Backpropagation Learning, Internal Representations and Semantics

Learning

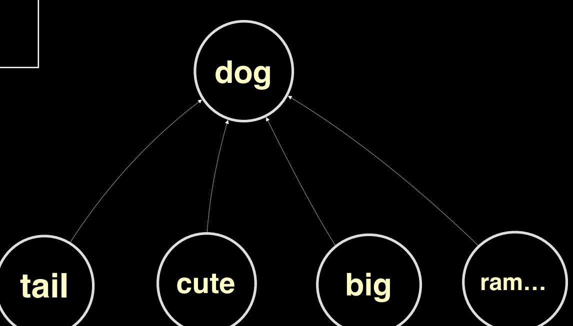
- Unsupervised Learning
  - Hebbian Learning Rule
  - Pattern associator
  - Self-organized maps
  - Topographic structure
  - Pattern detectors

- Supervised Learning
  - Scalar Learning
    - Classical and Instrumental Conditioning
    - Sequential learning and Prediction
  - Vector-Based Learning
    - Generalized Delta Rule
    - Backpropagation
    - Semantics (Deep Learning)

Simple Pattern Associator

Pure Correlational learning:

	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-

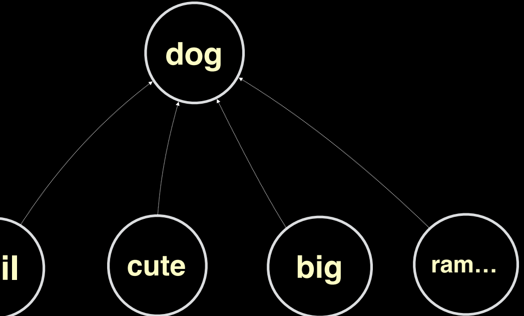


Simple Pattern Associator

Pure Correlational learning:

	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-

- **Observe:**
  - *dog* **is uncorrelated with** *tail*, *cute*, **and** *rambunctious*

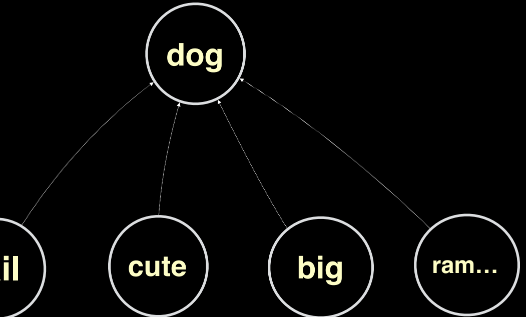


Simple Pattern Associator

Pure Correlational learning:

	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-

- **Observe:**
  - *dog* is uncorrelated with *tail*, *cute*, and *rambunctious*
  - ∴ weights from these units to *dog* will be 0

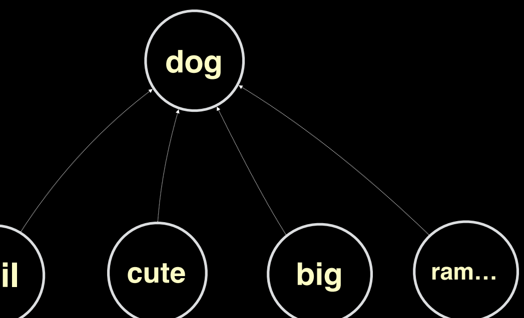


Simple Pattern Associator

Pure Correlational learning:

	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-

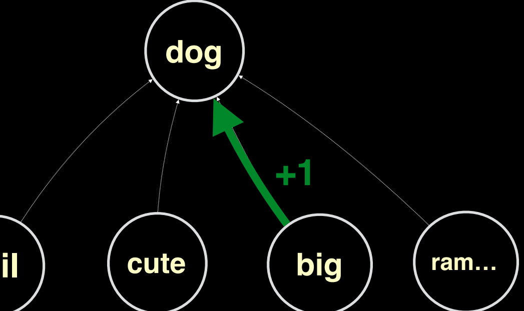
- Observe:
  - *dog* is uncorrelated with *tail*, *cute*, and *rambunctious*
  - ∴ weights from these units to *dog* will be 0
  - *dog* is positively correlated with *big*



Simple Pattern Associator

Pure Correlational learning:

	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-

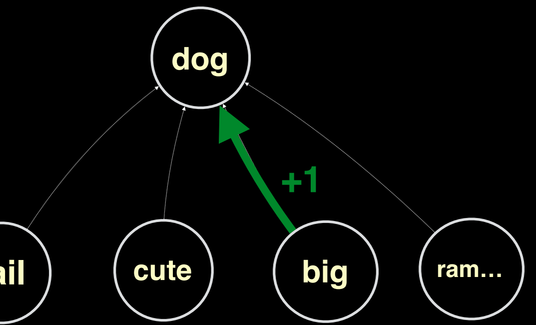


- **Observe:**
  - *dog* is uncorrelated with *tail*, *cute*, and *rambunctious*
  - ∴ weights from these units to *dog* will be 0
  - *dog* is positively correlated with *big*
  - ∴ a positive weight will develop between *big* and *dog*, and *dog* will be positive for *Felix* when it should be negative

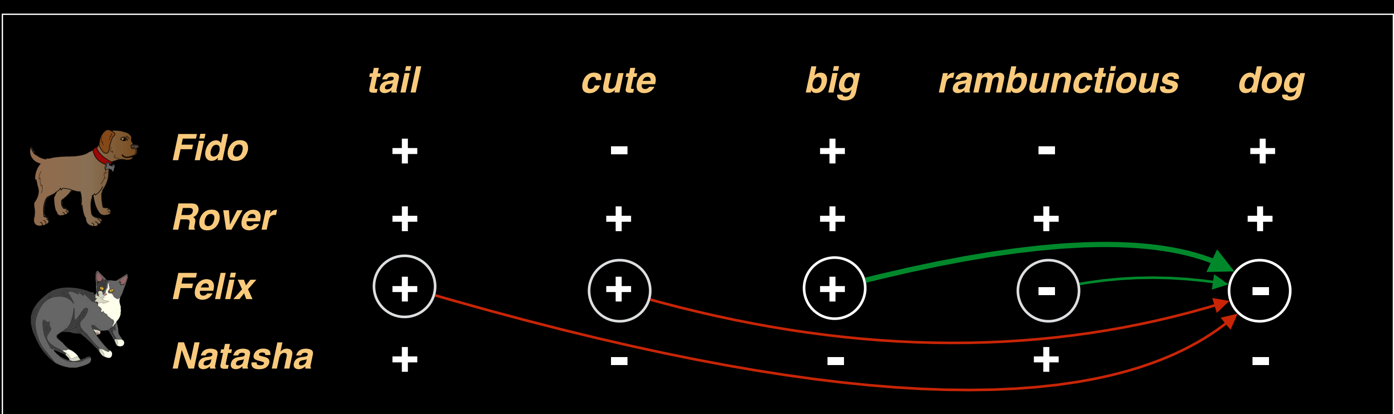
Simple Pattern Associator

	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-

- There is a set of weights that will do the trick

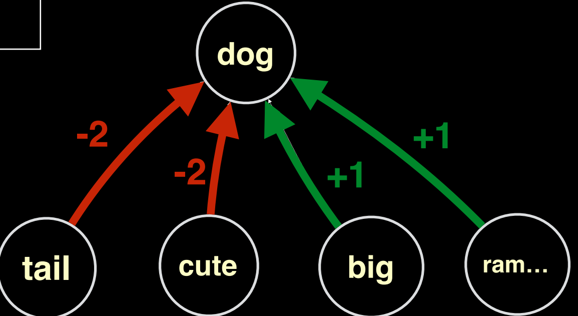


Simple Pattern Associator

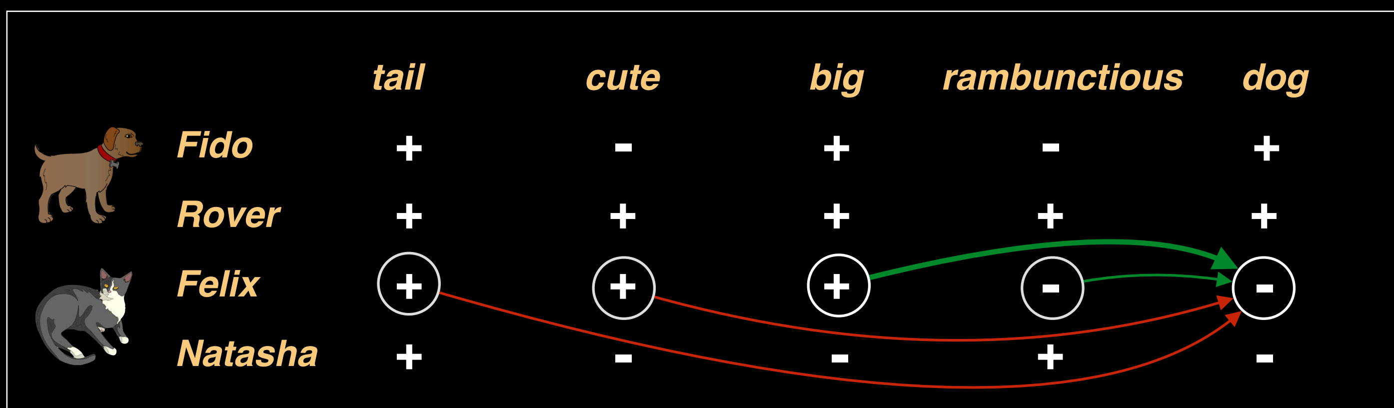


	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-

- There is a set of weights that will do the trick

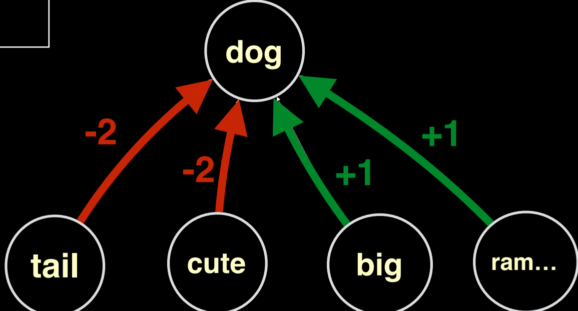


Simple Pattern Associator



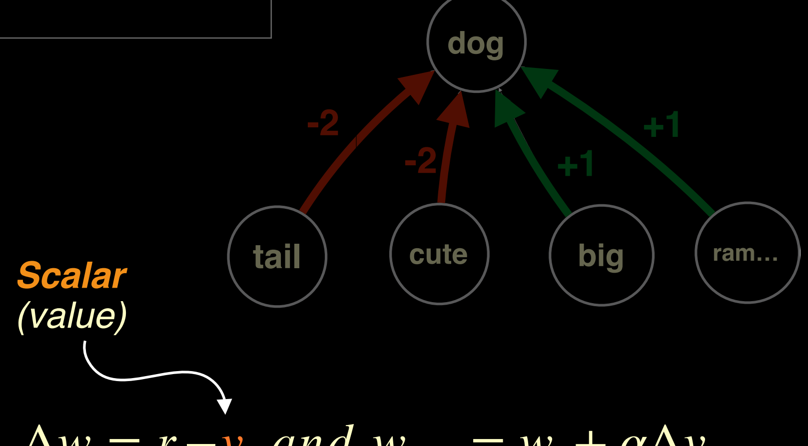
	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-

- There is a set of weights that will do the trick
  - how can these be learned?



Simple Pattern Associator

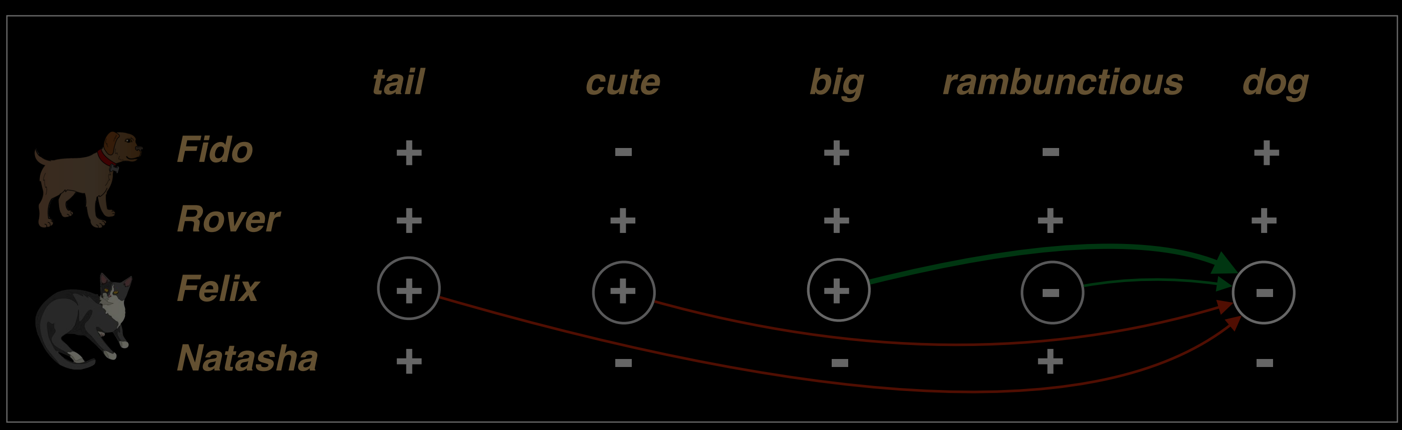
- There is a set of weights that will do the trick
  - how can these be learned?
- **Prediction (error-driven) learning:**
  - **Reinforcement learning** *(Rescorla-Wagner)*:



Scalar *(value)*

$$\Delta w = r_t - v_t \ and \ w_{t+1} = w_t + \alpha \Delta v$$

Simple Pattern Associator



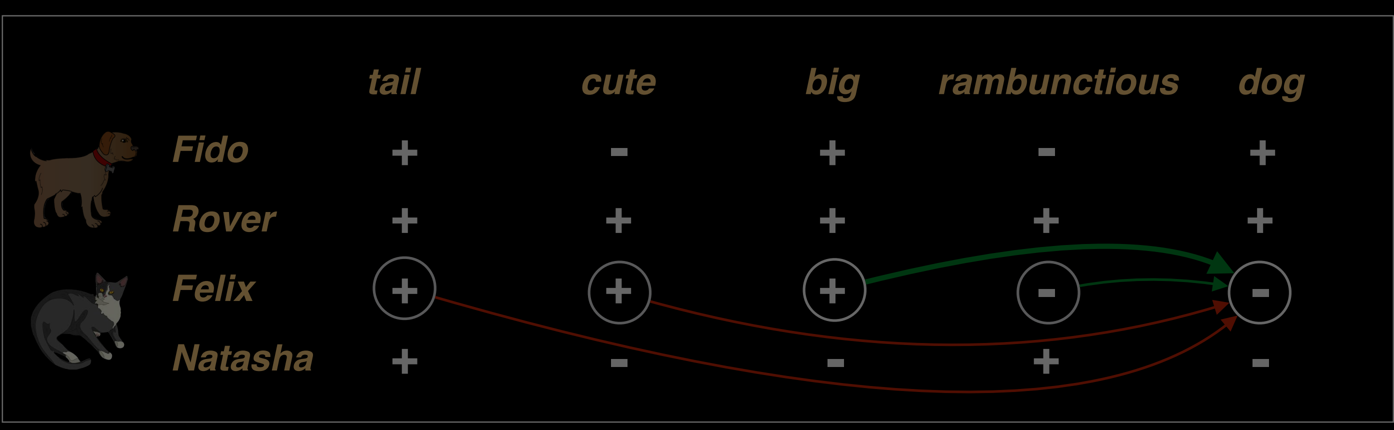
	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-

dog
-2
-2
+1
+1
tail
cute
big
ram…

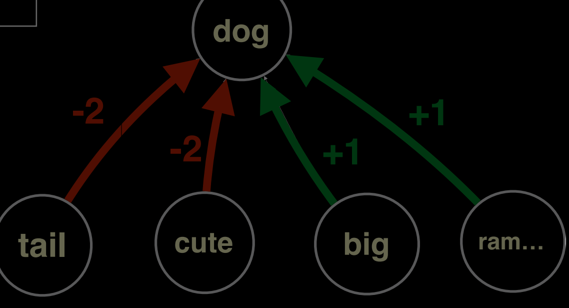
- There is a set of weights that will do the trick
  - how can these be learned?
- **Prediction (error-driven) learning:**
  - Reinforcement learning *(Rescorla-Wagner)*: $\Delta w = r_t - v_t \; and \; w_{t+1} = w_t + \alpha \Delta v$
  - **Delta rule** *(Widrow-Hoff)*: $\Delta w = r_t - a_t \; and \; w_{t+1} = w_t + \alpha \Delta v$

Vector *(activity)*

Simple Pattern Associator



	tail	cute	big	rambunctious	dog
Fido	+	-	+	-	+
Rover	+	+	+	+	+
Felix	+	+	+	-	-
Natasha	+	-	-	+	-



- There is a set of weights that will do the trick
  - how can these be learned?
- Prediction (error-driven) learning:

  - Reinforcement learning *(Rescorla-Wagner)*: $\Delta w = r_t - v_t \; and \; w_{t+1} = w_t + \alpha \Delta v$
  - **Delta rule** *(Widrow-Hoff)*: $\Delta w = r_t - a_t \; and \; w_{t+1} = w_t + \alpha \Delta v$
- Can be extended to train many output units…

Vector (activity)

Delta Rule

(also known as the Widrow Hoff or
Least Mean Square (LMS) learning rule)

Delta Rule

(also known as the Widrow Hoff or Least Mean Square (LMS) learning rule)

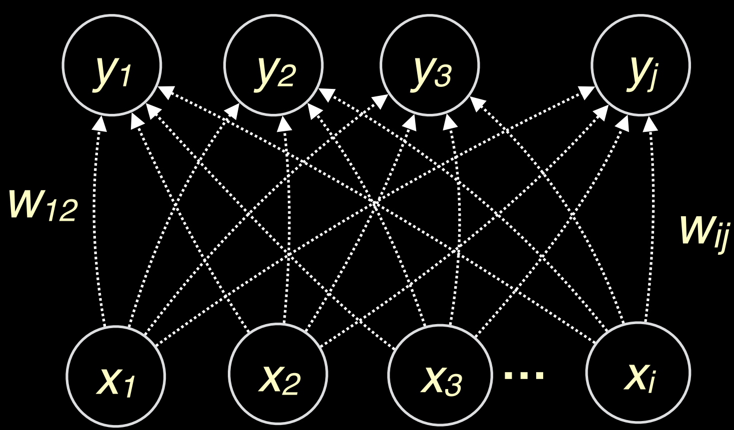
- **Generalization of Rescorla-Wagner to many output units:**

Delta Rule

(also known as the Widrow Hoff or
Least Mean Square (LMS) learning rule)

- **Generalization of Rescorla-Wagner to many output units:**

$$\Delta w_{ij} = \epsilon e_j x_j$$



Delta Rule

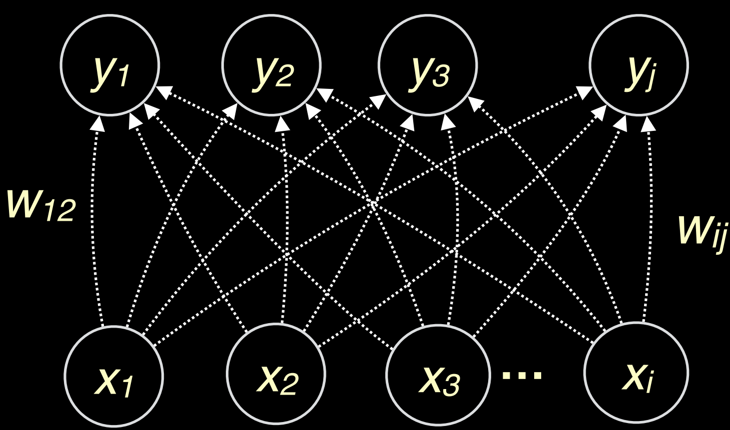
(also known as the Widrow Hoff or
Least Mean Square (LMS) learning rule)

- **Generalization of Rescorla-Wagner to many output units:**

$$\Delta w_{ij} = \epsilon e_j x_j$$

where e_j is the error for output unit_i:

$$e_j = t_j - y_j$$



Delta Rule

(also known as the Widrow Hoff or
Least Mean Square (LMS) learning rule)

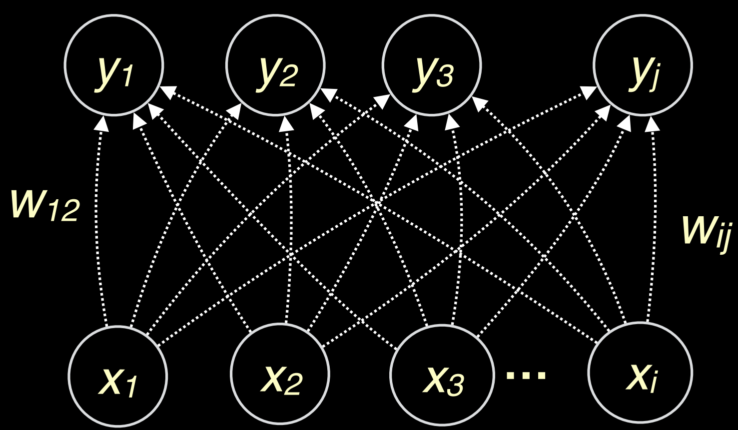
- **Generalization of Rescorla-Wagner to many output units:**

$$\Delta w_{ij} = \epsilon e_j x_j$$

where e_j is the error for output unit$_i$:

$$e_j = t_j - y_j$$

t_j is the target activation for y_j,
(in the training pattern)



Delta Rule

(also known as the Widrow Hoff or
Least Mean Square (LMS) learning rule)

- **Generalization of Rescorla-Wagner to many output units:**

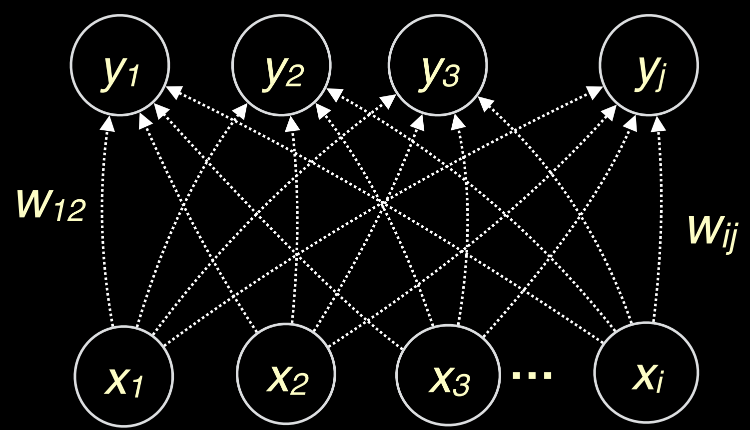
$$\Delta w_{ij} = \epsilon e_j x_j$$

where e_j is the error for output $unit_i$:

$$e_j = t_j - y_j$$

t_j is the target activation for y_j,
(in the training pattern)

and y_j is its actual activity



Delta Rule

(also known as the Widrow Hoff or
Least Mean Square (LMS) learning rule)

- **Generalization of Rescorla-Wagner to many output units:**

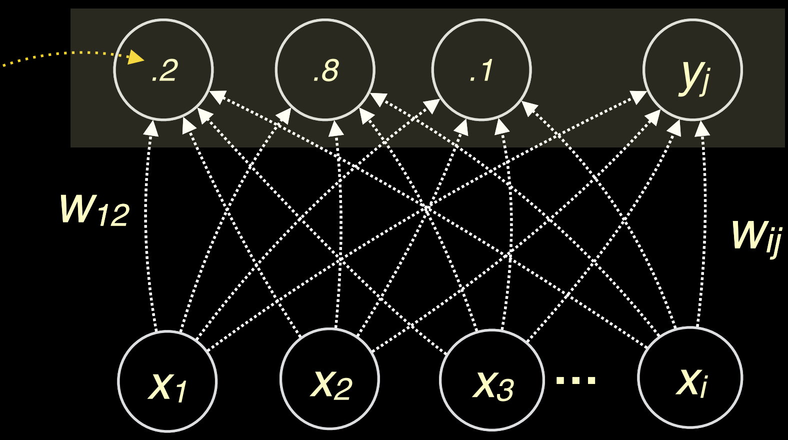
$$\Delta w_{ij} = \epsilon e_j x_j$$

where e_j is the error for output unit$_i$:

$$e_j = t_j - y_j$$

t_j is the target activation for y_j,
(in the training pattern)

and y_j is its actual activity



Delta Rule

(also known as the Widrow Hoff or Least Mean Square (LMS) learning rule)

- **Generalization of Rescorla-Wagner to many output units:**

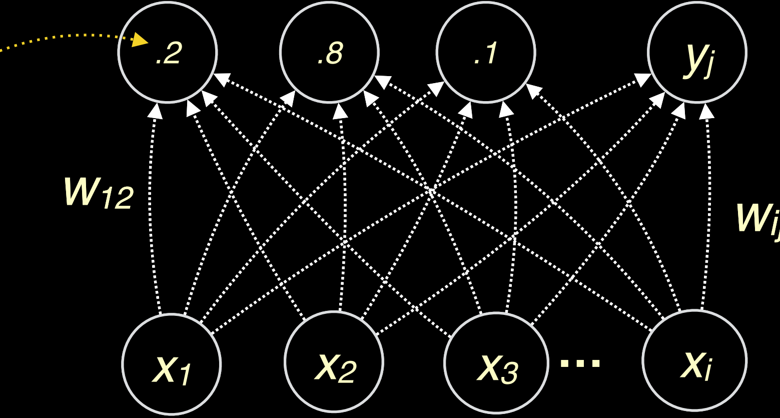
$$\Delta w_{ij} = \epsilon e_j x_j$$

where e_j is the error for output unit$_i$:

$$e_j = t_j - y_j$$

t_j is the target activation for y_j, *(in the training pattern)*

and y_j is its actual activity



Delta Rule

(also known as the Widrow Hoff or
Least Mean Square (LMS) learning rule)

- **Generalization of Rescorla-Wagner to many output units:**

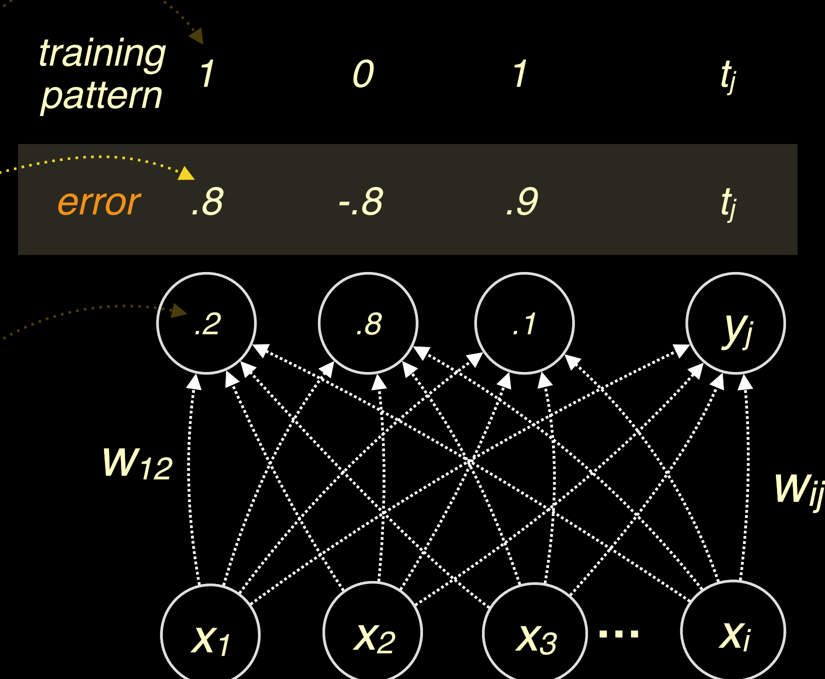
$$\Delta w_{ij} = \epsilon e_j x_j$$

where e_j is the error for output unit$_i$:

$$e_j = t_j - y_j$$

t_j is the target activation for y_j,
(in the training pattern)

and y_j is its actual activity



Training Procedure

Training Procedure

- Two phases:
  1. Activation
  2. Error computation and weight adjustment

Training Procedure

- Two phases:
  1. Activation
  2. Error computation and weight adjustment
- For each pattern:
  - Turn on input units

Training Procedure

- Two phases:
  1. Activation
  2. Error computation and weight adjustment

- For each pattern:
  – Turn on input units
  – Determine the resulting output

Training Procedure

- Two phases:
  1. Activation
  2. Error computation and weight adjustment

- For each pattern:
  - Turn on input units
  - Determine the resulting output
  - Compute error: difference between target and actual output

Training Procedure

- Two phases:
  1. Activation
  2. Error computation and weight adjustment

- For each pattern:
  - Turn on input units
  - Determine the resulting output
  - Compute error: difference between target and actual output
  - Adjust strength of each weight according to the delta rule

Training Procedure

- Two phases:
  1. Activation
  2. Error computation and weight adjustment
- For each pattern:
  - Turn on input units
  - Determine the resulting output
  - Compute error: difference between target and actual output
  - Adjust strength of each weight according to the delta rule
- Epoch:
  - One pass of training through all patterns

Training Procedure

- Two phases:
  1. Activation
  2. Error computation and weight adjustment
- For each pattern:
  - Turn on input units
  - Determine the resulting output
  - Compute error: difference between target and actual output
  - Adjust strength of each weight according to the delta rule
- Epoch:
  - One pass of training through all patterns
- Performance measures:
  - ***Sum squared error*** *(SSE)* **over output units y_j for pattern p (pss):** $\Sigma_j (t_j - y_j)^2$

Training Procedure

- Two phases:
  1. Activation
  2. Error computation and weight adjustment

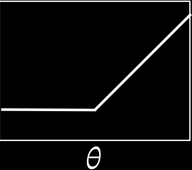
- For each pattern:
  - Turn on input units
  - Determine the resulting output
  - Compute error: difference between target and actual output
  - Adjust strength of each weight according to the delta rule

- Epoch:
  - One pass of training through all patterns

- Performance measures:
  - ***Sum squared error*** *(SSE)* **over output units y_j for pattern p (pss):** $\Sigma_j(t_j - y_j)^2$
  - ***Total sum of squares*** **(tss) over a set of patterns:** $\Sigma_p SSE_p$

The Perceptron

Frank Rosenblatt, 1957

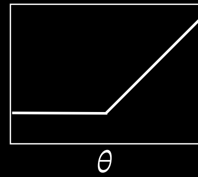


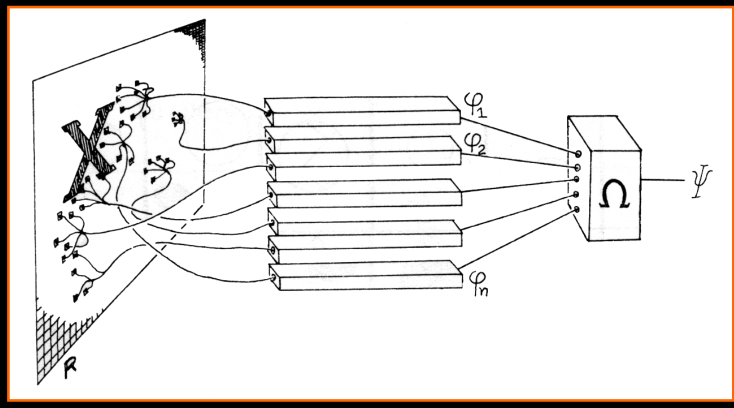
The Perceptron

Frank Rosenblatt, 1957

- **Perceptron**
  - **binary classifier**
  - **linearly thresholded units:**
    aka Rectified Linear Units (ReLU)

$$\text{Output} = \begin{cases} 1 \text{ if net} = \Sigma_i w_i i_i > \theta \\ 0 \text{ otherwise} \end{cases}$$



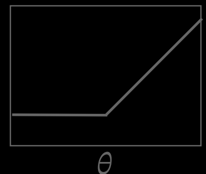


The Perceptron

Frank Rosenblatt, 1957

- **Perceptron**
  - **binary classifier**
  - **linearly thresholded units:**
    aka Rectified Linear Units (ReLU)

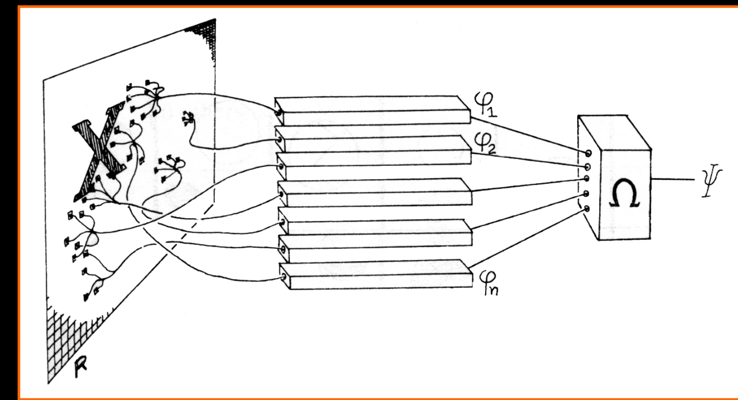
    Output = $\begin{cases} 1 \text{ if net} = \Sigma_i w_i i_i > \theta \\ 0 \text{ otherwise} \end{cases}$



  - **learning rule (similar to the Delta rule):**

$$\Delta\theta = -(t_p - o_p) = -\delta_p \quad \textbf{(threshold)}$$

$$\Delta w_i = (t_p - o_p)i_{pi} = \delta_p i_p \quad \textbf{(weights)}$$

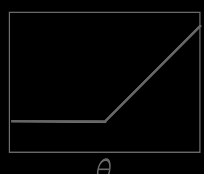


The Perceptron

Frank Rosenblatt, 1957

- **Perceptron**
  - **binary classifier**
  - **linearly thresholded units:**
    aka Rectified Linear Units (ReLU)

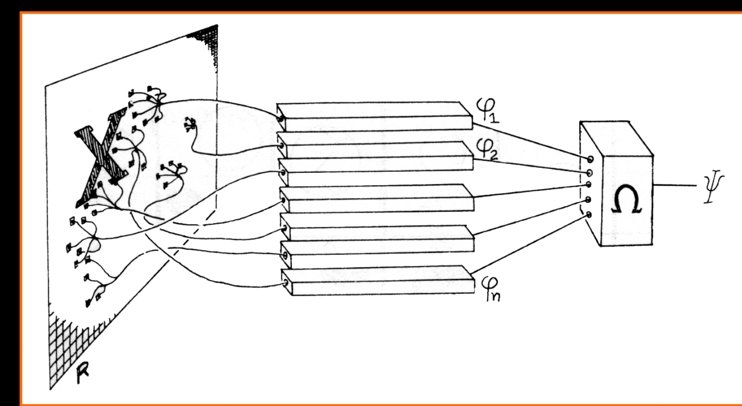
    Output = 1 if net = $\Sigma_i w_i i_i > \theta$; 0 otherwise

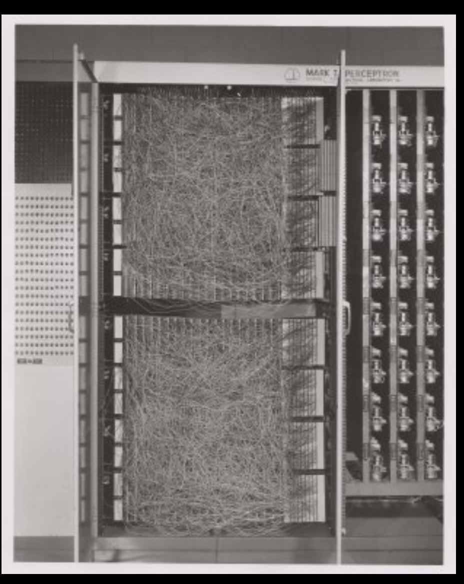


  - **learning rule (similar to the Delta rule):**

$$\Delta\theta = -(t_p - o_p) = -\delta_p \quad \textbf{(threshold)}$$

$$\Delta w_i = (t_p - o_p) i_{pi} = \delta_p i_p \quad \textbf{(weights)}$$





Mark 1 Perceptron

Input:
400 photocells

weights:
potentiometers

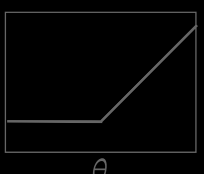
weight updates:
electric motors

The Perceptron

Frank Rosenblatt, 1957

- **Perceptron**
  - **binary classifier**
  - **linearly thresholded units:**
    aka Rectified Linear Units (ReLU)

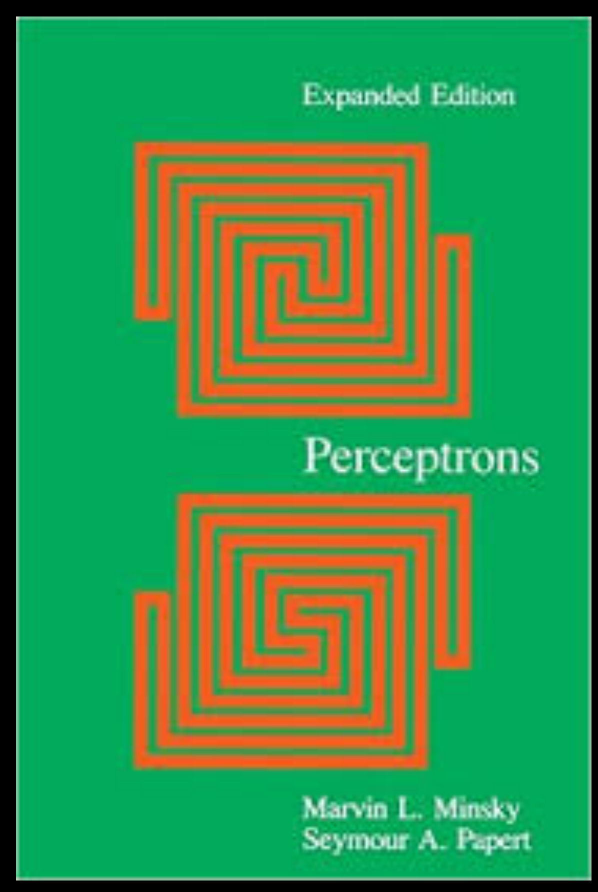
    Output = 1 if net = $\Sigma_i w_i i_i > \theta$
    0 otherwise



  - **learning rule (similar to the Delta rule):**

$$\Delta\theta = -(t_p - o_p) = -\delta_p \quad \text{(threshold)}$$

$$\Delta w_i = (t_p - o_p) i_{pi} = \delta_p i_p \quad \text{(weights)}$$

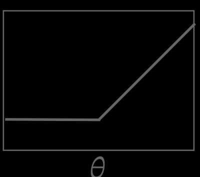


The Perceptron

Frank Rosenblatt, 1957

- **Perceptron**
  - **binary classifier**
  - **linearly thresholded units:**
    aka Rectified Linear Units (ReLU)

    Output = 1 if net = $\Sigma_i w_i i_i > \theta$; 0 otherwise

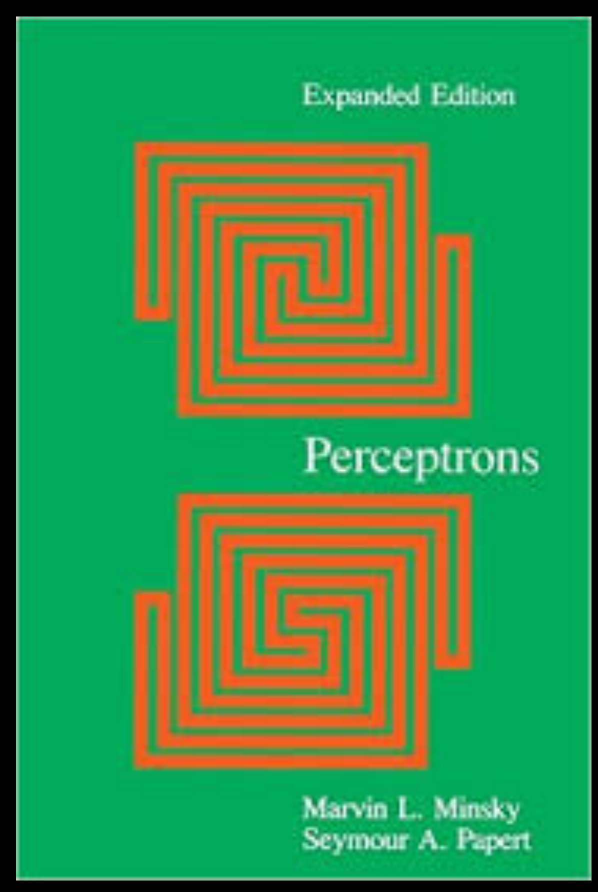


  - **learning rule (similar to the Delta rule):**

$$\Delta\theta = -(t_p - o_p) = -\delta_p \quad \textbf{(threshold)}$$

$$\Delta w_i = (t_p - o_p)i_{pi} = \delta_p i_p \quad \textbf{(weights)}$$

- **Minsky & Papert** *(1969)*
  - **Perceptrons can’t solve the XOR problem**
  - ∴ **not computationally general**



The Problem

The Problem

Actually, two problems:

1) Additive

- **can't solve problems that are not *"linearly separable"*** *(e.g., XOR)*…

The Problem

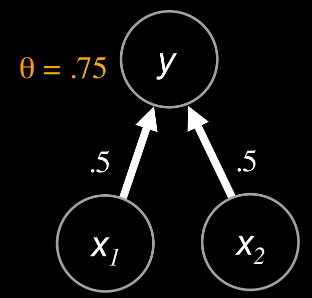
Actually, two problems:

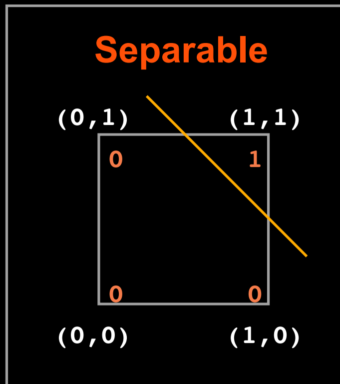
1) Additive

- **can't solve problems that are not *"linearly separable"*** *(e.g., XOR)*…

AND

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





The Problem

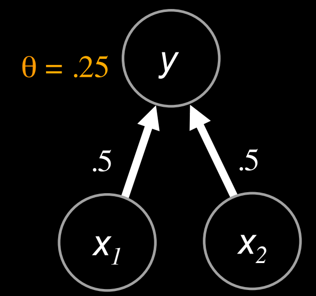
Actually, two problems:

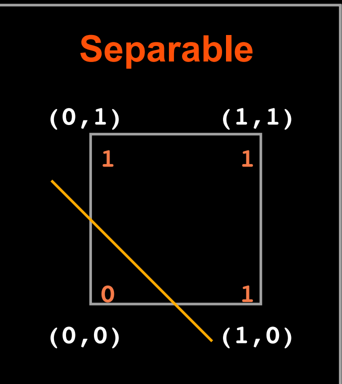
1) Additive

- **can't solve problems that are not *"linearly separable"*** *(e.g., XOR)*…

OR

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





The Problem

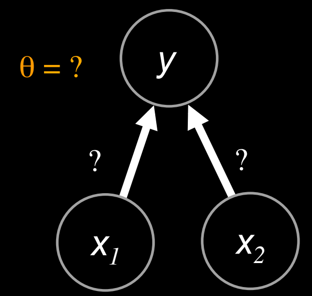
Actually, two problems:

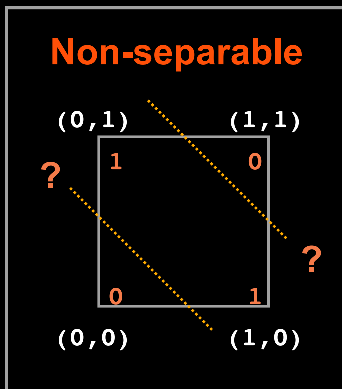
1) Additive

- **can't solve problems that are not *"linearly separable"* *(e.g., XOR)*…**

X-OR

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





The Problem

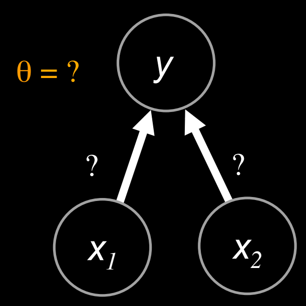
Actually, two problems:

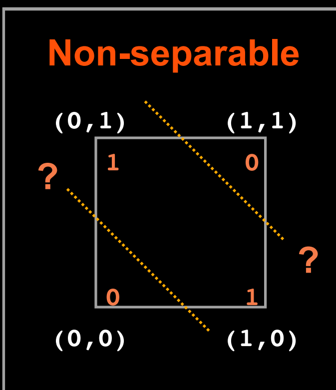
1) Additive

– **can't solve problems that are not** ***"linearly separable"*** *(e.g., XOR)*…

X-OR

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





Example:

no food ⇒ don't like

The Problem

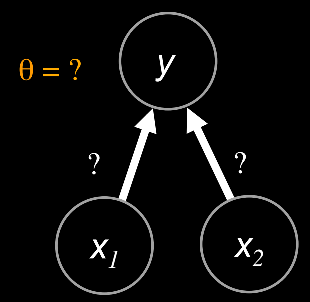
Actually, two problems:

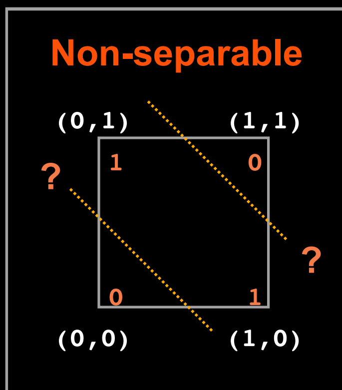
1) Additive

- **can't solve problems that are not** ***"linearly separable"*** *(e.g., XOR)*…

X-OR

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





Example:

no food	⇒	don't like
pickles	⇒	like

The Problem

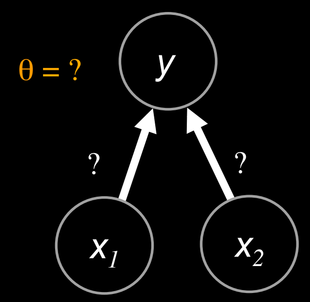
Actually, two problems:

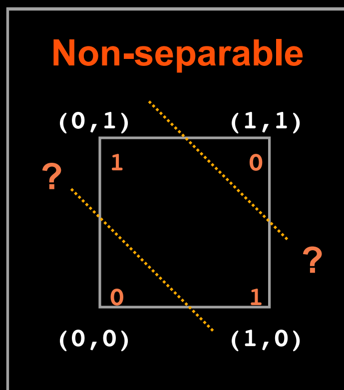
1) Additive

- can't solve problems that are not *"linearly separable"* (*e.g., XOR*)…

X-OR

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





Example:

no food	⇒	don't like
pickles	⇒	like
ice cream	⇒	like

The Problem

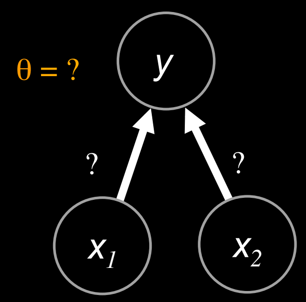
Actually, two problems:

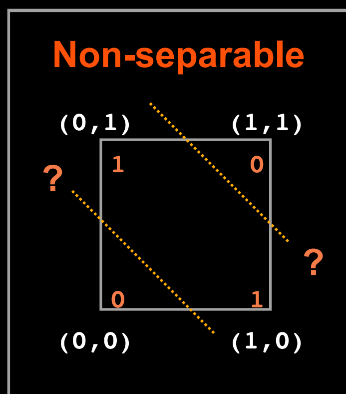
1) Additive

– can't solve problems that are not *"linearly separable"* (*e.g., XOR*)…

X-OR

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





Example:

no food ⇒ don't like
pickles ⇒ like
ice cream ⇒ like
pickles + ice cream ⇒ don't like

The Problem

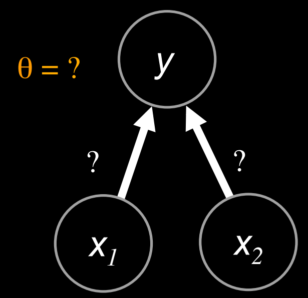
Actually, two problems:

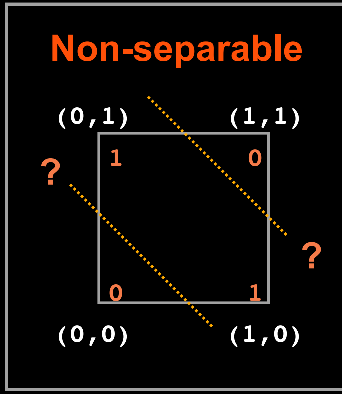
1) Additive

– **can't solve problems that are not *"linearly separable"*** *(e.g., XOR)*…

X-OR

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





Example:

no food ⇒ don't like
pickles ⇒ like
ice cream ⇒ like
pickles + ice cream ⇒ don't like *(unless its Bent Spoon)*

The Problem

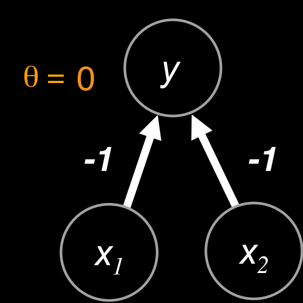
Actually, two problems:

1) **Multiplicative would solve the problem:**

(if inputs can range from *negative* to positive)

X-OR

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



Separable

(-1,1) (1,1)

(-1,-1) (1,-1)

The Problem

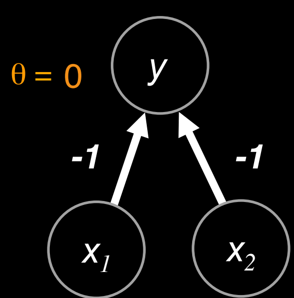
Actually, two problems:

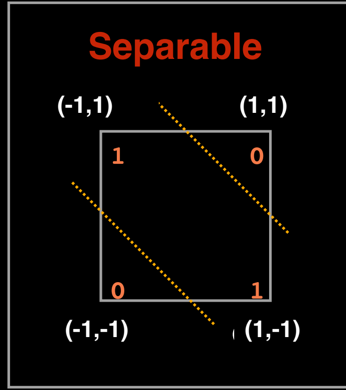
1) Multiplicative would solve the problem:

(if inputs can range from *negative* to positive)

X-OR

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

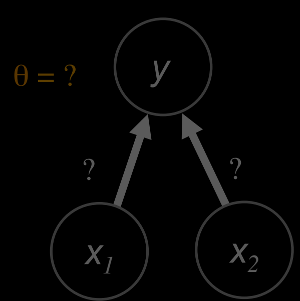


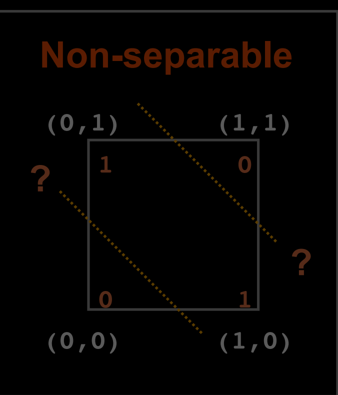


The Problem

X-OR

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





The Problem

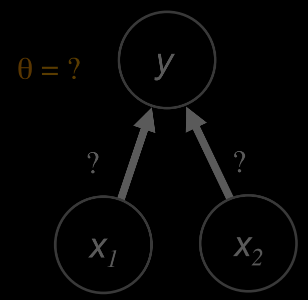
Actually, two problems:

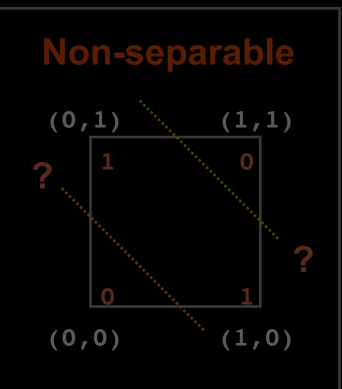
1) Additive

– can't solve problems that are not *"linearly separable"*…

X-OR

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





2) Single layered

– *"hidden units"* solve the problem…

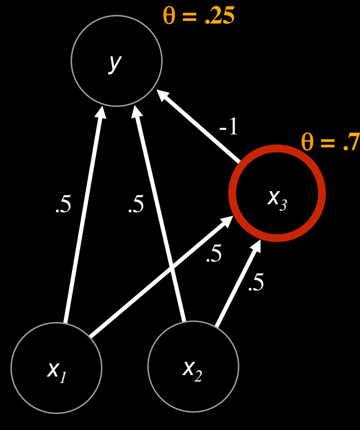
The Problem

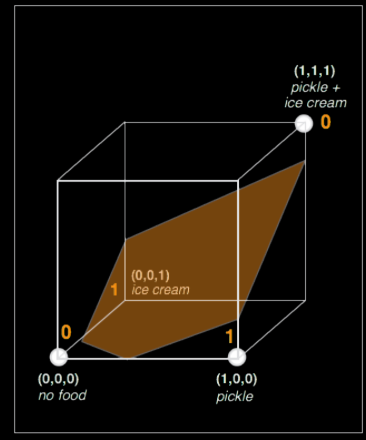
Actually, two problems:

1) Additive

– can't solve problems that are not *"linearly separable"*…

x_1	x_2	x_3	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

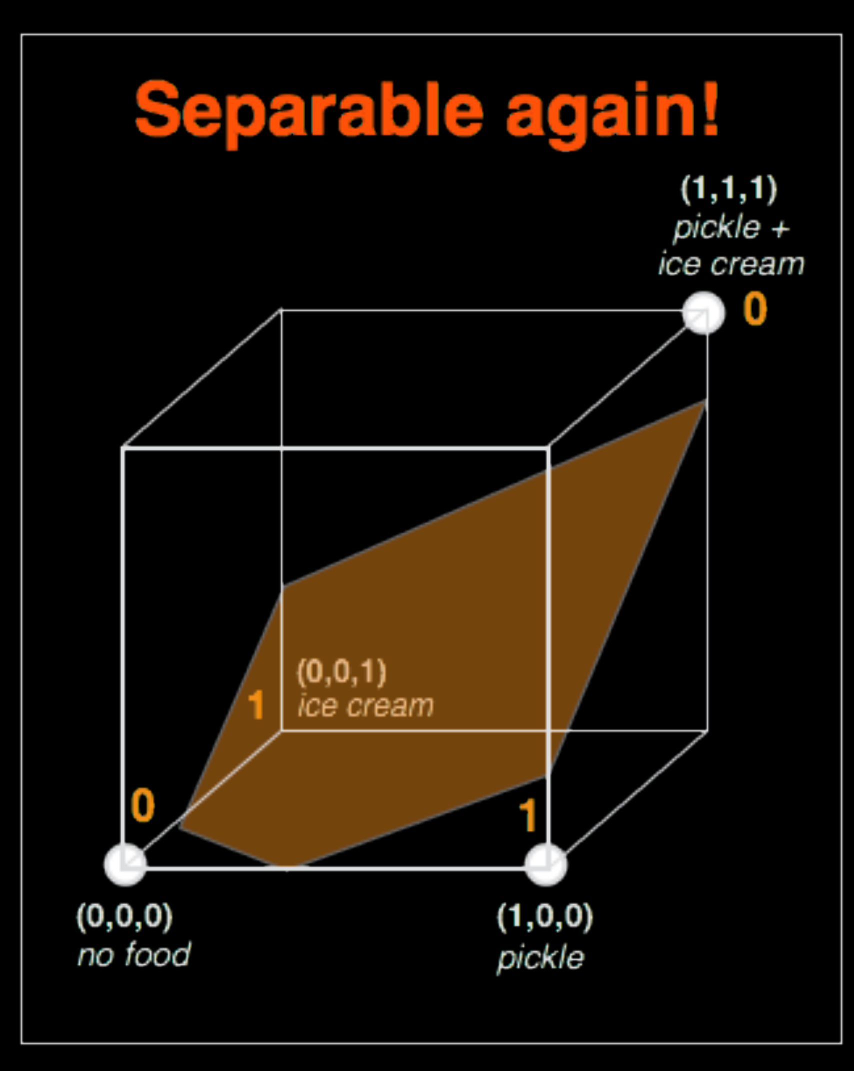




2) Multi-layered

– *"hidden units"* solve the problem

Add a 3rd Dimension



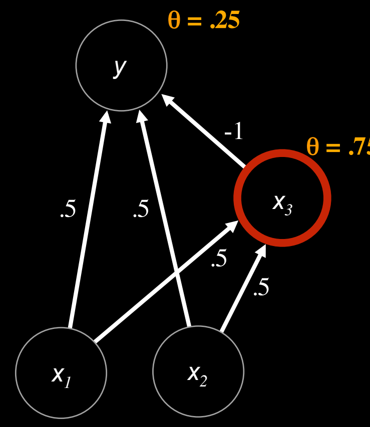
The Problem

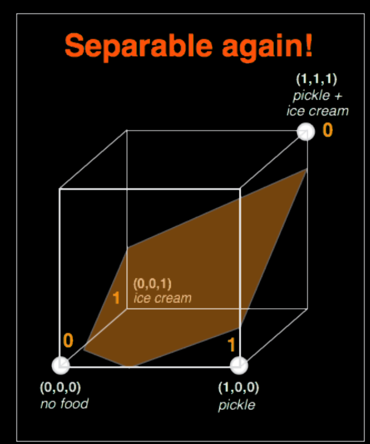
Actually, two problems:

1) Additive

– can't solve problems that are not *"linearly separable"*…

x_1	x_2	x_3	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





2) Multi-layered

– *"hidden units"* solve the problem *(Minksy & Papert knew that…)*

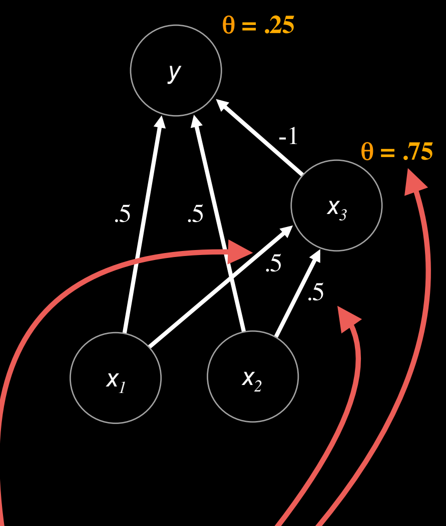
The Problem

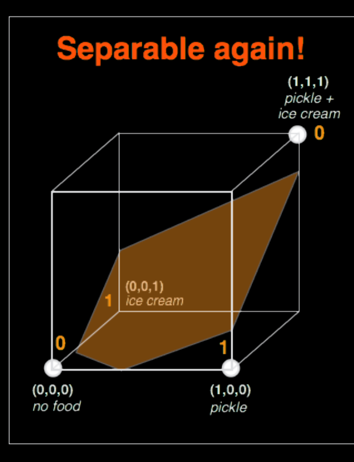
Actually, two problems:

1) Additive

– can't solve problems that are not *"linearly separable"*…

x_1	x_2	x_3	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





2) Multi-layered

– *"hidden units"* solve the problem *(Minksy & Papert knew that)…*

– ***but how do you train hidden units? where does the error come from?***

The Solution

The Solution

- **Backpropagation Algorithm** *(Generalized Delta Rule)*

  Generalizes the LMS procedure (Delta Rule) to apply to:

  - **Non-linear units**
  - **Hidden units**

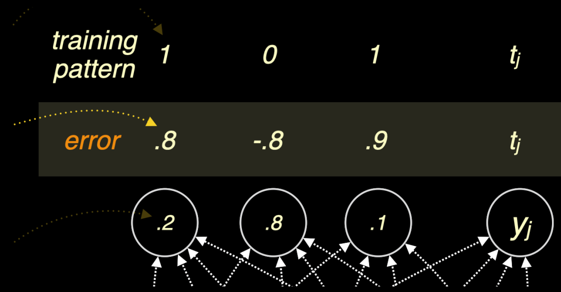
The Solution

- **Backpropagation Algorithm** *(Generalized Delta Rule)*

  Generalizes the LMS procedure (Delta Rule) to apply to:
  - **Non-linear units**
  - **Hidden units**

- **Start with Delta Rule:**

  $\Delta w_i j = \epsilon \delta_j x_i$



The Solution

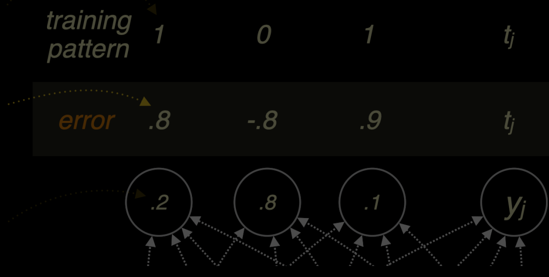
- **Backpropagation Algorithm** *(Generalized Delta Rule)*

  Generalizes the LMS procedure (Delta Rule) to apply to:

  - **Non-linear units**
  - **Hidden units**

- **Start with Delta Rule:**

  $\Delta w_i j = \epsilon \delta_j x_i$



- **For the output units:**

$$\delta_j = (t_j - y_j) \cdot f_j'(net_j) \quad \text{where} \quad f'_j(net_j) = \frac{d(x_j)}{d(net_j)}$$

The Solution

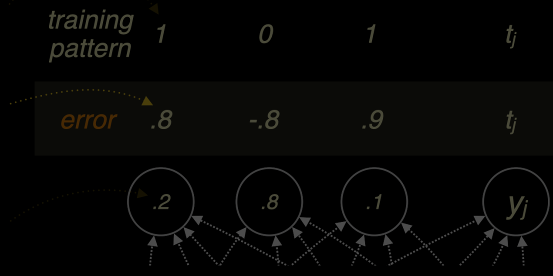
- **Backpropagation Algorithm** *(Generalized Delta Rule)*

  Generalizes the LMS procedure (Delta Rule) to apply to:

  - Non-linear units
  - Hidden units

- **Start with Delta Rule:**

  $\Delta w_i j = \epsilon \delta_j x_i$



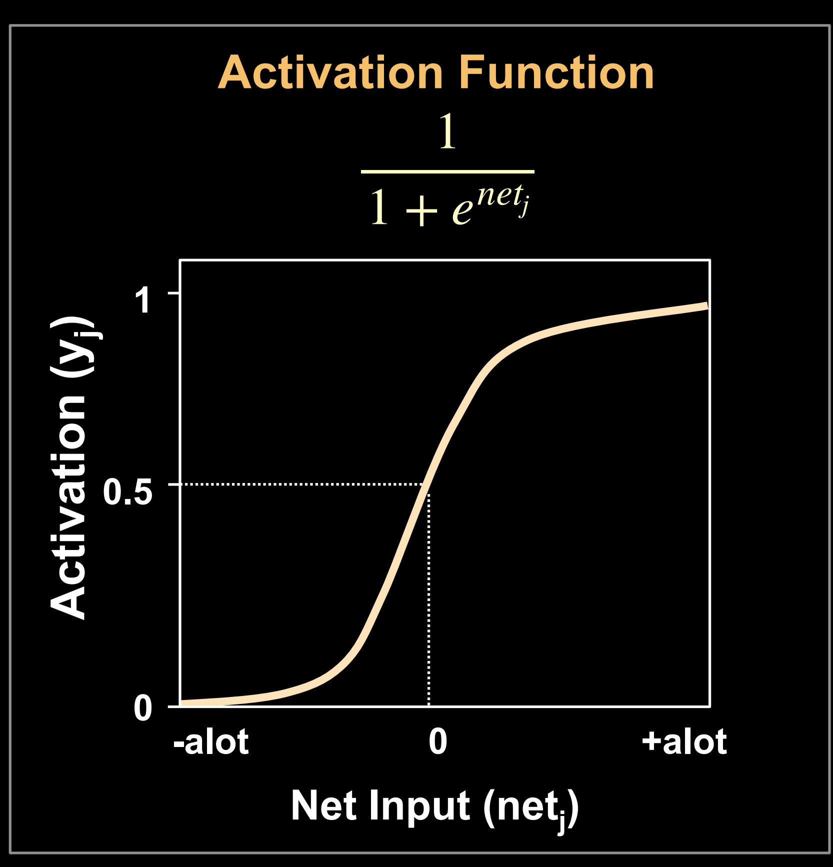
- **For the output units:**

$$\delta_j = (t_j - y_j) \cdot f_j'(net_j) \quad \text{where} \quad f'_j(net_j) = \frac{d(x_j)}{d(net_j)}$$

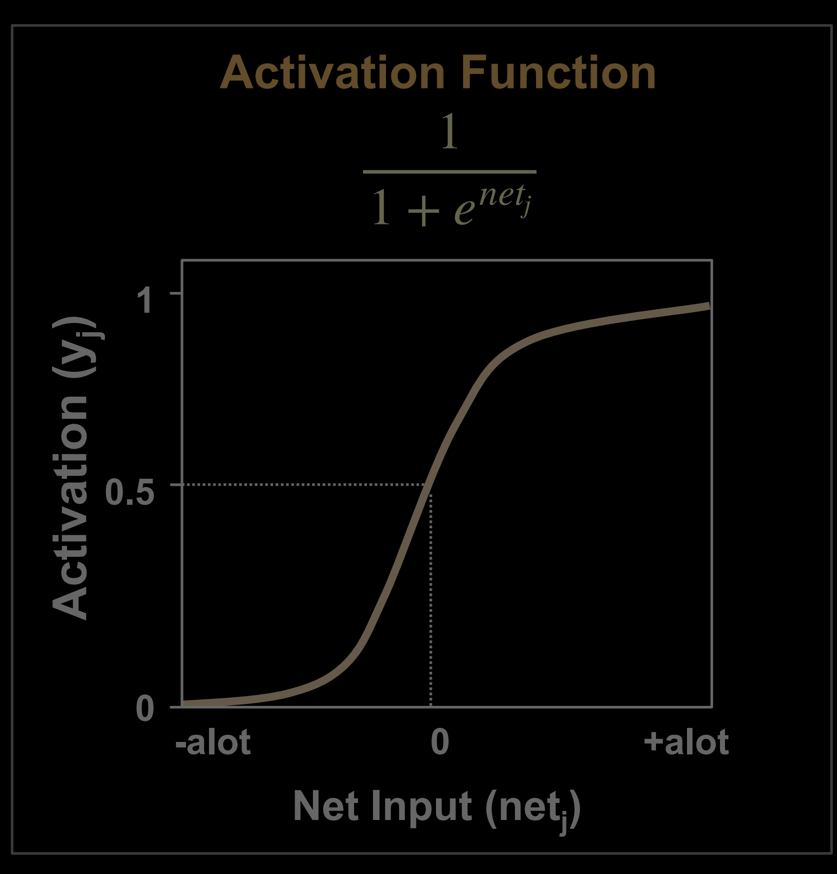
Note:

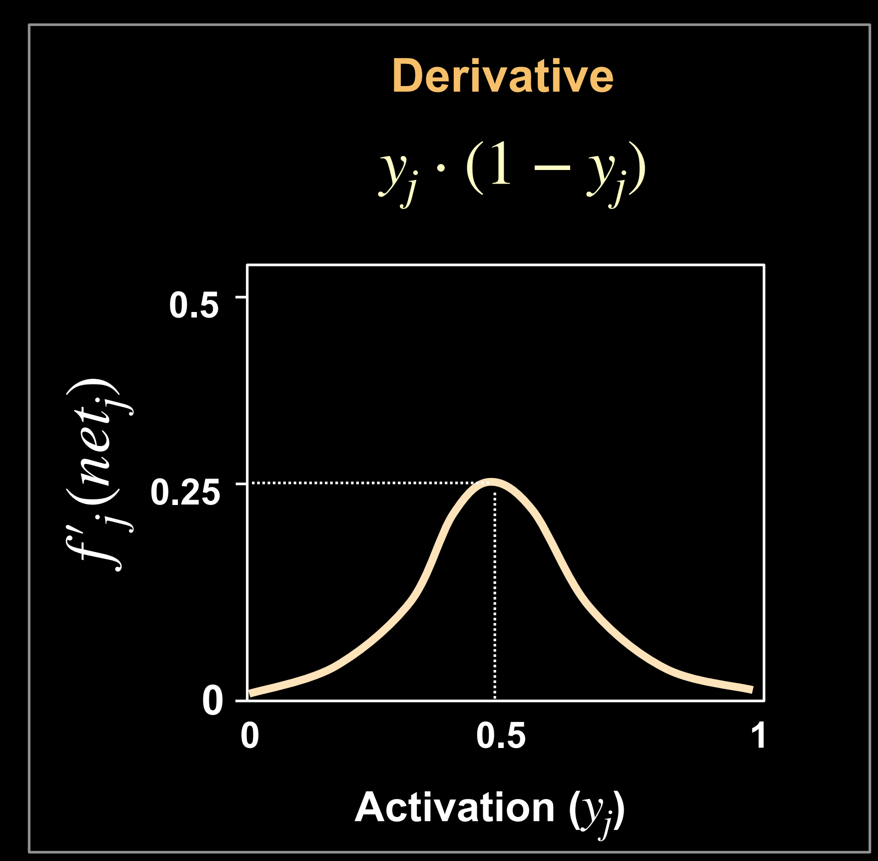
- this is similar to the Delta Rule, except that the error term is now scaled by the ***derivative*** of the ***activation function***
- this requires that the activation function be ***differentiable***, and therefore ***continuous***

Logistic Activation Function

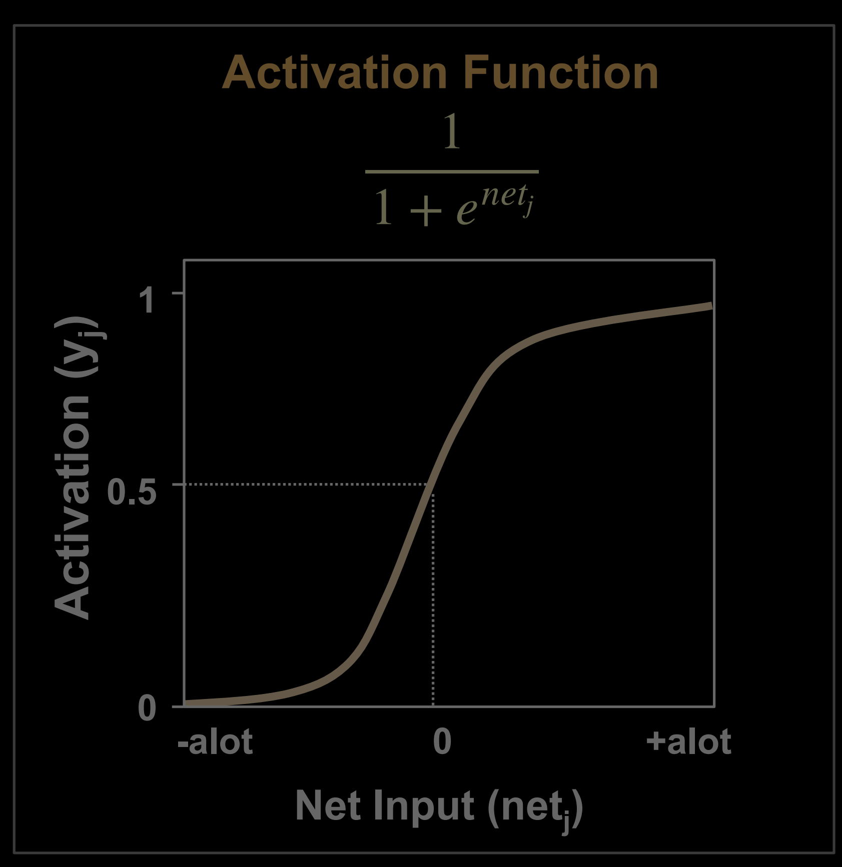


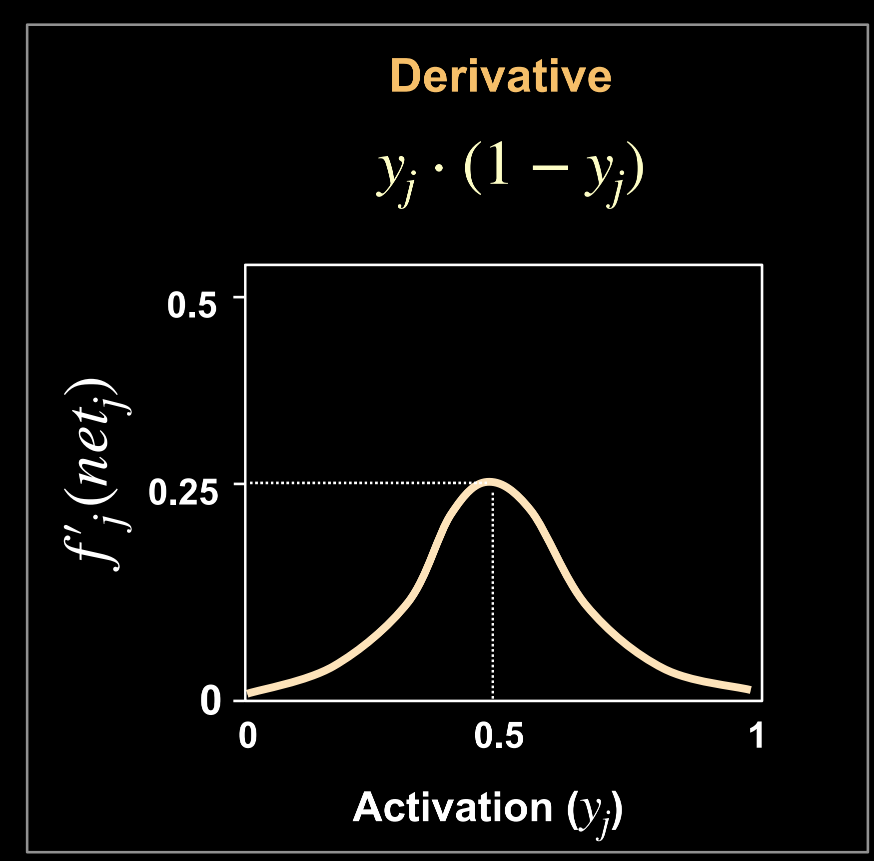
Logistic Activation Function





Logistic Activation Function



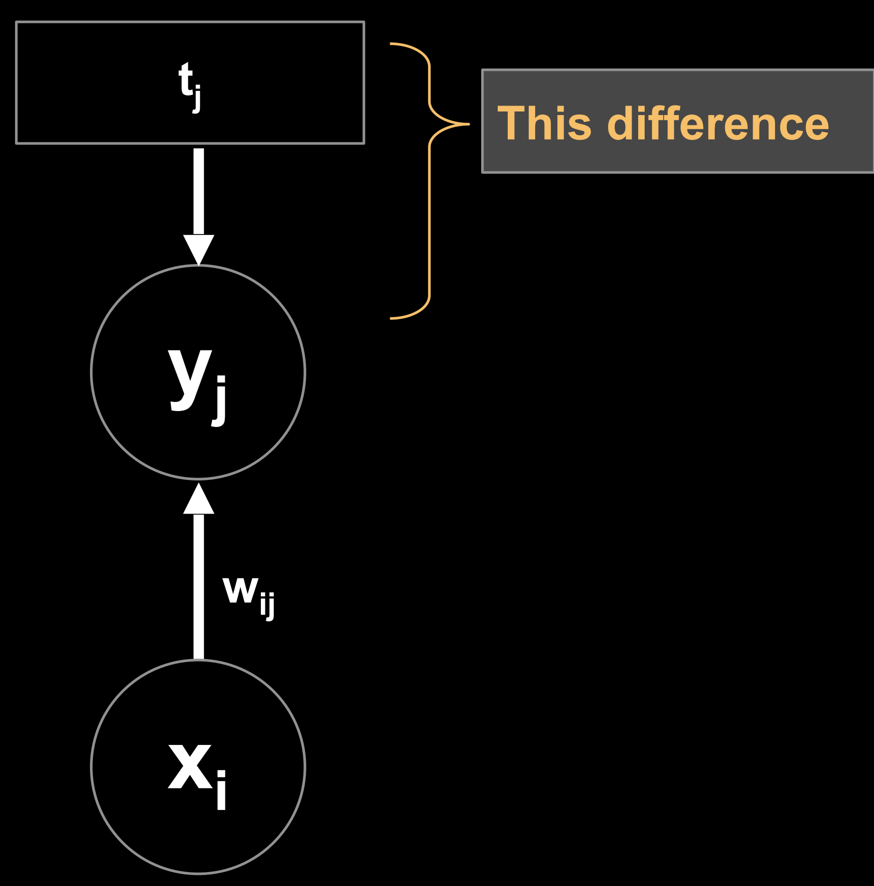


Note:

- Derivative (slope) is maximal when activation is 0.5 (i.e., unit is uncommited)

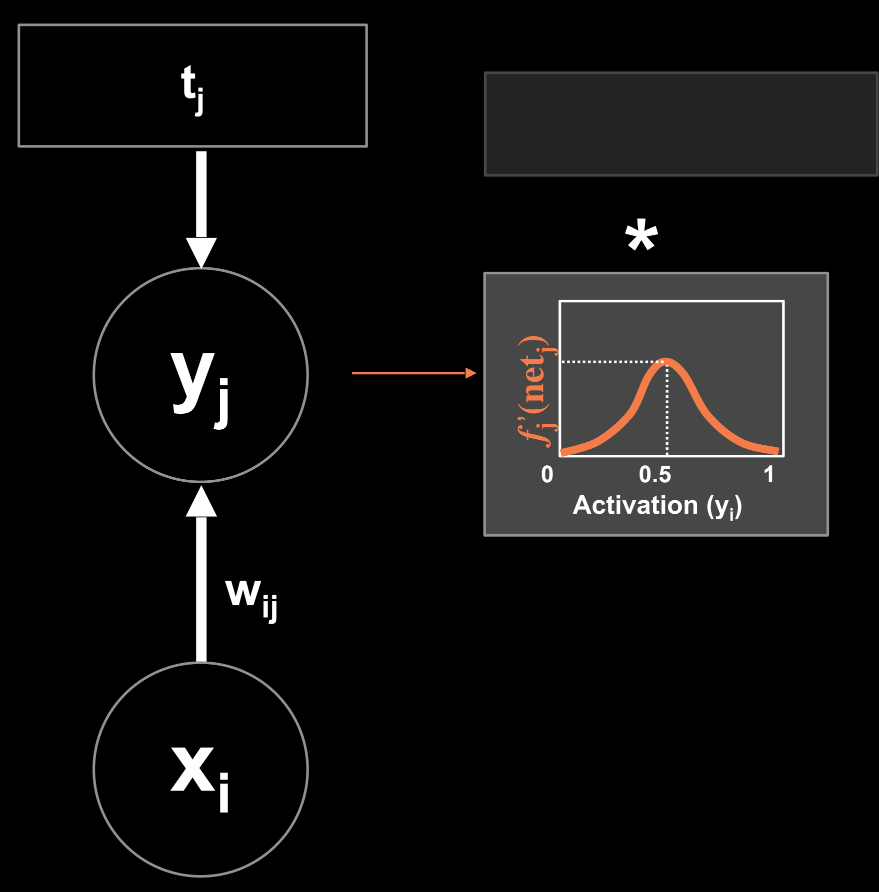
Effect of Non-Linearity

$$\Delta W_{ij} = (t_j - y_j)$$



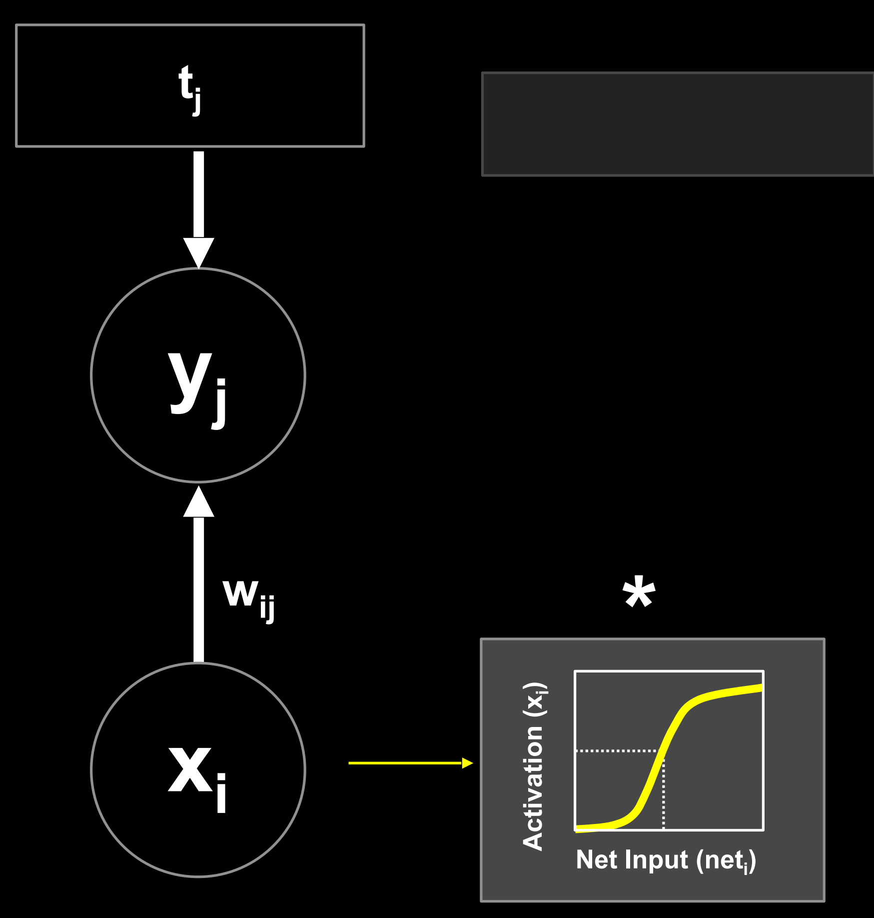
Effect of Non-Linearity

$$\Delta W_{ij} = (t_j - y_j) \cdot f'_j(net_j)$$



Effect of Non-Linearity

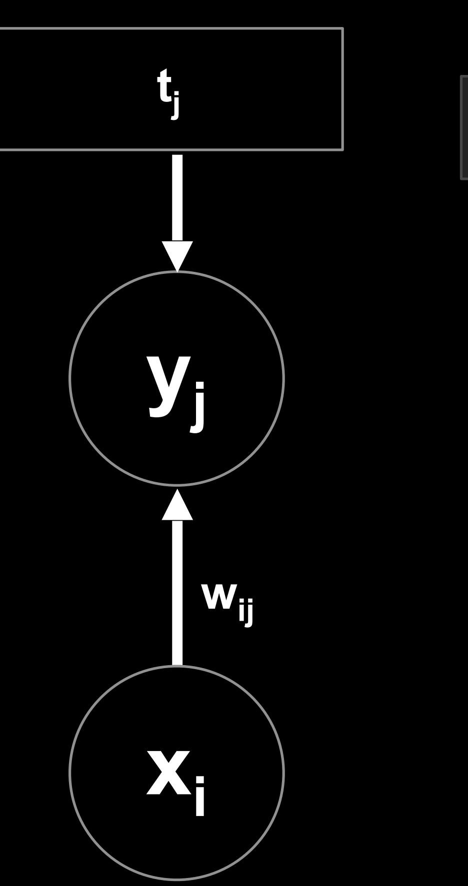
$$\Delta W_{ij} = (t_j - y_j) \cdot f'_j(net_j) \cdot x_j$$



Effect of Non-Linearity

$$\Delta W_{ij} = (t_j - y_j) \cdot f'_j(net_j) \cdot x_j$$

- **Note:**
  - **Biggest weights changes are made to "uncommitted" receiving units *(y)* (i.e., for which activity ~ 0.5)**
  - **Weight change is scaled by activity of sending unit *(x)***



Backpropagation Algorithm

- **Backpropagation Learning Algorithm**

  Generalizes the LMS procedure to apply to:
  - **Hidden units**
  - **Non-linear units**

- **Start with LMS (Delta) rule:**

  $$\Delta w_{ij} = \epsilon \delta_j x_j$$

- **For the output units:**

  $$\delta_j = (t_j - y_j) \cdot f'_j(net_j) \quad \textit{where} \quad f'_j(net_j) = \frac{d(x_j)}{d(net_j)}$$

- **For the hidden units:**

  $$\delta_k = (\Sigma_j \delta_j w_{jk}) \cdot f'_k(net_k) \quad \textbf{\textit{where}} \quad f'_k(net_k) = f'_j(net_j) = \frac{d(x_k)}{d(net_k)}$$

Backpropagation Algorithm

- **Backpropagation Learning Algorithm**

  Generalizes the LMS procedure to apply to:
  - **Hidden units**
  - **Non-linear units**

- **Start with LMS (Delta) rule:**

  $$\Delta w_{ij} = \epsilon \delta_j x_j$$

- **For the output units:**

  $$\delta_j = (t_j - y_j) \cdot f'_j(net_j) \quad \textit{where} \quad f'_j(net_j) = \frac{d(x_j)}{d(net_j)}$$

- **For the hidden units:**

  $$\delta_k = (\Sigma_j \delta_j w_{jk}) \cdot f'_k(net_k) \quad \textbf{\textit{where}} \quad f'_k(net_k) = f'_j(net_j) = \frac{d(x_k)}{d(net_k)}$$

KEY OBSERVATION:

- **The "error" δ_k for hidden unit *k* is the sum over the *errors* (deltas δj) *of the units$_j$ to which it projects*, *weighted by its connections w_{jk} to them*.**

Backpropagation Algorithm

- **Backpropagation Learning Algorithm**

  Generalizes the LMS procedure to apply to:
  - **Hidden units**
  - **Non-linear units**

- **Start with LMS (Delta) rule:**

$$\Delta w_{ij} = \epsilon \delta_j x_j$$

- **For the output units:**

$$\delta_j = (t_j - y_j) \cdot f'_j(net_j) \quad \textit{where} \quad f'_j(net_j) = \frac{d(x_j)}{d(net_j)}$$

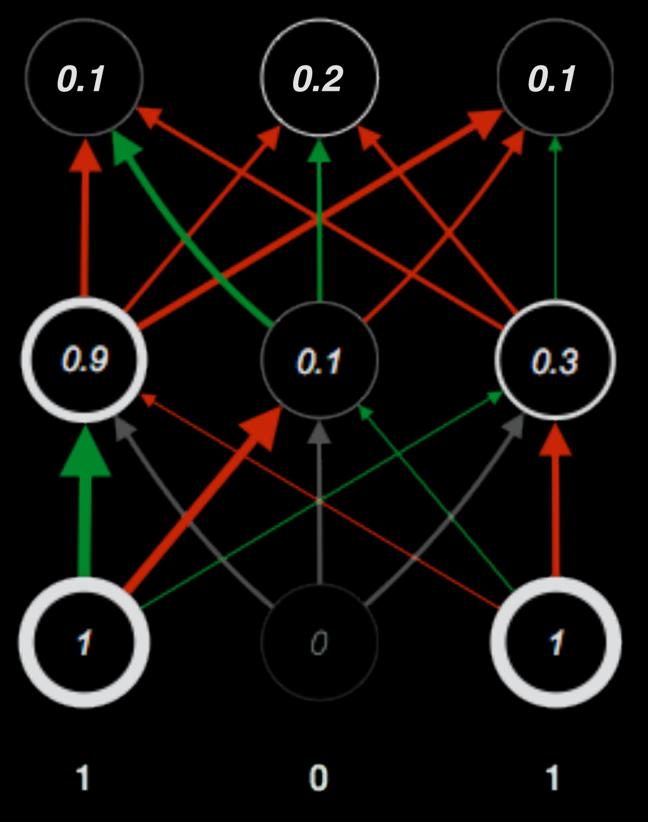
- **For the hidden units:**

$$\delta_k = (\Sigma_j \delta_j w_{jk}) \cdot f'_k(net_k) \quad \textbf{\textit{where}} \quad f'_k(net_k) = f'_j(net_j) = \frac{d(x_k)}{d(net_k)}$$

KEY OBSERVATION:

- **The “error” δ_k for hidden unit *k* is the sum over the *errors* (deltas δj) *of the units$_j$ to which it projects*, *weighted by its connections w_{jk} to them*.**
- **Minimizing this reduces the average *impact each unit k has on the error at the layer above it***

Error Calculation

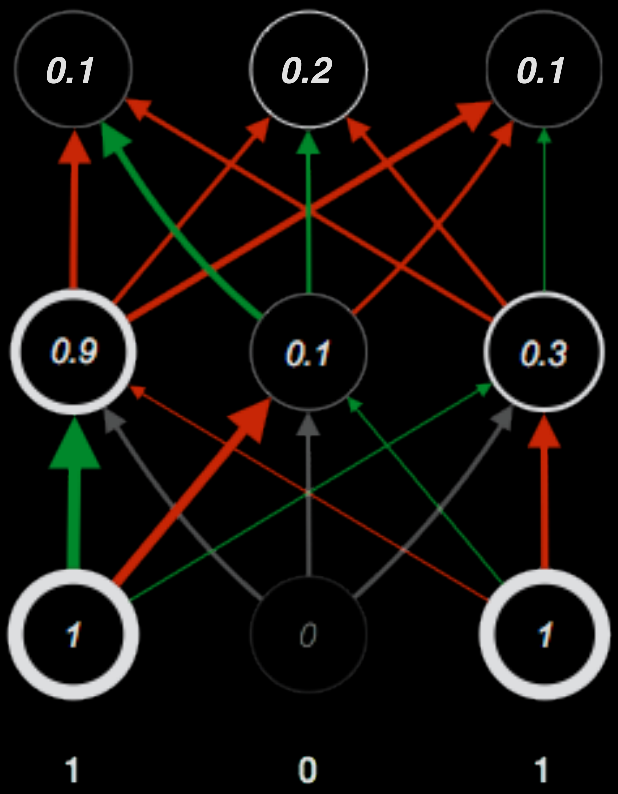


Excitatory weight

Inhibitory weight

Error Calculation

Target: 0 1 0



Input: 1 0 1

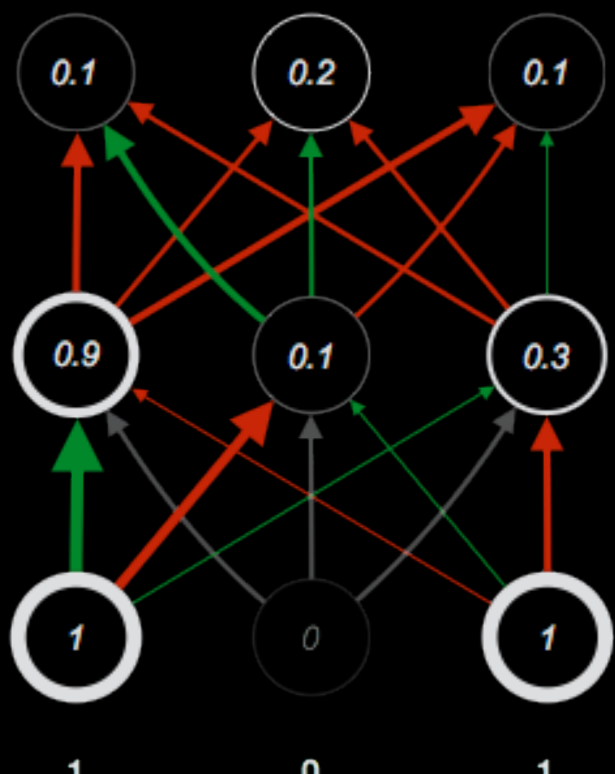
Excitatory weight

Inhibitory weight

Error Calculation

Target: 0 1 0

Output: *0.1* *0.2* *0.1*



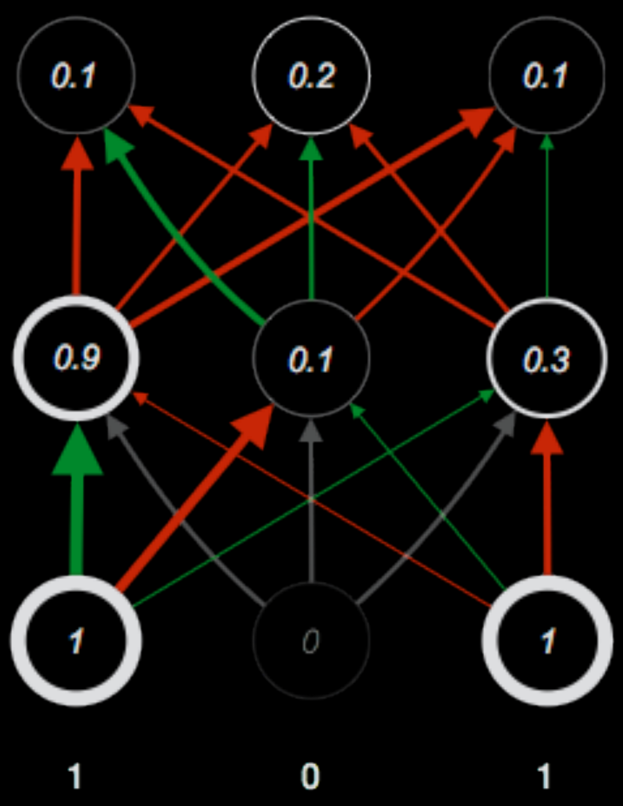
Excitatory weight

Inhibitory weight

Input: 1 0 1

Error Calculation

Target:	0	1	0
- Output:	*0.1*	*0.2*	*0.1*

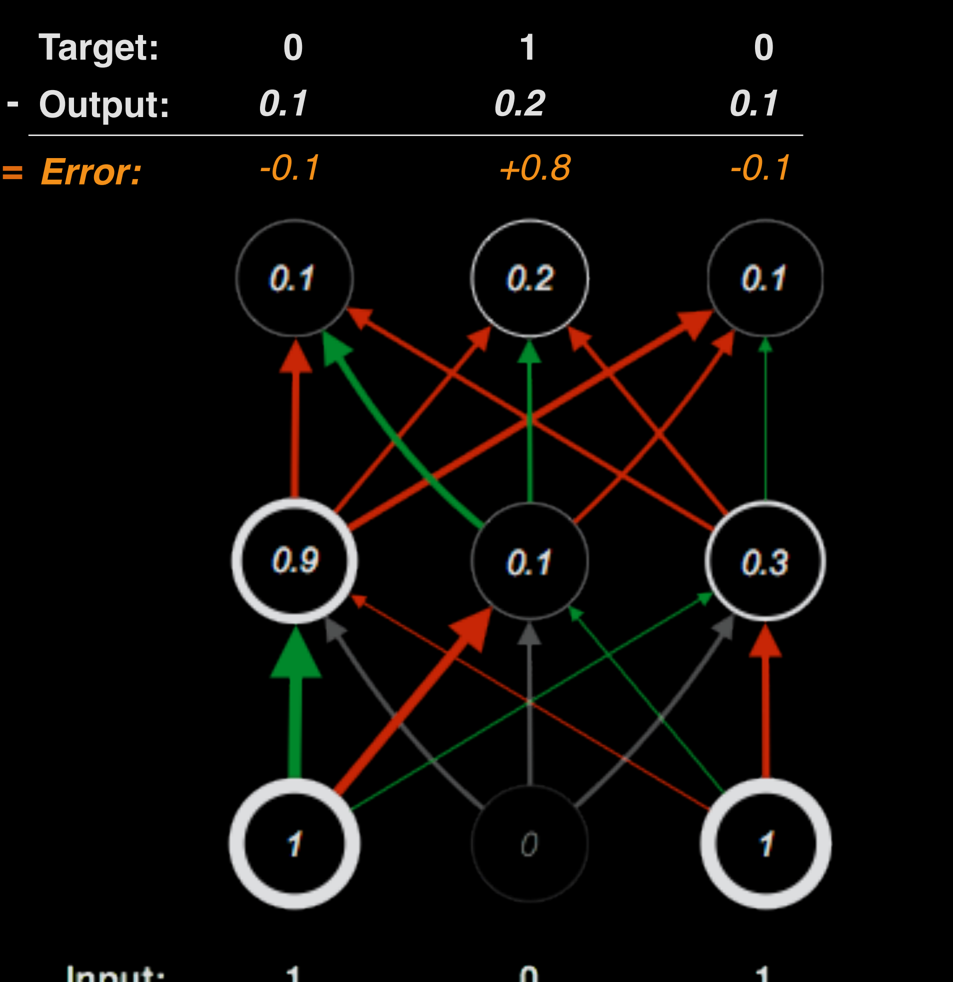


Input: 1 0 1

Excitatory weight

Inhibitory weight

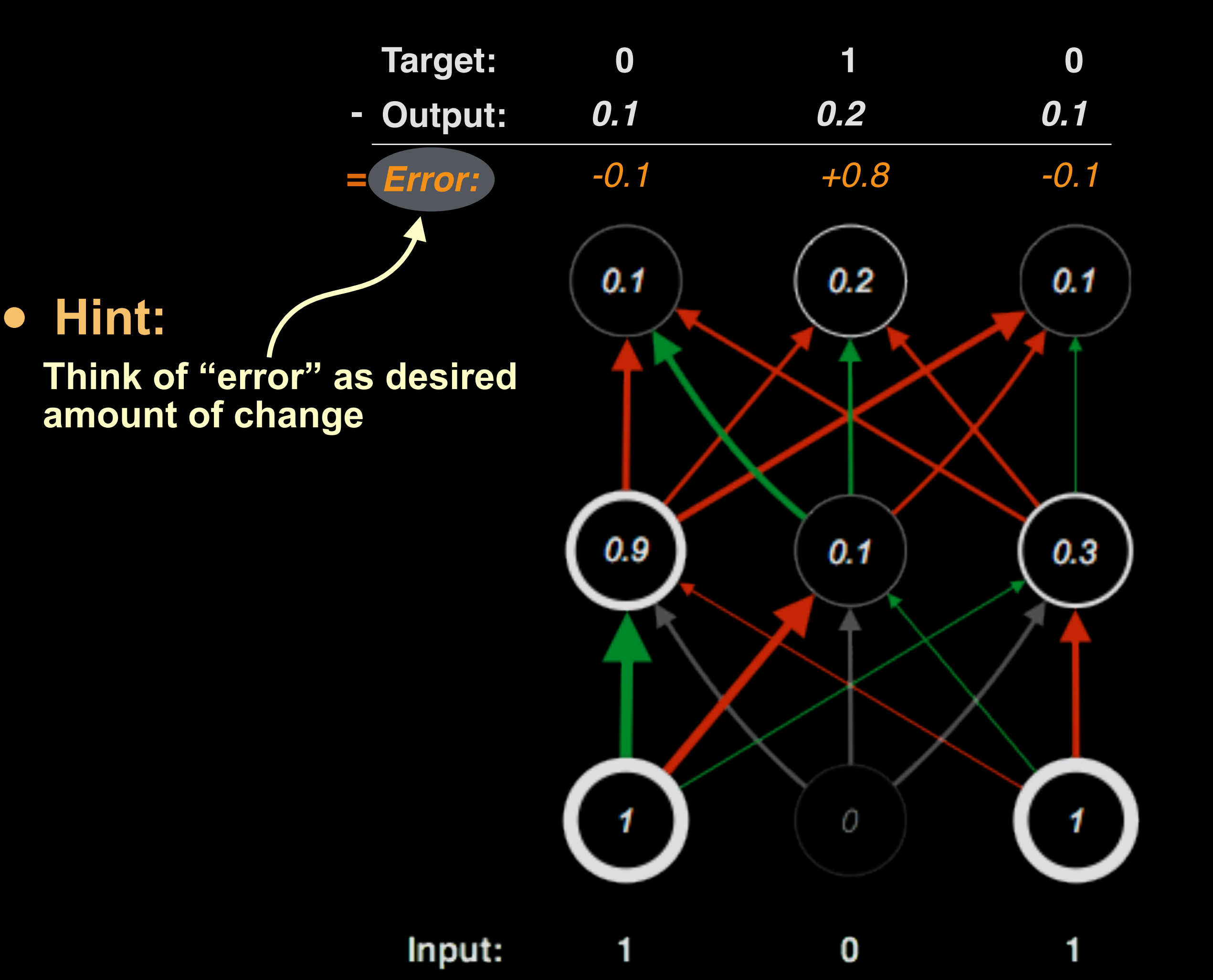
Error Calculation



Excitatory weight

Inhibitory weight

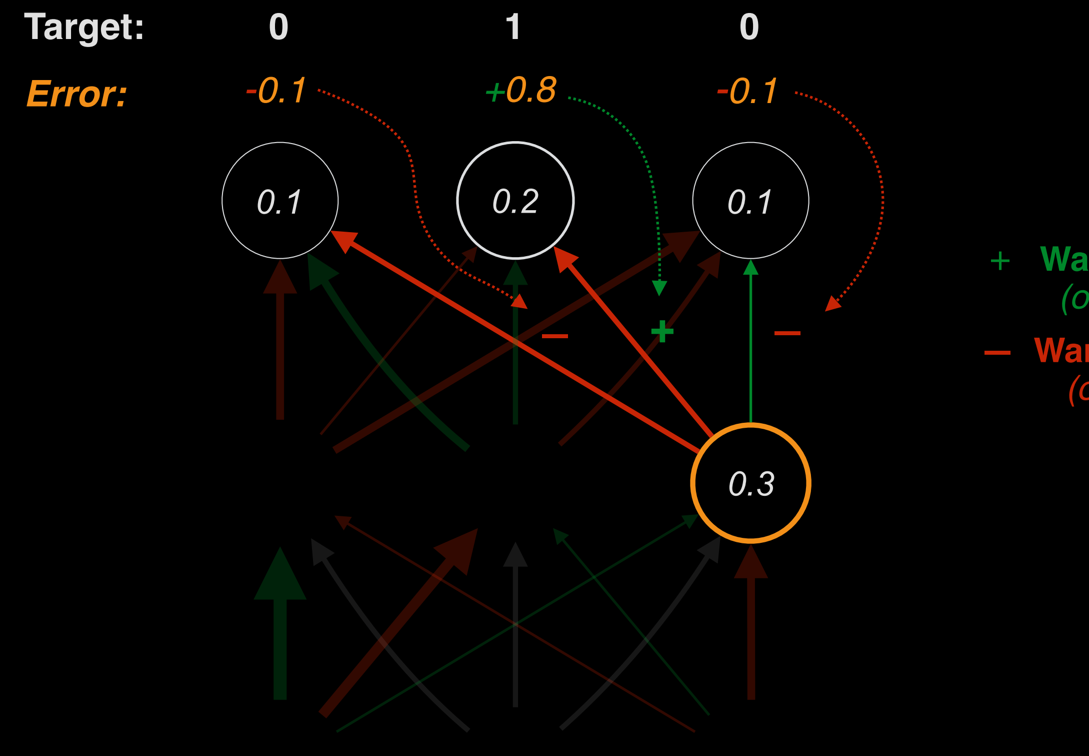
Error Calculation



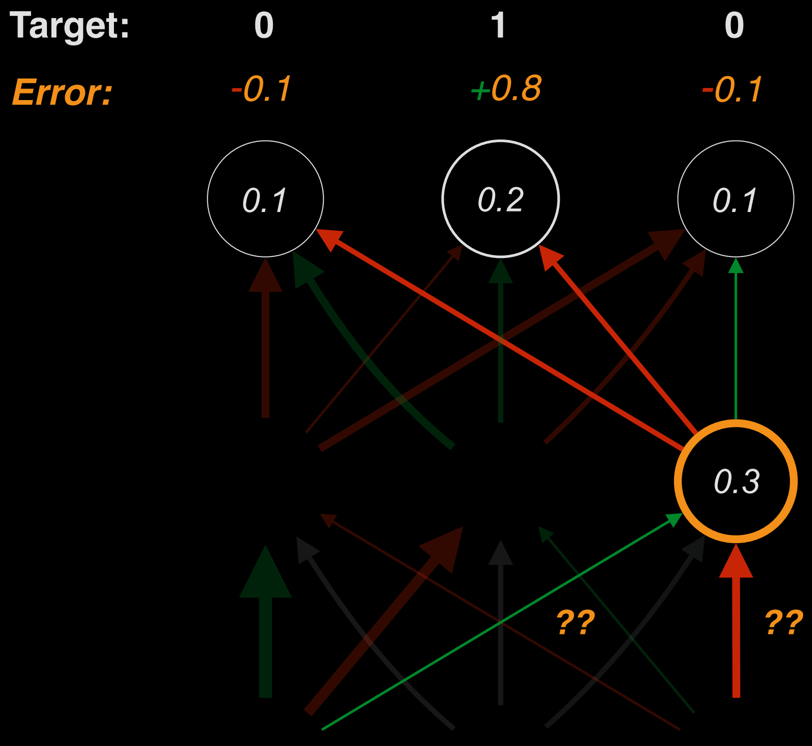
Excitatory weight

Inhibitory weight

Output Layer: Delta Rule

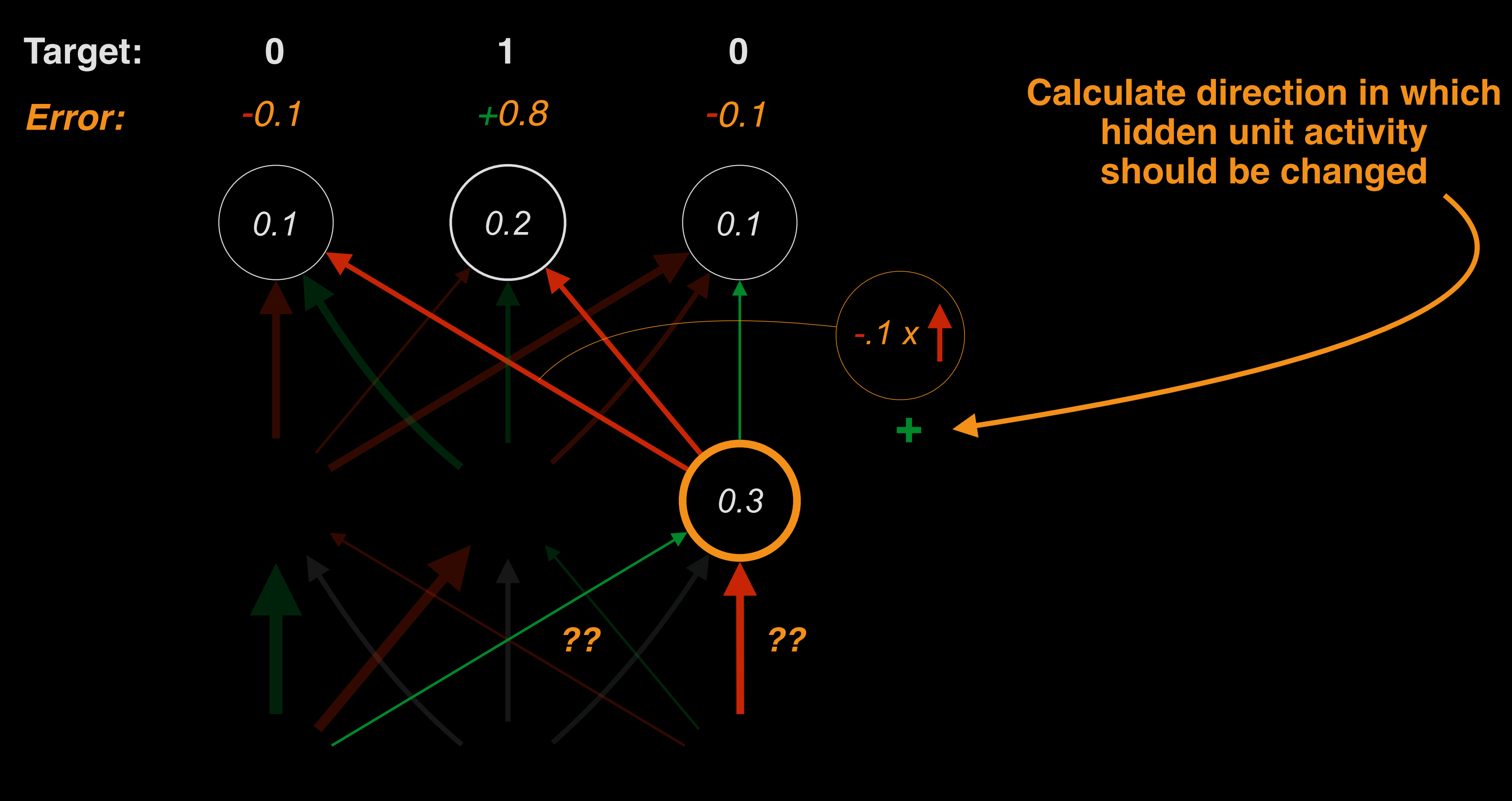


Hidden Layer: *Generalized* Delta Rule

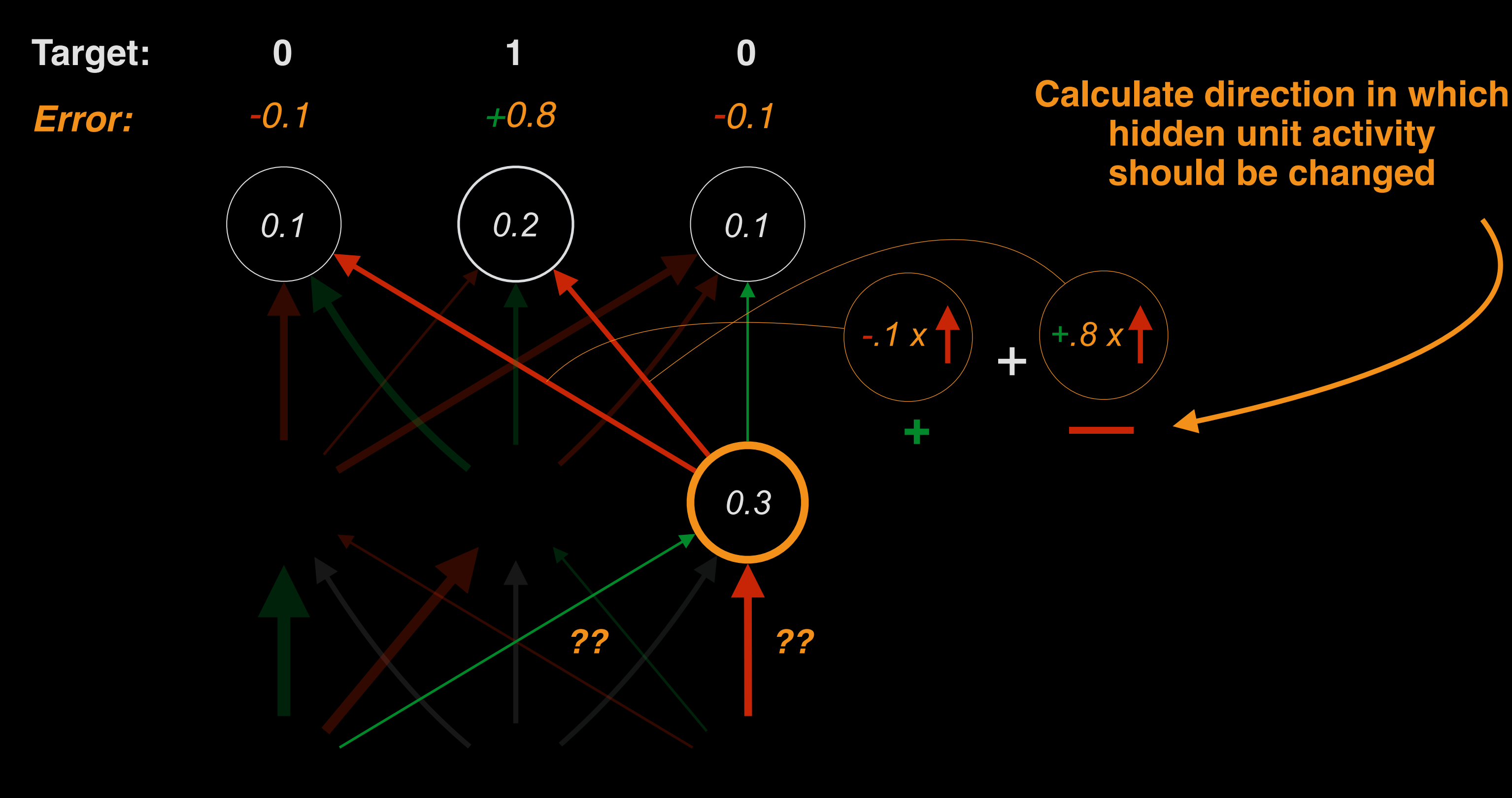


Calculate direction in which hidden unit activity should be changed

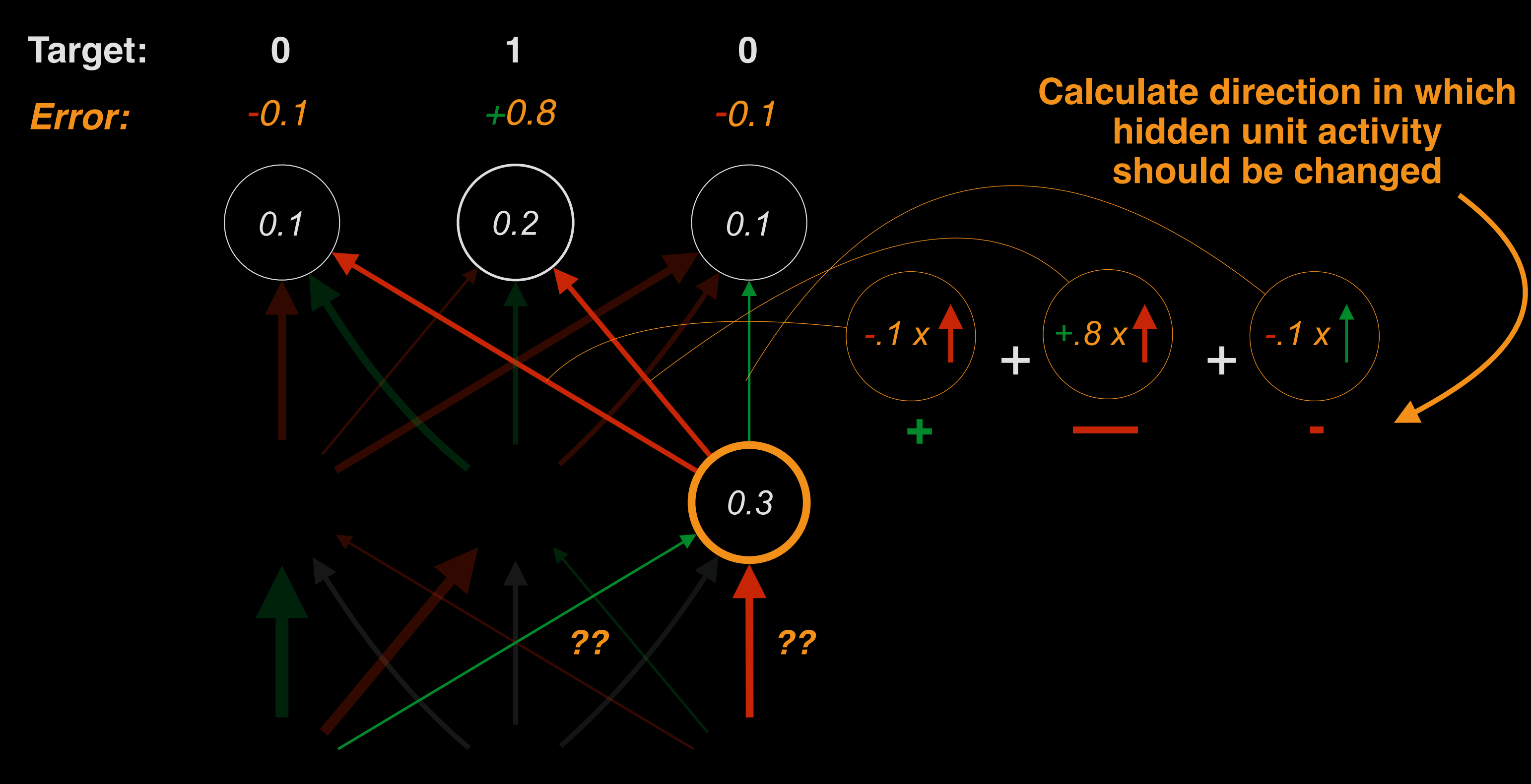
Hidden Layer: *Generalized* Delta Rule



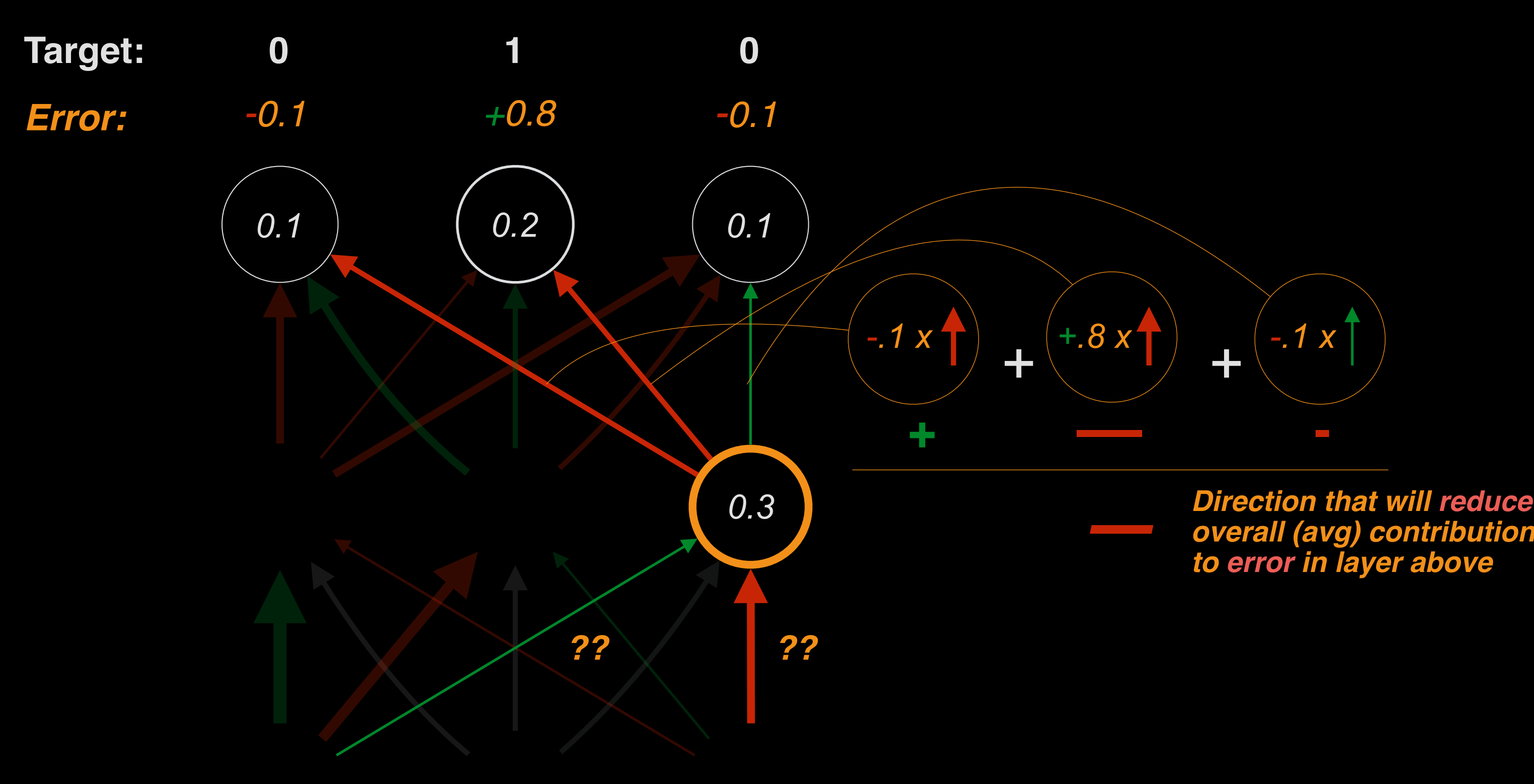
Hidden Layer: *Generalized* Delta Rule



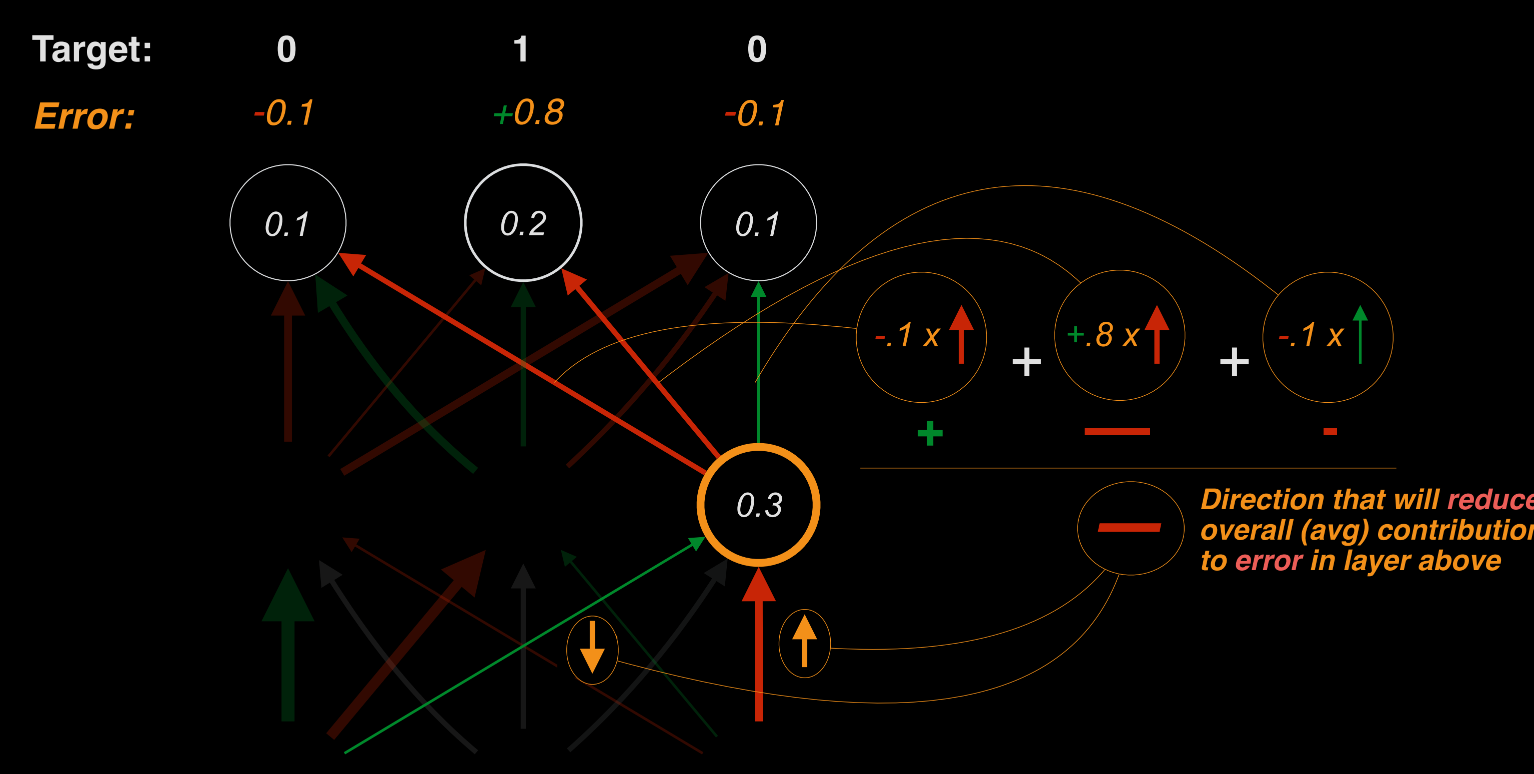
Hidden Layer: *Generalized* Delta Rule



Hidden Layer: *Generalized* Delta Rule

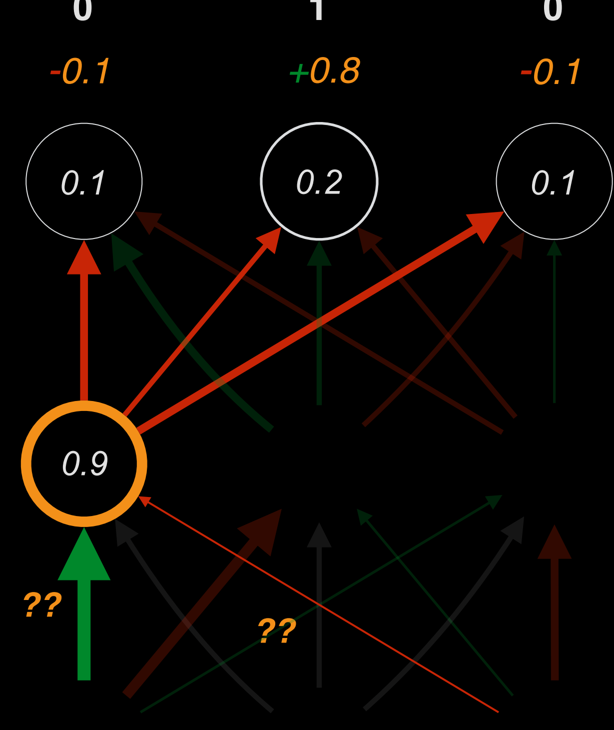


Hidden Layer: *Generalized* Delta Rule

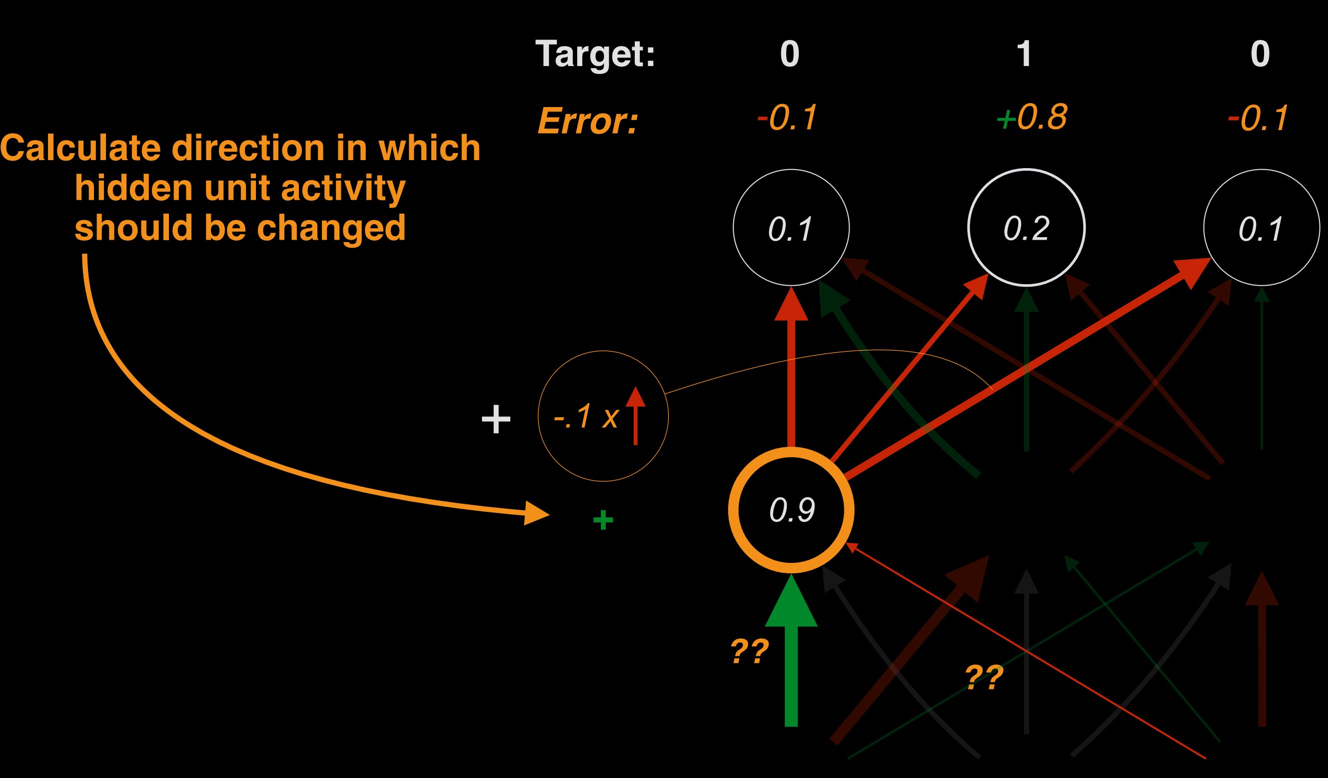


Hidden Layer: *Generalized* Delta Rule

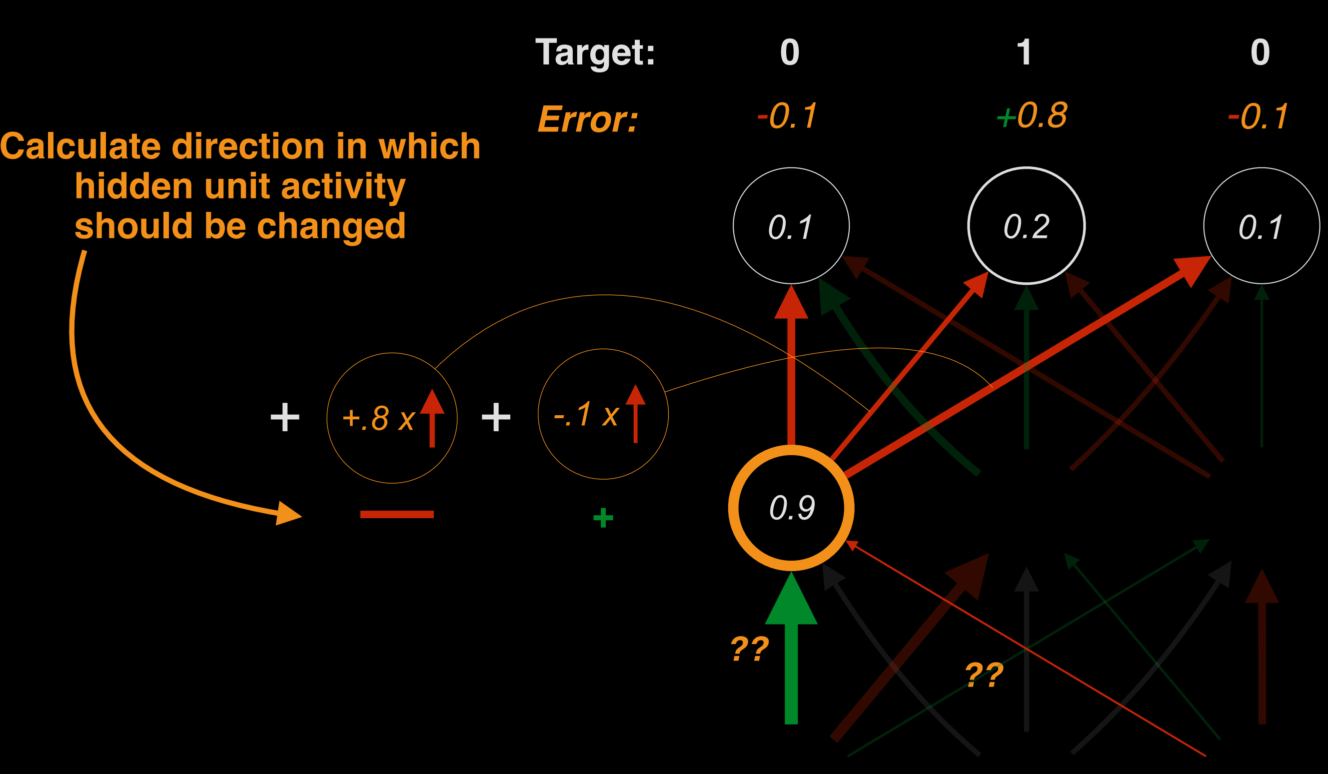
Calculate direction in which hidden unit activity should be changed



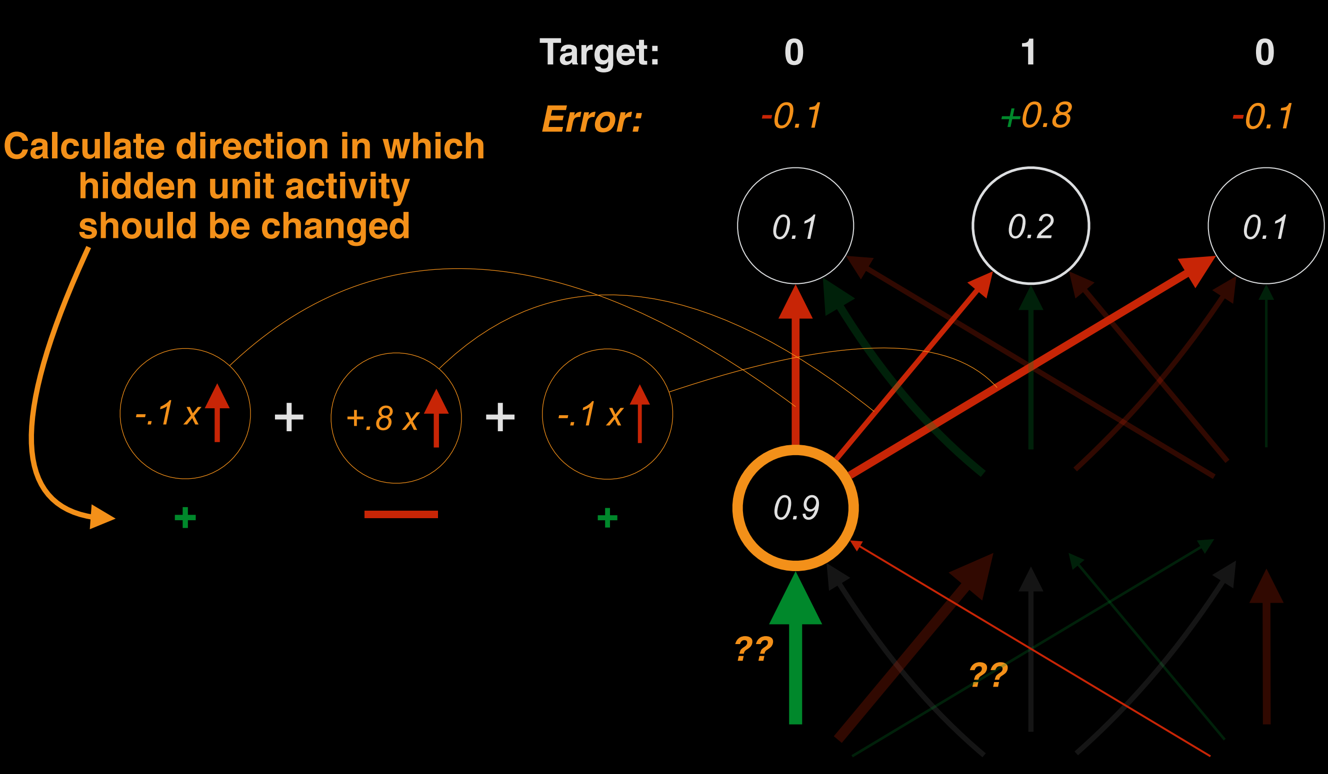
Hidden Layer: *Generalized* Delta Rule



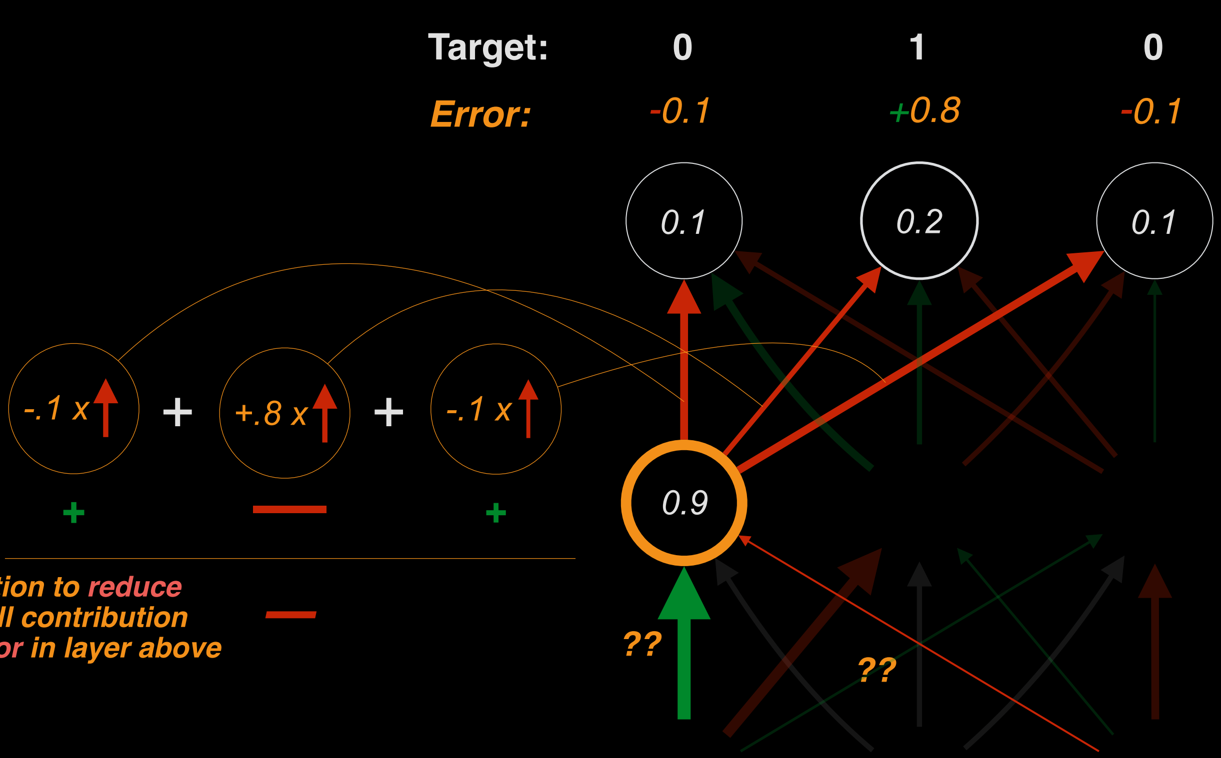
Hidden Layer: *Generalized* Delta Rule



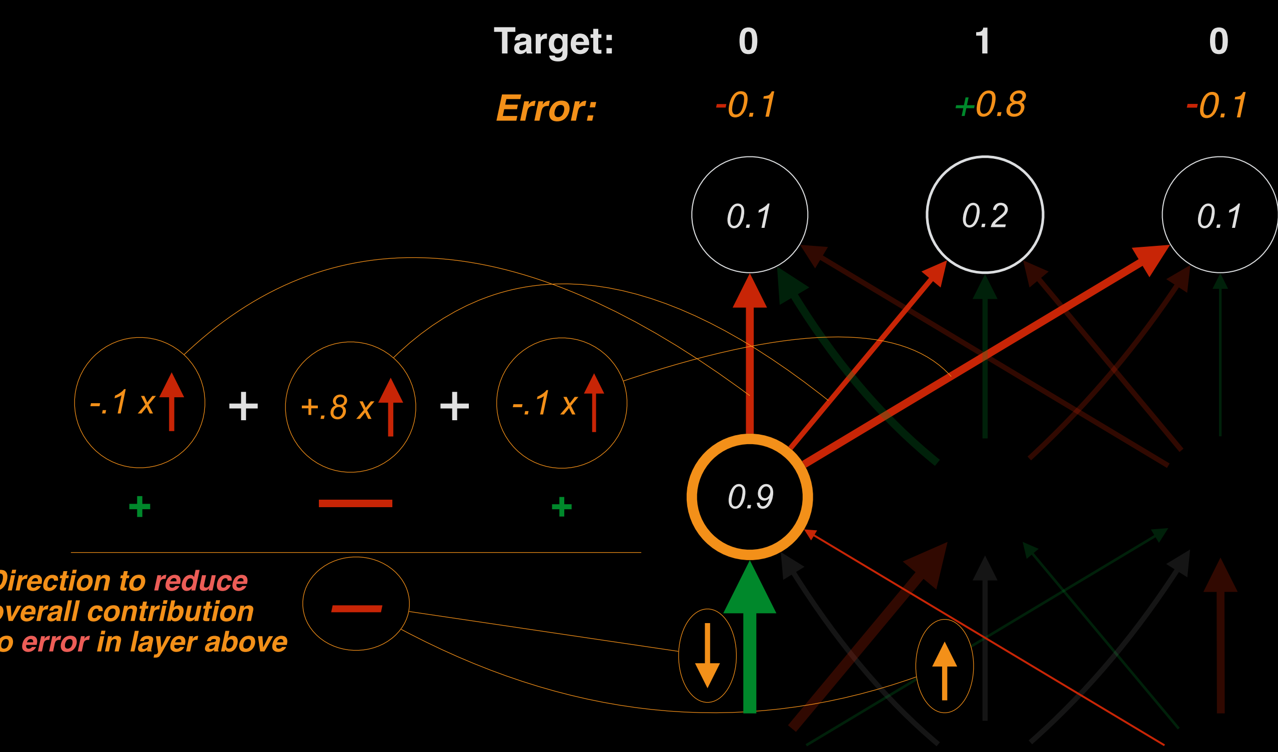
Hidden Layer: *Generalized* Delta Rule



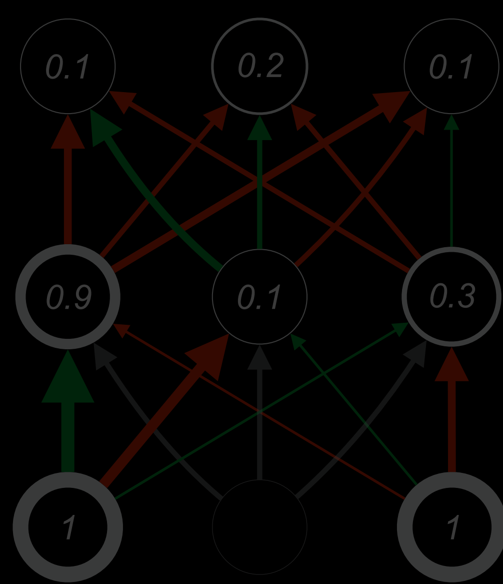
Hidden Layer: *Generalized* Delta Rule



Hidden Layer: *Generalized* Delta Rule

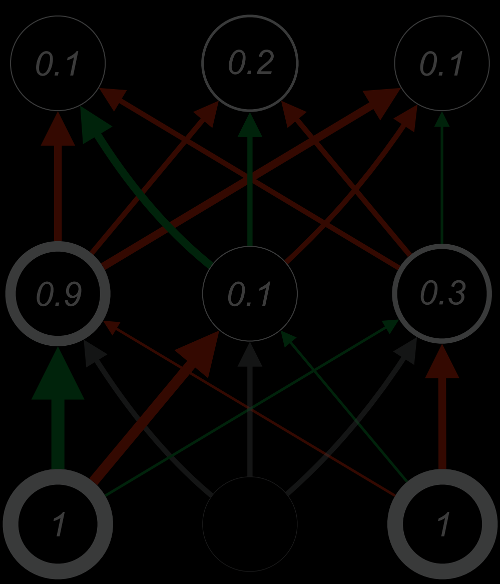


The Solution



The Solution

- The "Generalized Delta Rule:"
  - change each weight to the hidden unit so that it minimizes its contribution to the error at the layer above



- **Version that handles non-linear activation functions:** *(e.g., the logistic)*
  - ***Backpropagation Learning Algorithm*** *(Rumelhart, Hinton & Williams, 1986)* **(aka "deep learning"):**

    $\delta_j = (\Sigma_k \delta_k w_{jk}) * f_j'(net_j)$ *where* $f_j'(net_j) = d(x_j) / d(net_j)$ *(derivate of activation fct)*

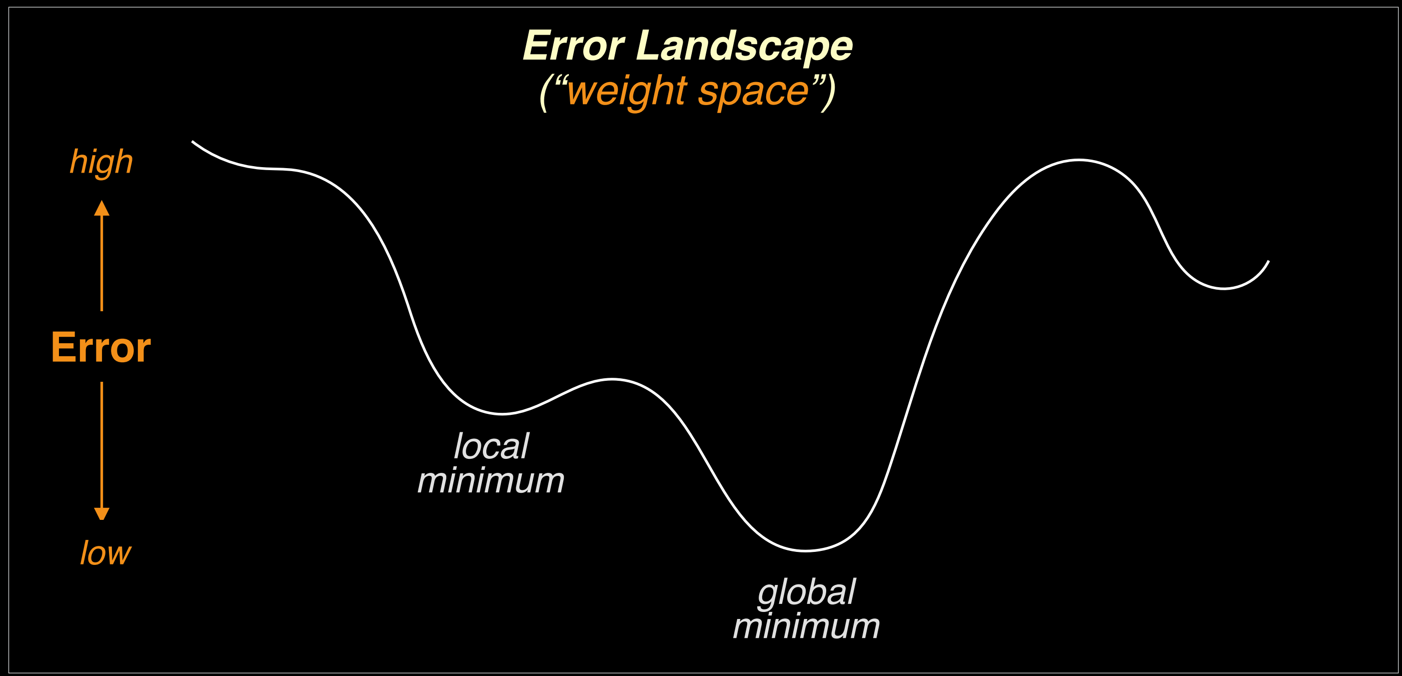
Backpropagation

Backpropagation

- **“Gradient descent” algorithm**
  - **subject to *“local minima”***

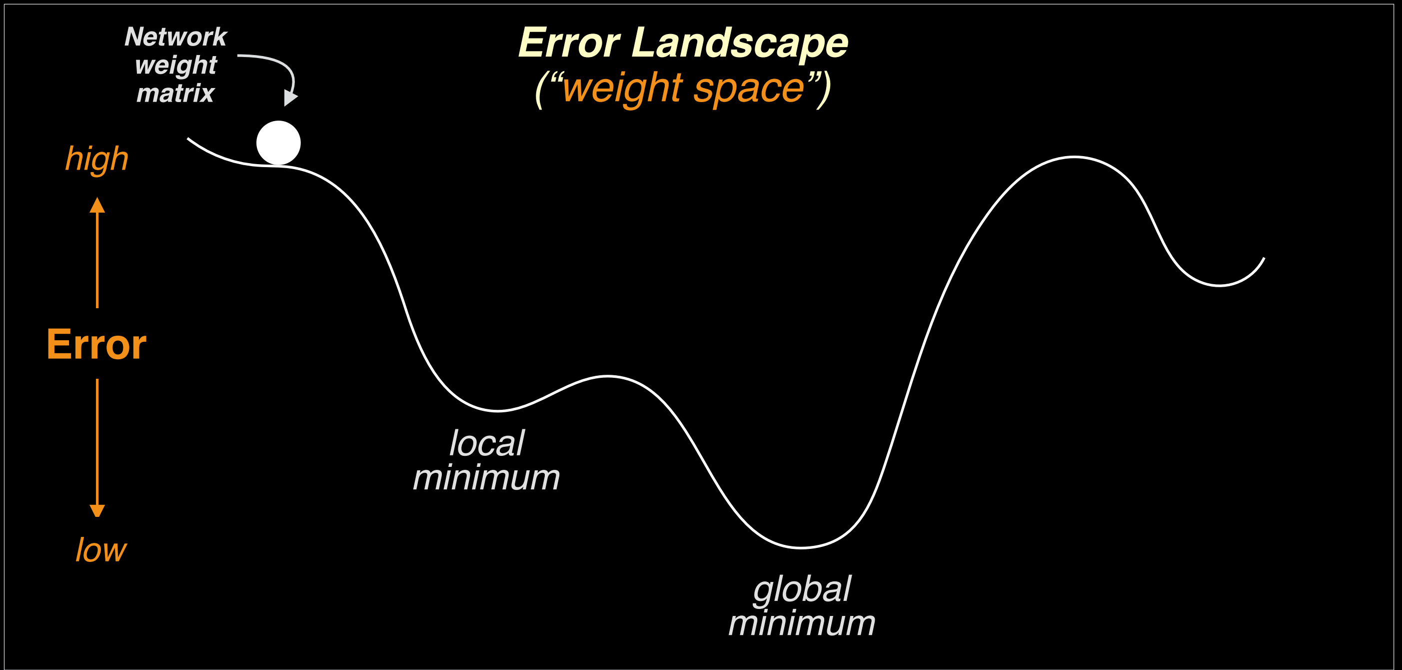
Backpropagation

- "Gradient descent" algorithm
  - subject to *"local minima"*



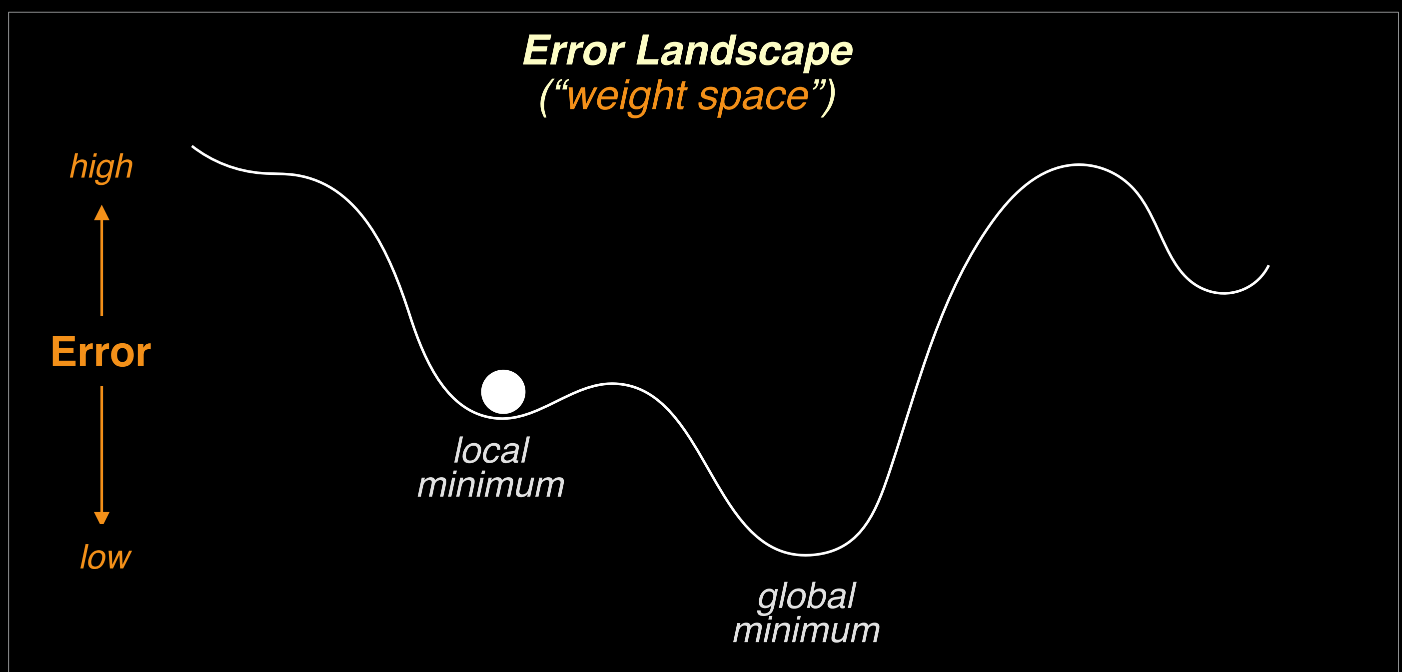
Backpropagation

- **"Gradient descent" algorithm**
  - **subject to *"local minima"***



Backpropagation

- **“Gradient descent” algorithm**
  - **subject to *“local minima”***



Backpropagation

Backpropagation

- **Learning rule:**
  $\delta_j = (\Sigma_k \delta_k w_{jk}) * f_j'(net_j)$ *where* $f_j'(net_j) = d(x_j) / d(net_j)$ *(derivate of activation fct)*

- **"Gradient descent" algorithm**
  - **subject to *"local minima"***

- **Nevertheless, works almost all of the time**
  - **example of how models / simulation:**
    - **are an antidote to self-deception** *(mental "muddles")*
    - **help develop intuitions about behavior of complex** *(e.g., nonlinear)* **systems**

Backpropagation

- **Learning rule:**
  $\delta_j = (\Sigma_k \delta_k w_{jk}) * f_j'(net_j)$ *where* $f_j'(net_j) = d(x_j)/d(net_j)$ *(derivate of activation fct)*

- **"Gradient descent" algorithm**
  - **subject to *"local minima"***

- **Nevertheless, works almost all of the time**
  - **example of how models / simulation:**
    - **are an antidote to self-deception** *(mental "muddles")*
    - **help develop intuitions about behavior of complex** *(e.g., nonlinear)* **systems**

- ***Learns internal representations!***
  (hidden units)

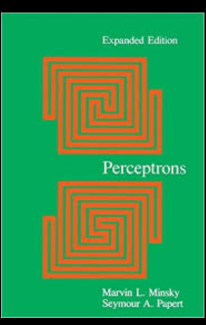
Backpropagation

- **Learning rule:**
  $\delta_j = (\Sigma_k \delta_k w_{jk}) * f_j'(net_j)$ *where* $f_j'(net_j) = d(x_j)/d(net_j)$ *(derivate of activation fct)*

- **"Gradient descent" algorithm**
  - **subject to *"local minima"***

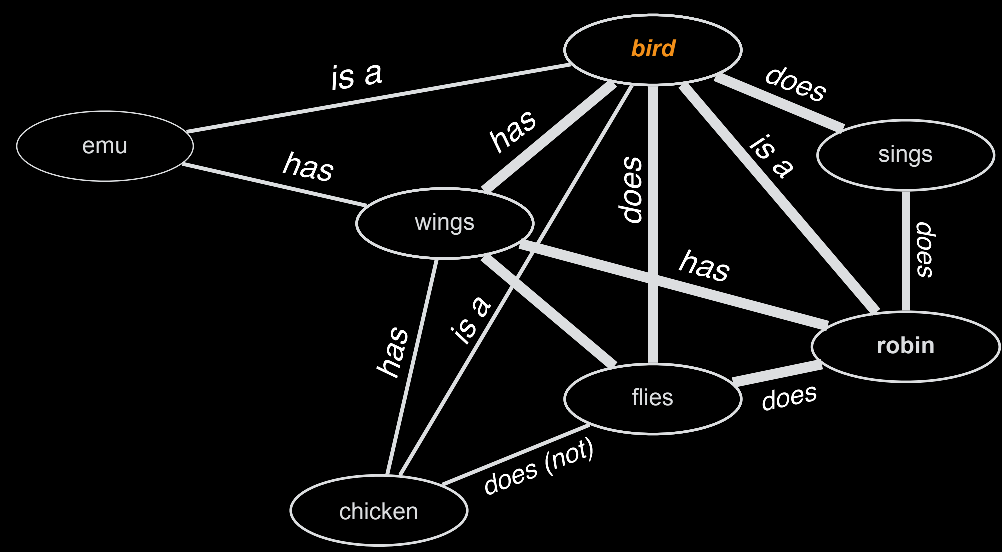
- **Nevertheless, works almost all of the time**
  - **example of how models / simulation:**
    - **are an antidote to self-deception** *(mental "muddles")*
    - **help develop intuitions about behavior of complex** *(e.g., nonlinear)* **systems**

- ***Learns internal representations! (hidden units)***



Backpropagation

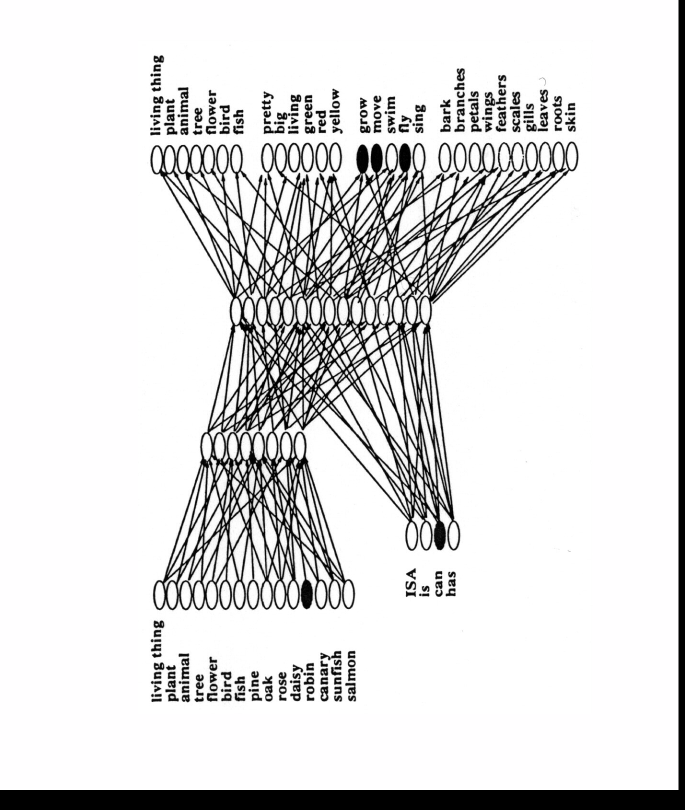
Semantic Knowledge



Semantic Knowledge

Rumelhart's Semantic Network

(Rumelhart & Todd, 1987)

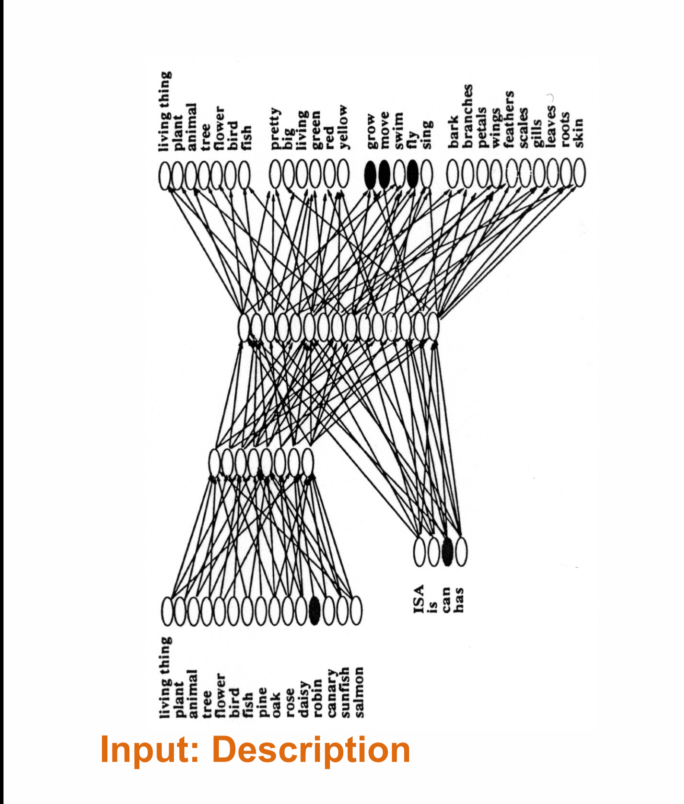


Semantic Knowledge

Rumelhart's Semantic Network

(Rumelhart & Todd, 1987)

- **Inputs:**
  - **Descriptions**
    - features (mouth, wings, leaves...)
    - category (bird, tree…)
    - actions (moves, lives, flies, grows…)
    - names (robin, pine, emu..)

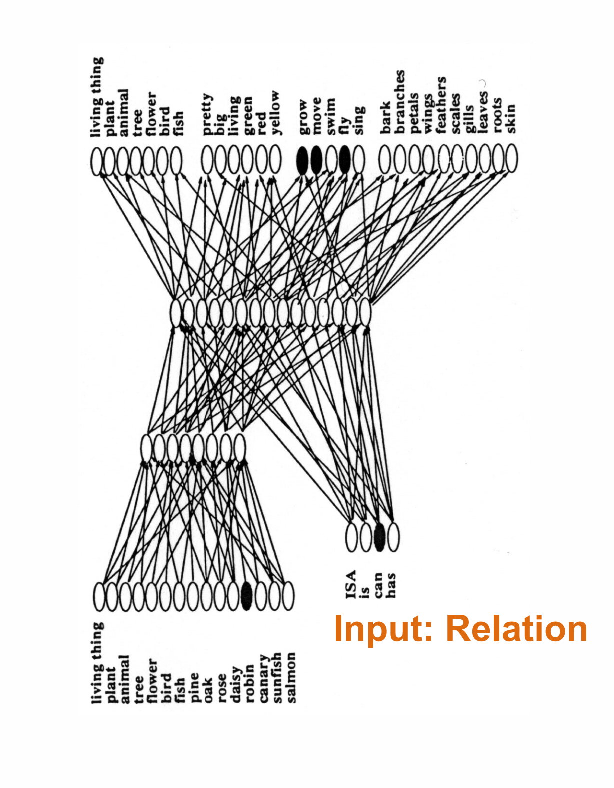


Semantic Knowledge

Rumelhart's Semantic Network

(Rumelhart & Todd, 1987)

- **Inputs:**
  - **Descriptions**
    - features (mouth, wings, leaves...)
    - category (bird, tree…)
    - actions (moves, lives, flies, grows…)
    - names (robin, pine, emu..)
  - **Relations (ISA, is, can, has)**

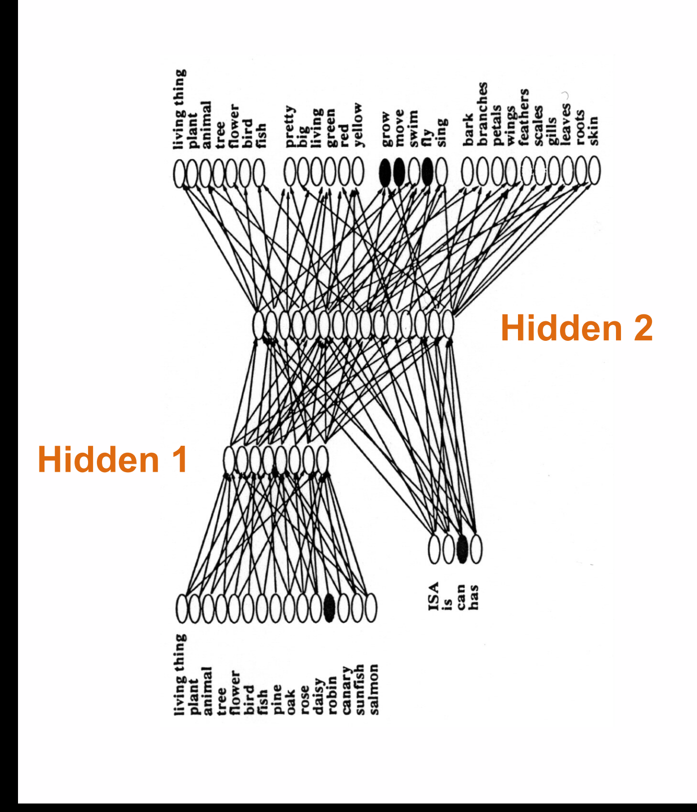


Semantic Knowledge

Rumelhart's Semantic Network

(Rumelhart & Todd, 1987)

- **Inputs:**
  - **Descriptions**
    - features (mouth, wings, leaves...)
    - category (bird, tree…)
    - actions (moves, lives, flies, grows…)
    - names (robin, pine, emu..)
  - **Relations (ISA, is, can, has)**
- **Hidden Layers: associative**

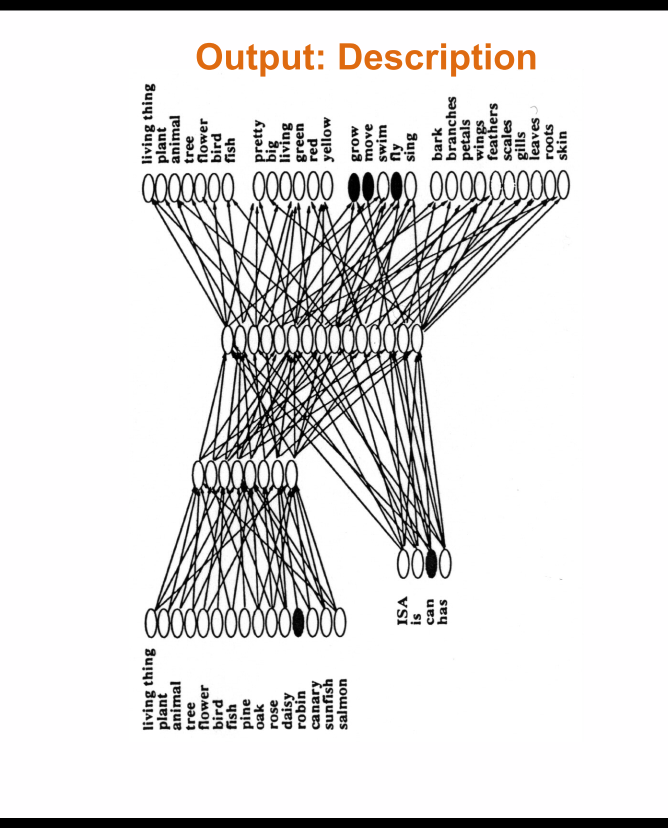


Semantic Knowledge

Rumelhart's Semantic Network

(Rumelhart & Todd, 1987)

- **Inputs:**
  - **Descriptions**
    - features (mouth, wings, leaves...)
    - category (bird, tree…)
    - actions (moves, lives, flies, grows…)
    - names (robin, pine, emu..)
  - **Relations (ISA, is, can, has)**
- **Hidden Layers: associative**
- **Output:**
  - **Descriptions**
    (similar to set used for input)

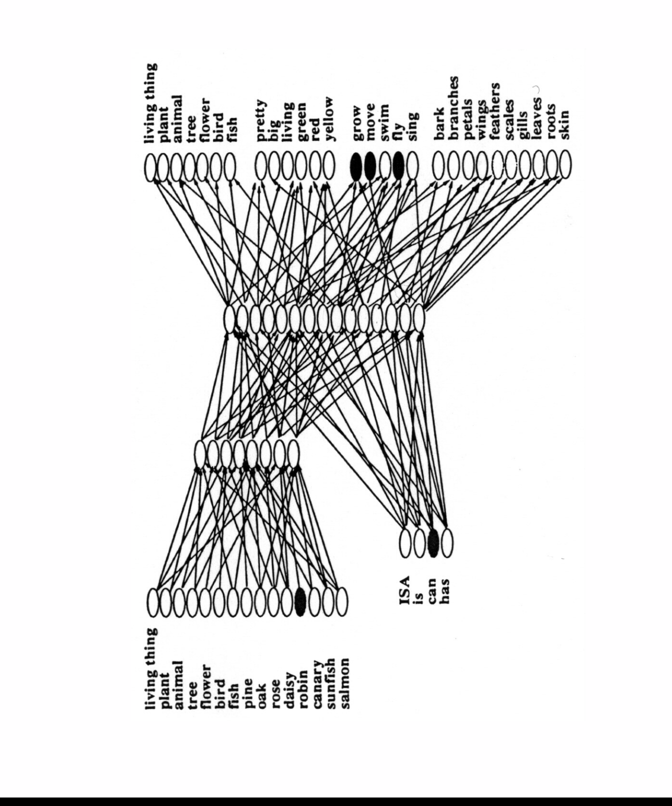


Semantic Knowledge

Rumelhart's Semantic Network

(Rumelhart & Todd, 1987)

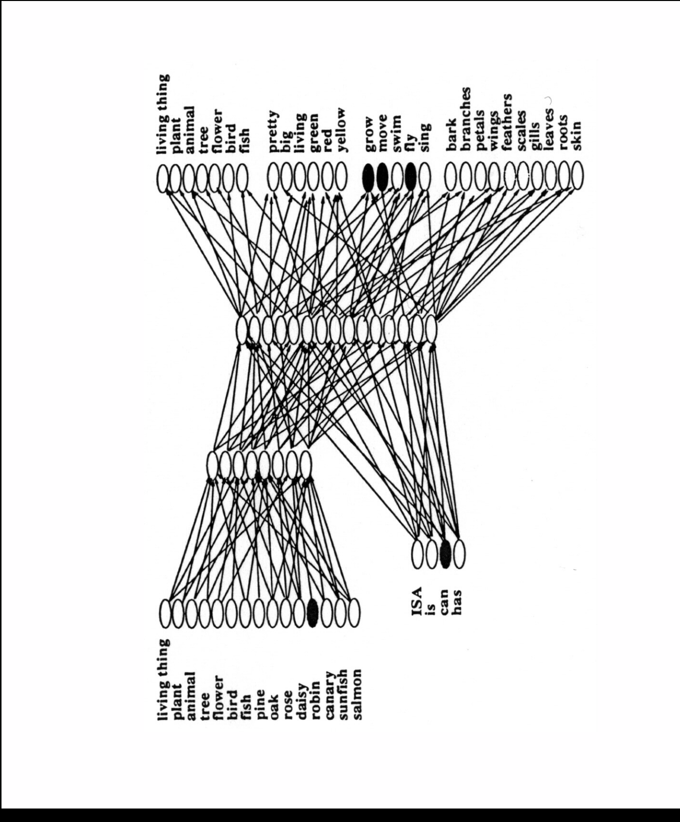
- **Inputs:**
  - **Descriptions**
    - features (mouth, wings, leaves...)
    - category (bird, tree…)
    - actions (moves, lives, flies, grows…)
    - names (robin, pine, emu..)
  - **Relations (ISA, is, can, has)**
- **Hidden Layers: associative**
- **Output:**
  - **Descriptions**
    (similar to set used for input)
- **Training:**
  - **Exemplars of birds and trees**



Semantic Knowledge

Rumelhart's Semantic Network

(Rumelhart & Todd, 1987)



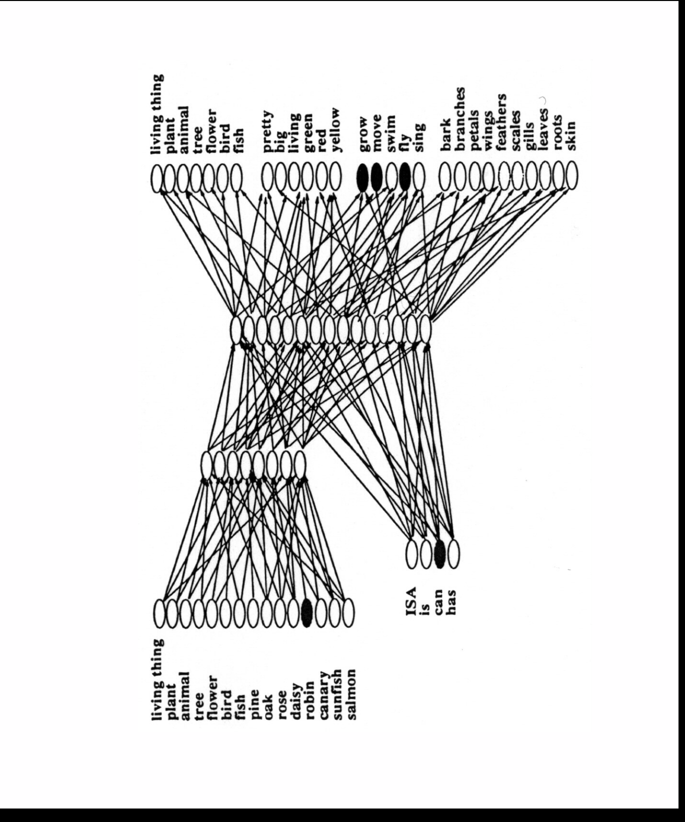
- **Inputs:**
  - **Descriptions**
    - features (mouth, wings, leaves...)
    - category (bird, tree…)
    - actions (moves, lives, flies, grows…)
    - names (robin, pine, emu..)
  - **Relations (ISA, is, can, has)**
- **Hidden Layers: associative**
- **Output:**
  - **Descriptions**
    (similar to set used for input)
- **Training:**
  - **Exemplars of birds and trees**
- **Results: developed a set of internal representations that:**
  - **Had distinct recognizable patterns for different categories (e.g., birds vs. trees)**

Semantic Knowledge

Rumelhart's Semantic Network

(Rumelhart & Todd, 1987)

- **Inputs:**
  - **Descriptions**
    - features (mouth, wings, leaves...)
    - category (bird, tree…)
    - actions (moves, lives, flies, grows…)
    - names (robin, pine, emu..)
  - **Relations (ISA, is, can, has)**
- **Hidden Layers: associative**
- **Output:**
  - **Descriptions**
    (similar to set used for input)
- **Training:**
  - **Exemplars of birds and trees**

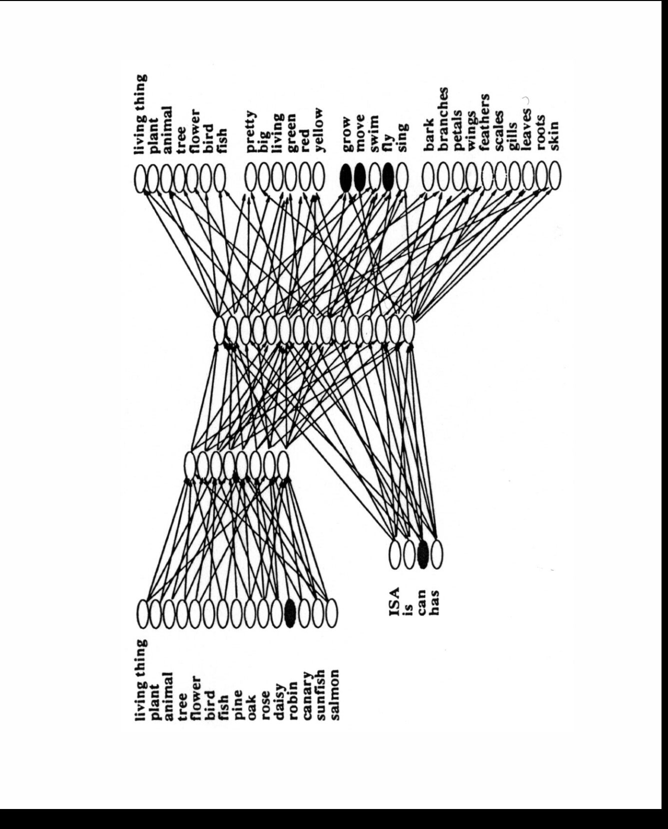


- **Results: developed a set of internal representations that:**
  - **Had distinct recognizable patterns for different categories (e.g., birds vs. trees)**
  - **Assigned dedicated representations to exceptional cases**

Semantic Knowledge

Rumelhart's Semantic Network

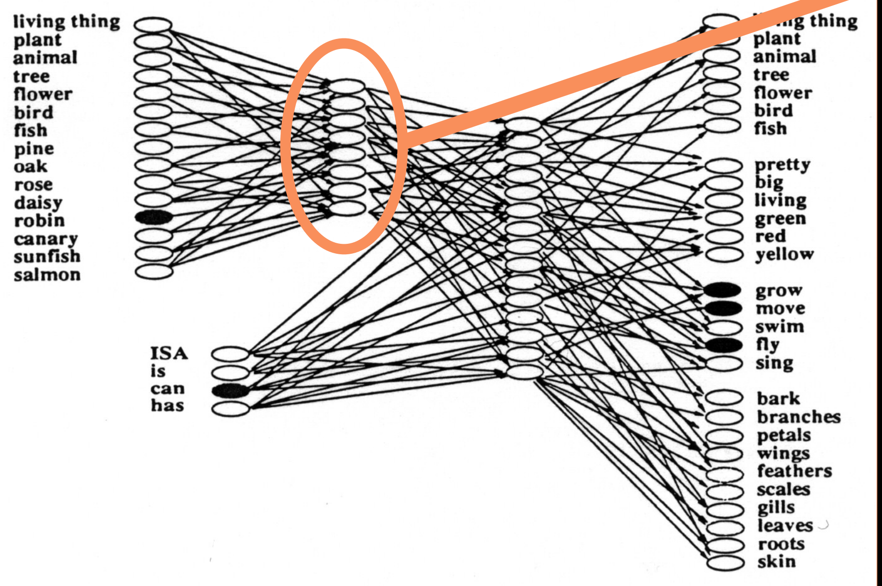
(Rumelhart & Todd, 1987)



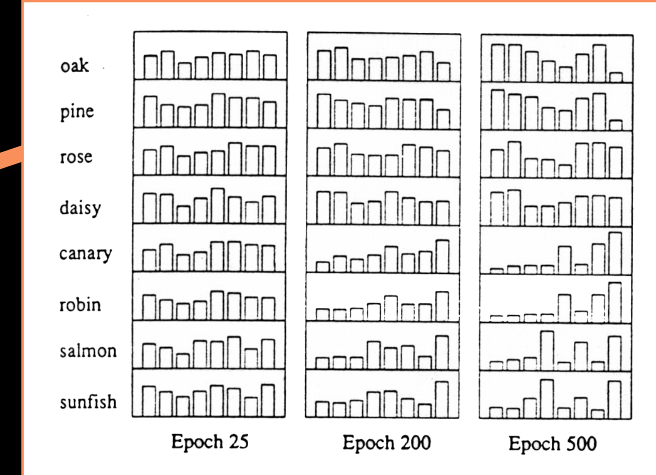
- **Inputs:**
  - **Descriptions**
    - features (mouth, wings, leaves...)
    - category (bird, tree…)
    - actions (moves, lives, flies, grows…)
    - names (robin, pine, emu..)
  - **Relations (ISA, is, can, has)**
- **Hidden Layers: associative**
- **Output:**
  - **Descriptions**
    (similar to set used for input)
- **Training:**
  - **Exemplars of birds and trees**
- **Results: developed a set of internal representations that:**
  - **Had distinct recognizable patterns for different categories (e.g., birds vs. trees)**
  - **Assigned dedicated representations to exceptional cases**
  - **Could properly categorize novel exemplars if they were "typical"**

Rumelhart's Semantic Network

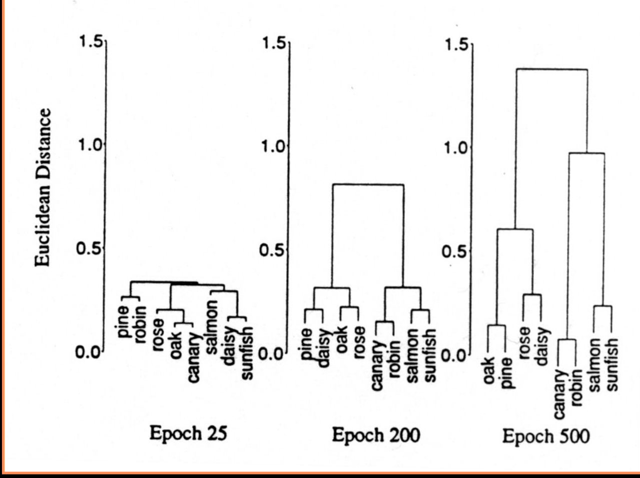
Rumelhart's Semantic Network



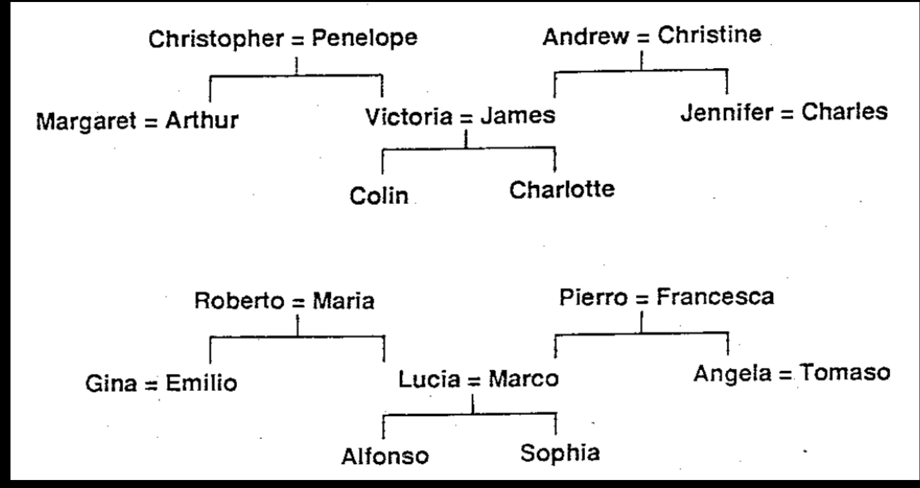
With sufficient training...

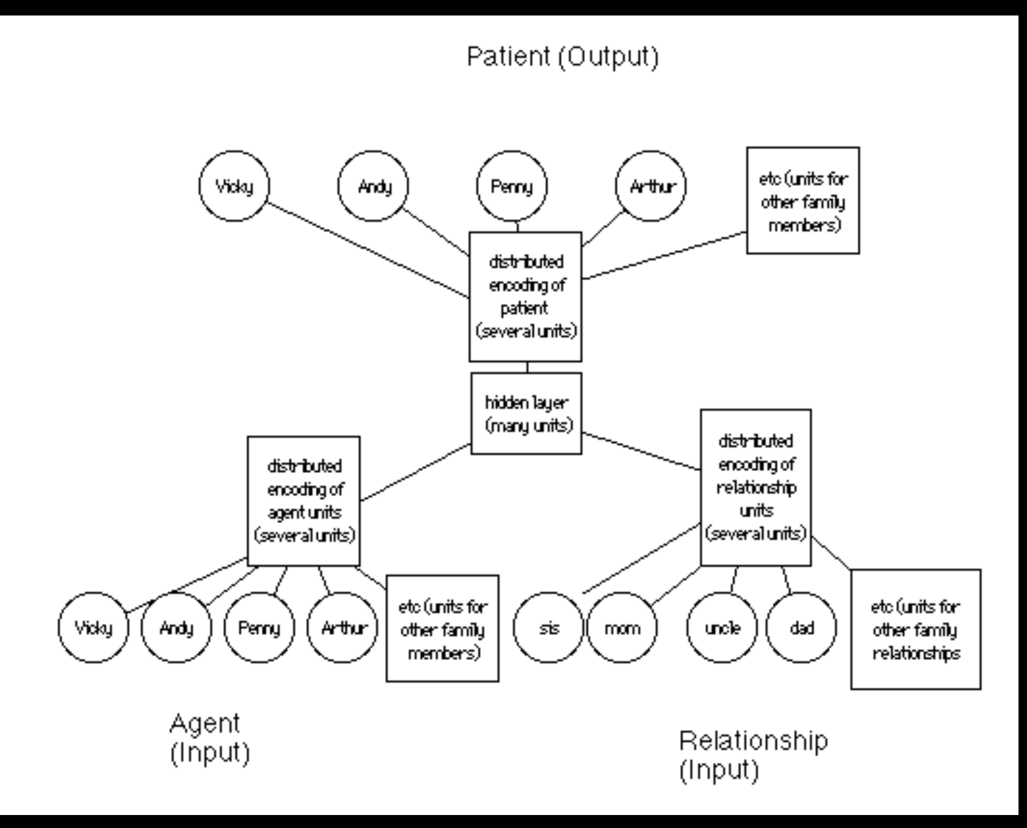


acquired rich semantic structure



Hinton’s Family Trees Model





“Deep Learning”

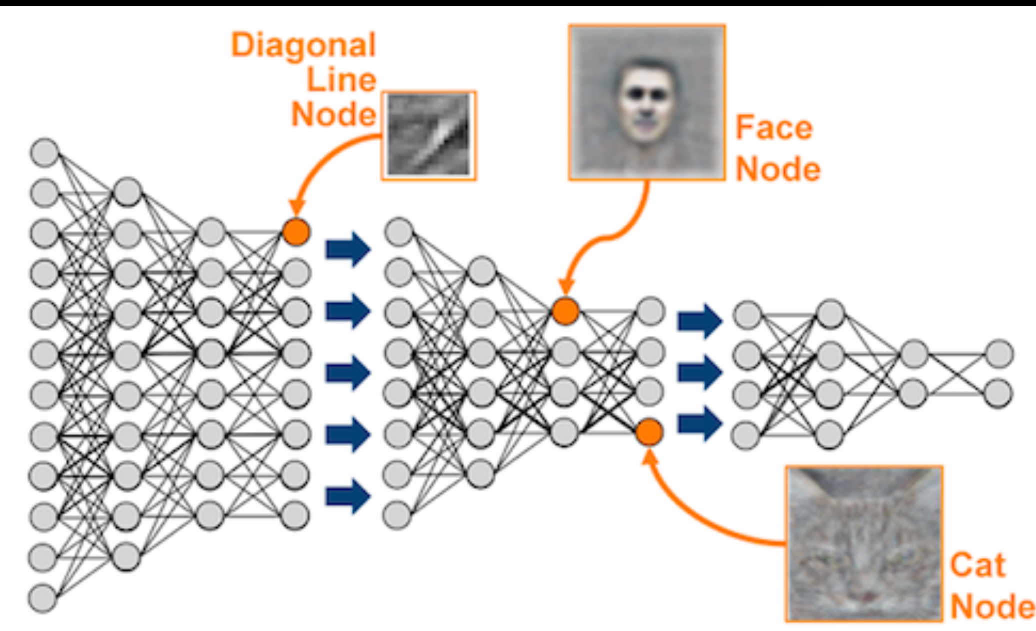
Dave Rumelhart
(UCSD / Stanford)
Backpropagation

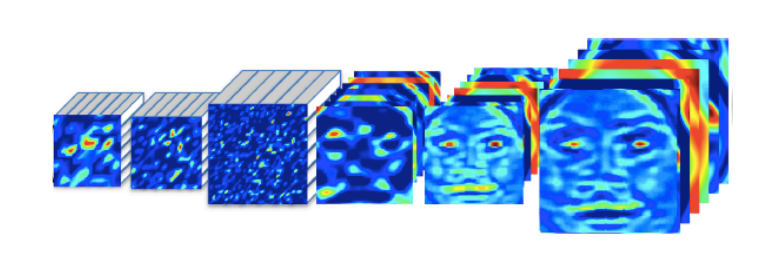
Jann LeCun
(NYU)
Convolutional Networks

Geoff Hinton
(Toronto / Google)
Alex Net

Bruno Olshausen
(Redwood Institute)
Sparse Coding

Fei Fei Li
(Princeton/Stanford)
Image Net





“Deep Learning”