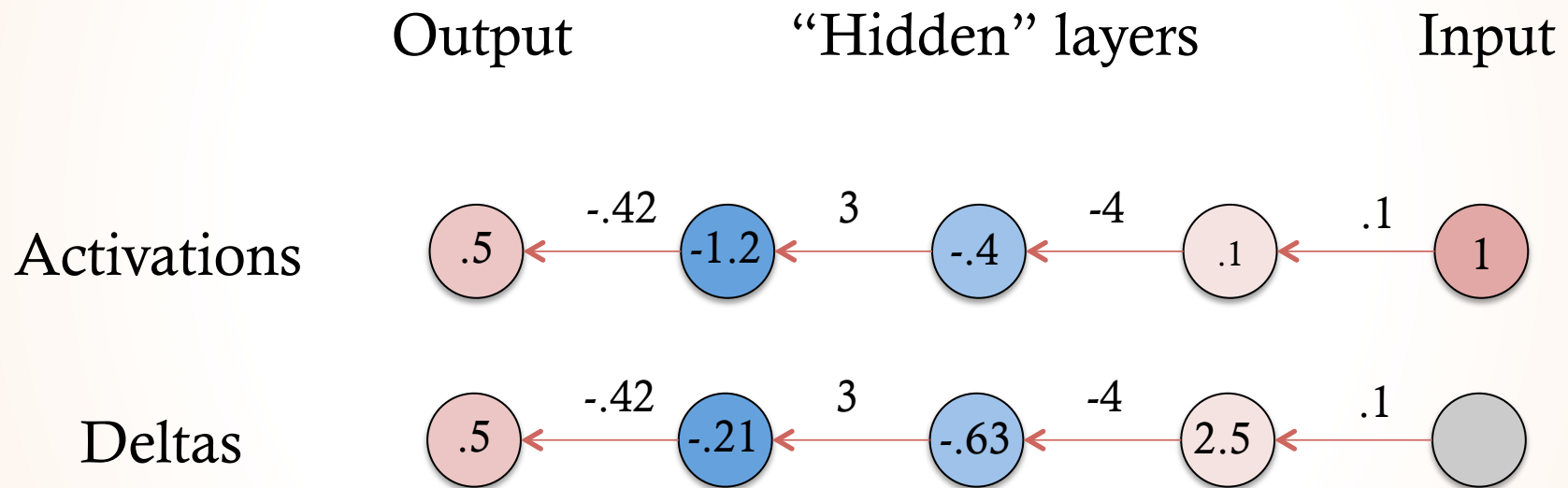


Backpropagation



$$\delta_j = w_{j+1} \delta_{j+1}$$

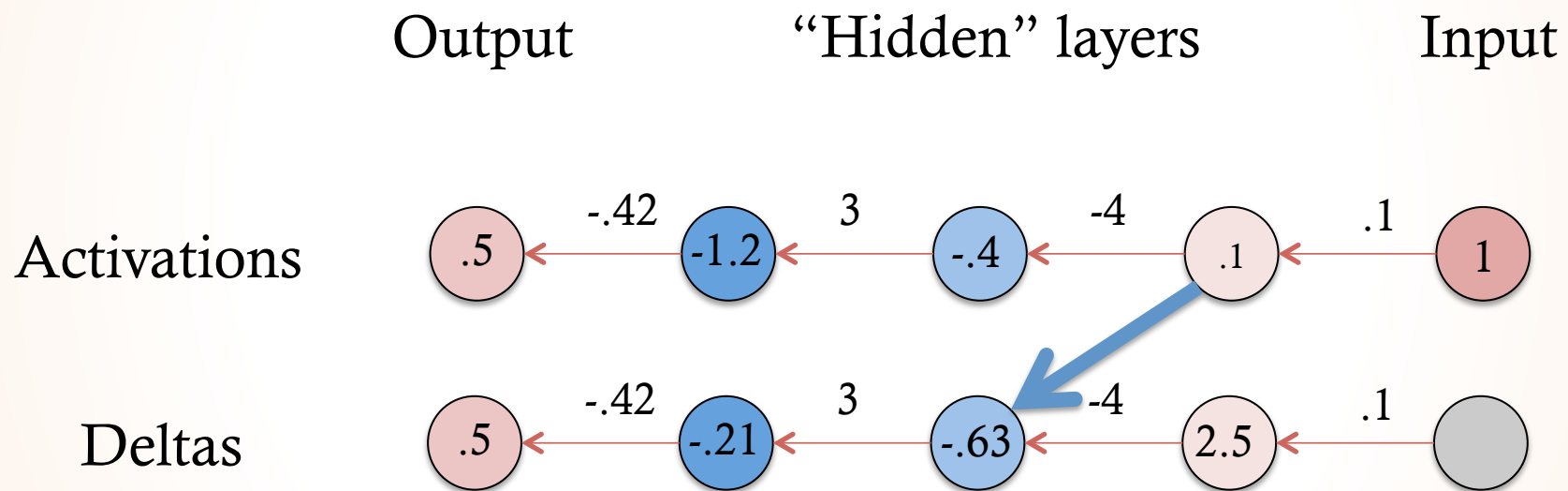
Backpropagation



Learning rule:

$$\Delta w_j = \lambda \delta_{j+1} h_j$$

Backpropagation



Learning rule:

$$\Delta w_j = \lambda \delta_{j+1} h_j$$

Derivation as gradient descent

$$\begin{aligned}\frac{\partial}{\partial w_2}(t - o)^2 &= -2(t - o) \frac{\partial}{\partial w_2} w_4 w_3 w_2 w_1 i \\ &= -2(t - o) \underbrace{w_4 w_3}_{\delta_3} \underbrace{w_1 i}_{h_2}\end{aligned}$$

Learning rule: $\Delta w_j = \lambda \delta_{j+1} h_j$

Delta vs Backprop

Delta rule:

$$\Delta w = \lambda e i^T$$

- Requires actual error and input
- Restricted to one layer

Backprop:

$$\Delta w_j = \lambda \delta_{j+1} h_j$$

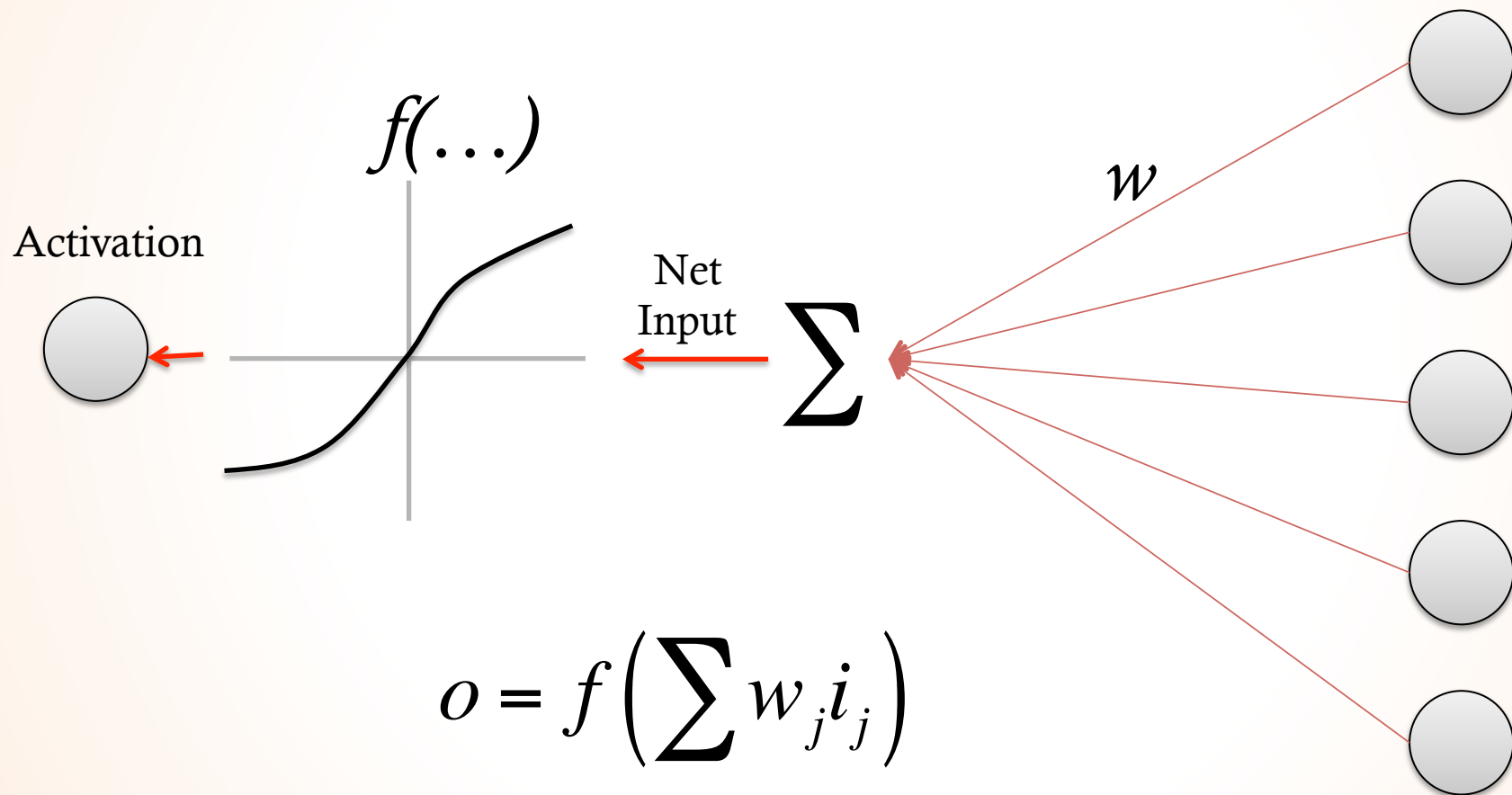
(where j indexes the layer)

- Uses backpropagated error and hidden layer activity
 - Works in deep network
-
- Both implement gradient descent

Nonlinearities

- So far have just looked at linear networks
- Linear networks cannot represent complicated functions (like XOR)
- Introduce neural nonlinearity or “activation function”

Activation function



Gradient learning with nonlinearities

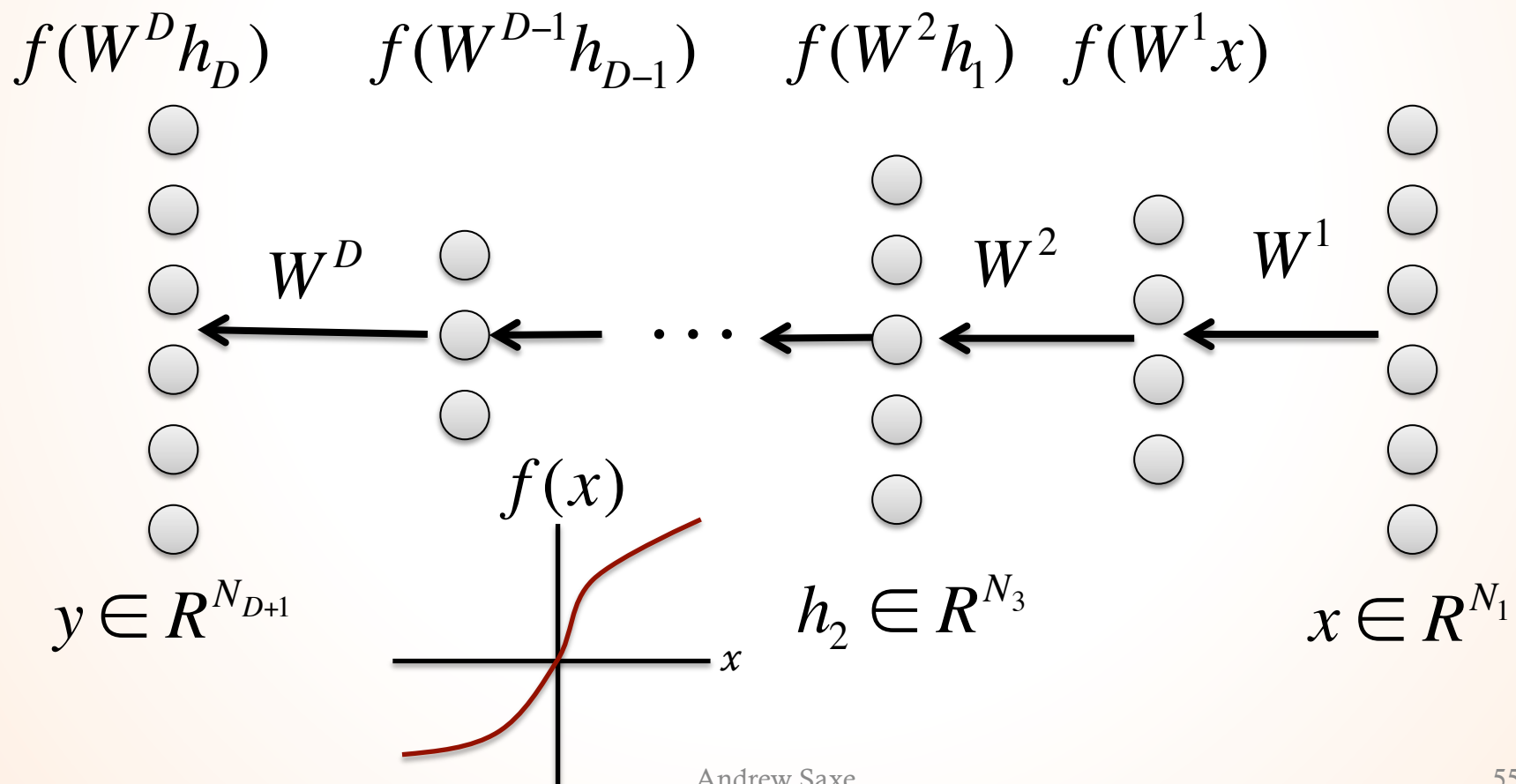
$$\begin{aligned}\frac{\partial E}{\partial w_k} &= -2(t - o) \frac{\partial E}{\partial w_k} f\left(\sum w_j i_j\right) \\ &= -2(t - o) f'(n) \frac{\partial E}{\partial w_k} \sum w_j i_j \\ &= -2(t - o) f'(n) i_k\end{aligned}$$

Scaled delta rule: $\Delta w = \lambda e f'(n) i^T$

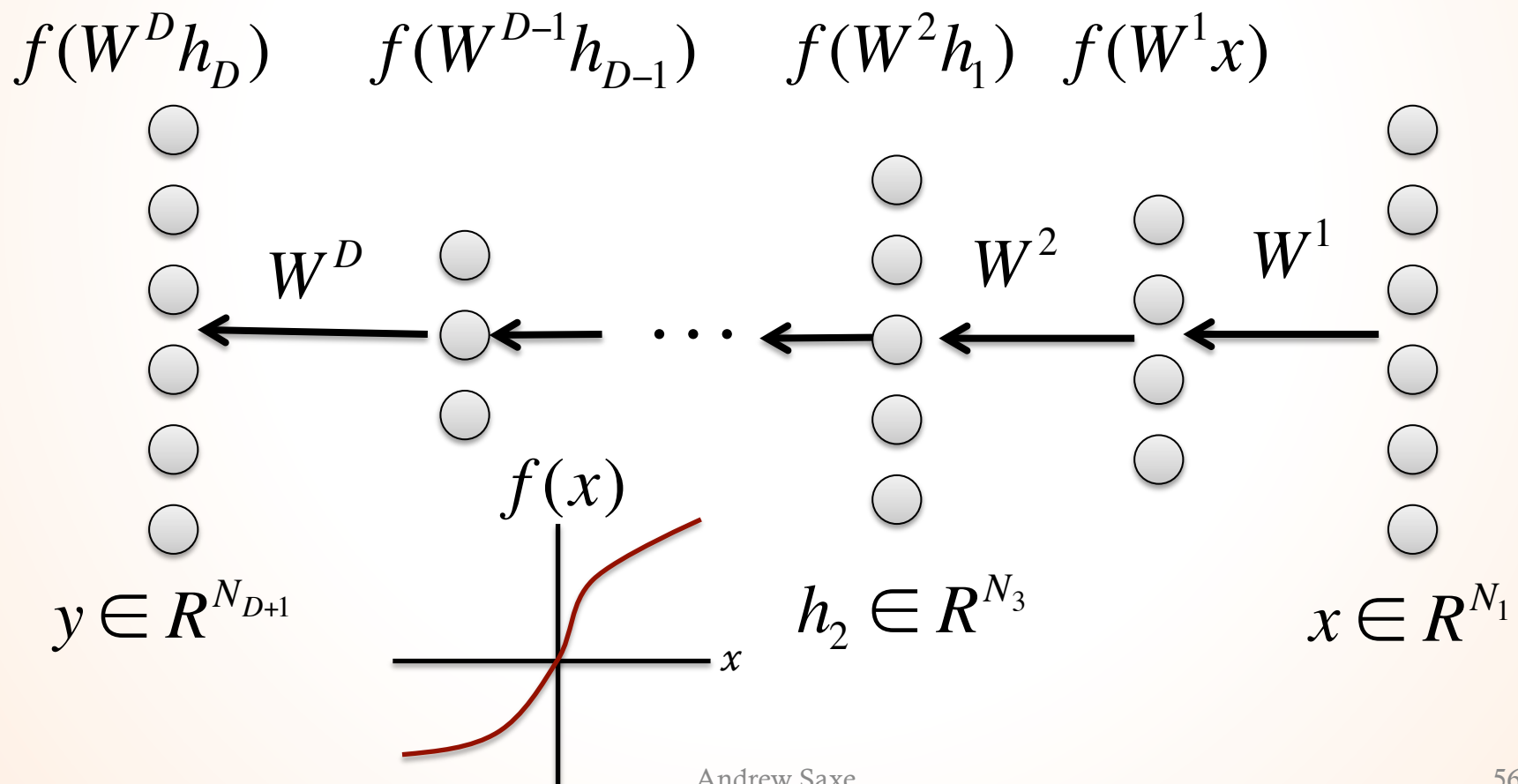
Beyond the chain...

Now for the general case:

a mixture of serial and parallel structure with nonlinearities



Matrix notation



Matrix notation

$$o = f(W^D f(W^{D-1} \dots f(W^2 f(W^1 i)) \dots))$$

Column vector

Column vector

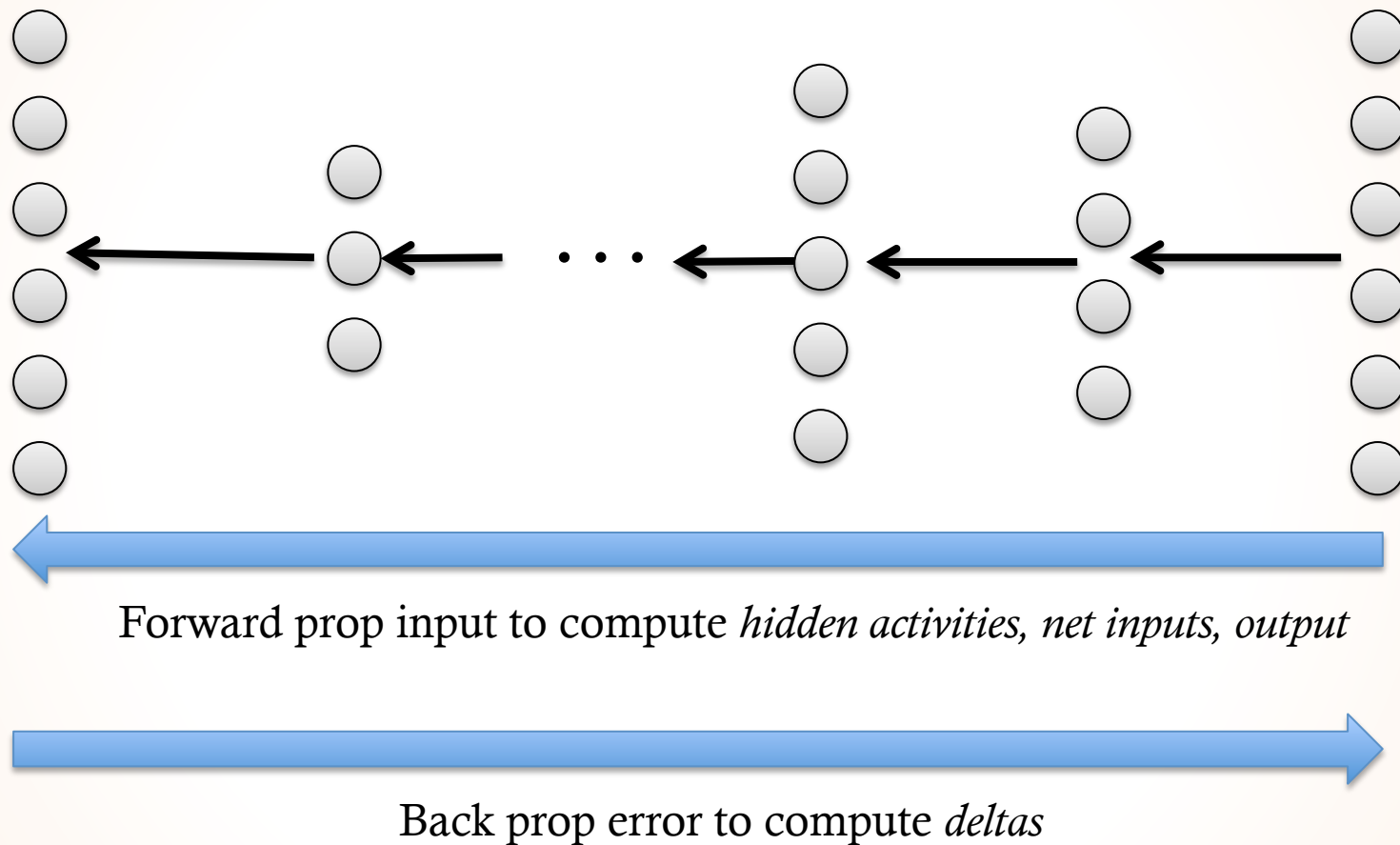
Matrix multiplication

Nonlinearity applied elementwise

Putting the pieces together

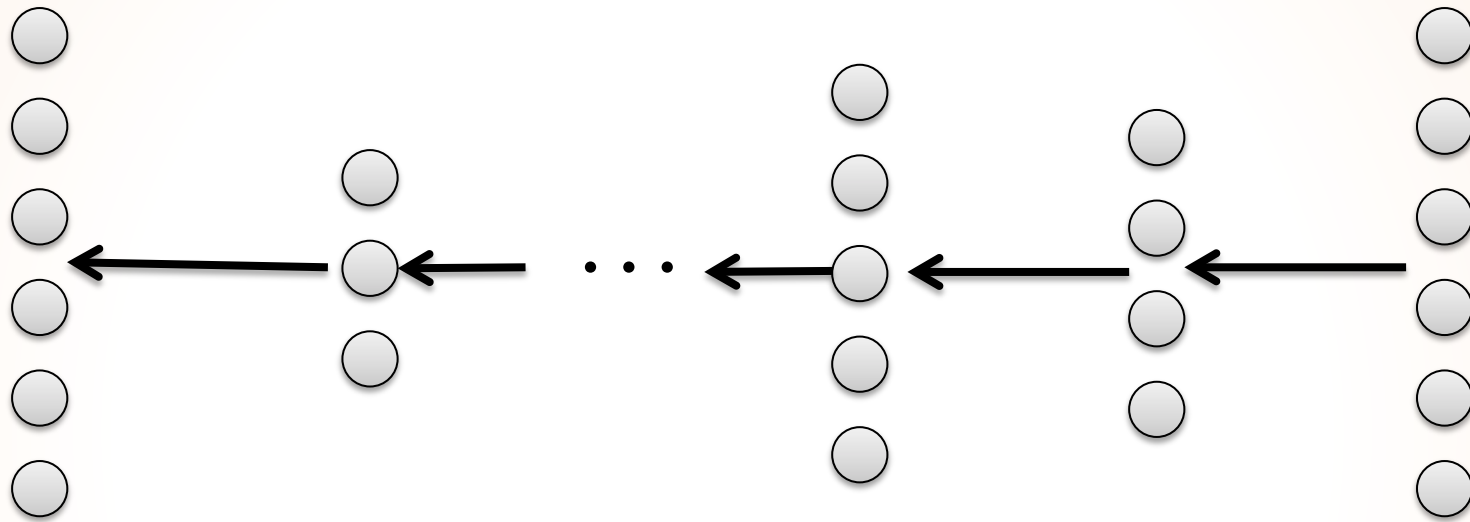
- Single layer parallel: the delta rule
- Multilayer serial: backpropagation
- Nonlinearities: scale by derivative

Putting the pieces together



Learning rule: $\Delta W^j = \lambda \delta_{j+1} h_j^T$

Backpropagation



$$\delta_D = -(t - o) \bullet f'(n_D)$$

$$\delta_j = \left[\left(W^j \right)^T \delta_{j+1} \right] \bullet f'(n_j)$$

Learning rule: $\Delta W^j = \lambda \delta_{j+1} h_j^T$

Summary

- Computational level
 - Optimization view of learning
 - Gradient descent
- Algorithmic level
 - Backprop-as-algorithm
- Implementation level
 - Haven't touched on it
 - Few do!
 - Not at all obvious how to implement in the brain