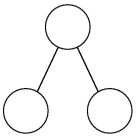
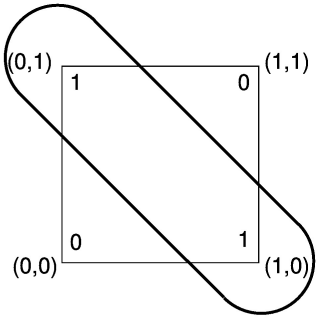
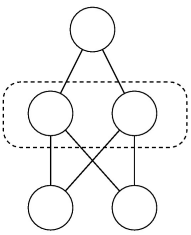
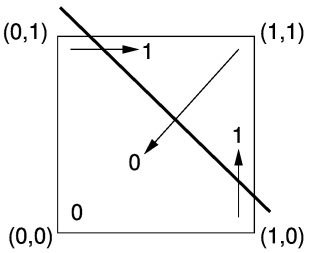
XOR with intermediate ("hidden") units

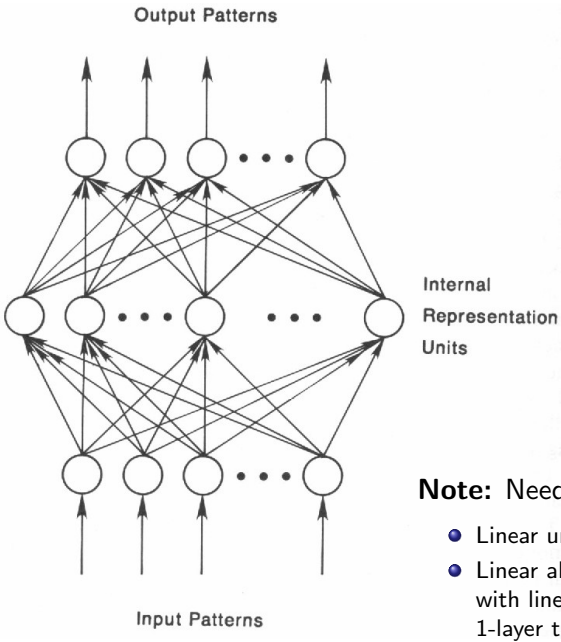


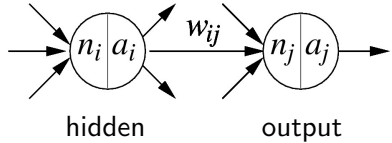






- Intermediate units can re-represent input patterns as new patterns with **altered similarities**
- Target assignments which are not linearly separable in the input space can be linearly separable in the intermediate representational space
- Intermediate units are called "hidden" because their activations are not determined directly by the training environment (inputs and targets) (as in *hidden Markov models*)





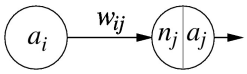
- Hidden-to-output weights can be trained with the Delta rule
- How can we train input-to-hidden weights?
  - Hidden units do not have targets (for determining error)
  - **Trick:** We don't need targets, we just need to know how hidden activations affect error (i.e., error derivatives)

Note: Need **nonlinear** unit function for hidden units

- Linear units cannot alter the relative similarities of patterns
- Linear algebra: Each layer of weights is a matrix multiplication; with linear units, can just multiply the matrices to get equivalent 1-layer transformation (which we know is insufficient)

Delta rule as gradient descent in error (sigmoid units)

$$
\begin{aligned}
n_j &= \sum_i a_i \, w_{ij} \\
a_j &= \frac{1}{1 + \exp(-n_j)} \\
\text{Error } E &= \frac{1}{2} \sum_j (t_j - a_j)^2
\end{aligned}
$$



$$
\begin{array}{ccccccc}
w_{ij} & \searrow & & & t_j & \searrow & \\
a_i & \rightarrow & n_j & \rightarrow & a_j & \rightarrow & E
\end{array}
$$

Gradient descent:

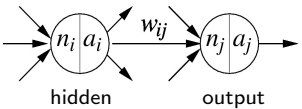
$$
\begin{aligned}
\triangle w_{ij} &= -\epsilon \, \frac{\partial E}{\partial w_{ij}} \\
\frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial a_j} \quad \frac{\mathrm{d}a_j}{\mathrm{d}n_j} \quad \frac{\partial n_j}{\partial w_{ij}} \\
&= -(t_j - a_j) \quad a_j(1 - a_j) \quad a_i \\
\triangle w_{ij} &= -\epsilon \, \frac{\partial E}{\partial w_{ij}} = \epsilon (t_j - a_j) \; a_j (1 - a_j) \; a_i
\end{aligned}
$$

Generalized Delta rule ("back-propagation")

$$n_j = \sum_i a_i \, w_{ij}$$

$$a_j = \frac{1}{1 + \exp(-n_j)}$$

$$\text{Error } E = \frac{1}{2} \sum_j (t_j - a_j)^2$$



$$n_i \rightarrow a_i \xrightarrow{w_{ij}} n_j \rightarrow a_j \xrightarrow{t_j} E$$

Gradient descent: $\triangle w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}}$

Intermediate notation ("input derivatives" in Lens)

$$\frac{\partial E}{\partial n_j} = \frac{\partial E}{\partial a_j} \frac{\mathrm{d}a_j}{\mathrm{d}n_j} = -(t_j - a_j) \; a_j (1 - a_j)$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial n_j} \frac{\partial n_j}{\partial w_{ij}} = \frac{\partial E}{\partial n_j} \, a_i$$

"output derivatives" for hidden units like those for output units

$$\frac{\partial E}{\partial a_i} = \sum_j \frac{\partial E}{\partial n_j} \frac{\partial n_j}{\partial a_i} = \sum_j \frac{\partial E}{\partial n_j} \, w_{ij}$$

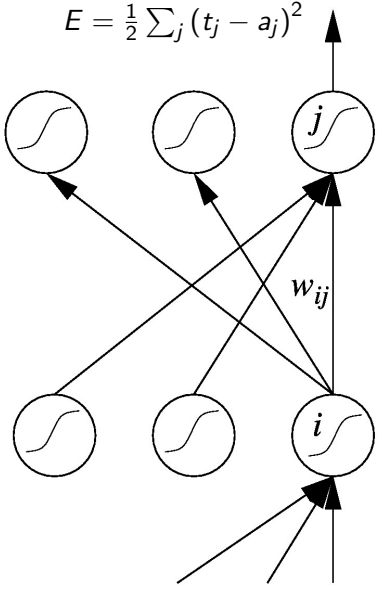
Back-propagation (generalization of Delta rule to multilayer networks)

Forward pass ($\Uparrow$)

$$a_j = \frac{1}{1+\exp(-n_j)}$$

$$n_j = \sum_i a_i w_{ij}$$

$$a_i = \frac{1}{1+\exp(-n_i)}$$



Backward pass ($\Downarrow$)

$$\frac{\partial E}{\partial a_j} = -(t_j - a_j)$$

$$\frac{\partial E}{\partial n_j} = \frac{\partial E}{\partial a_j} a_j(1-a_j)$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial n_j} a_i$$

$$\frac{\partial E}{\partial a_i} = \sum_j \frac{\partial E}{\partial n_j} w_{ij}$$

$$\frac{\partial E}{\partial n_i} = \frac{\partial E}{\partial a_i} a_i(1-a_i)$$