3.1 The Backpropagation Algorithm (Rumelhart et al., 1986)

Formulation for a feed-forward 2-layer network of sigmoid units, the stochastic version

Idea: Gradient descent over the entire vector of network weights.

Initialize all weights to small random numbers.

Until satisfied, // *stopping criterion* to be (later) defined

for each training example,

1. **input the training example to the network, and compute the network outputs**
2. **for each output unit k:**
   $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
3. **for each hidden unit h:**
   $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} \delta_k$
4. **update each network weight w_{ji}:**
   $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$ **where** $\Delta w_{ji} = \eta \delta_j x_{ji}$,
   and x_{ji} is the ith input to unit j.

Derivation of the Backpropagation rule,

(following [Tom Mitchell, 1997], pag. 101–103)

Notations:

x_{ji}: the ith input to unit j;
(j could be either hidden or output unit)

w_{ji}: the weight associated with the ith input to unit j

$\boldsymbol{net}_j = \sum_i w_{ji} x_{ji}$

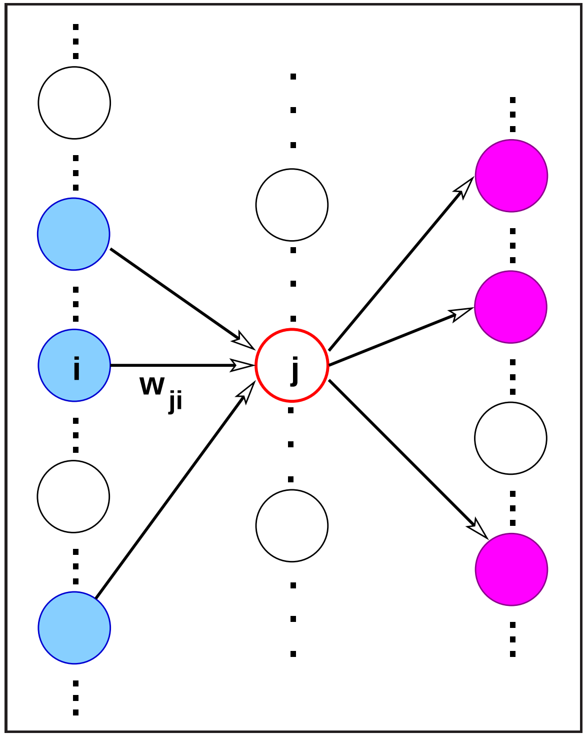
σ: the sigmoid function

o_j: the output computed by unit j; ($o_j = \sigma(\boldsymbol{net}_j)$)

outputs: the set of units in the final layer of the network

Downstream(j): the set of units whose immediate inputs include the output of unit j

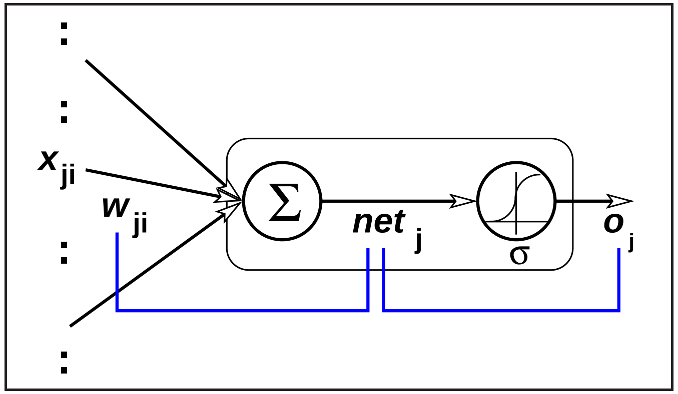
E_d: the training error on the example d (summing over all of the network output units)



Legend: in magenta color, units belonging to *Downstream(j)*

Preliminaries

$$E_d(\vec{w}) \;=\; \frac{1}{2}\sum_{k\in \boldsymbol{outputs}}(t_k - o_k)^2 = \frac{1}{2}\sum_{k\in \boldsymbol{outputs}}(t_k - \sigma(\boldsymbol{net}_k))^2$$

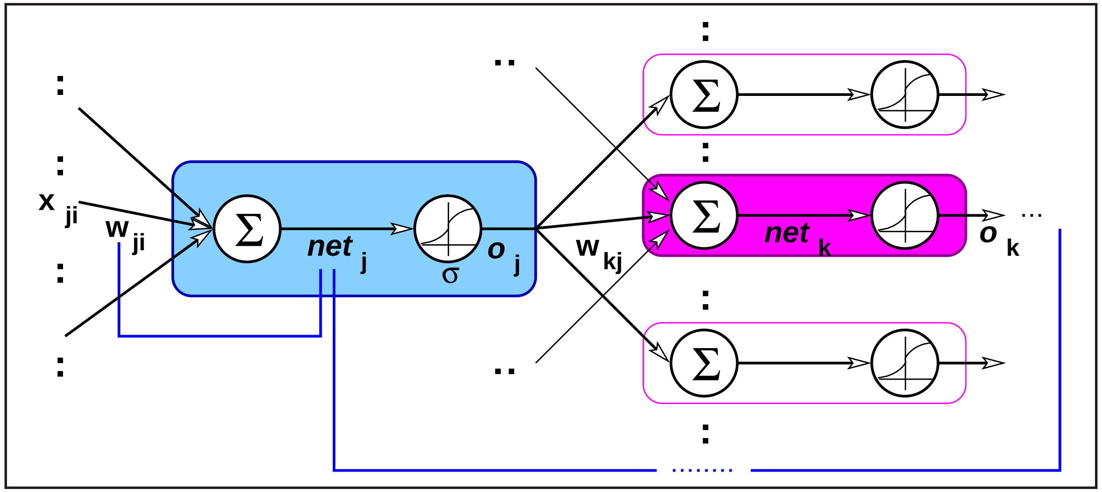


Common staff for both hidden and output units:

$$\boldsymbol{net}_j \;=\; \sum_i w_{ji}x_{ji}$$

$$\Rightarrow \frac{\partial E_d}{\partial w_{ji}} \;=\; \frac{\partial E_d}{\partial \boldsymbol{net}_j}\frac{\partial \boldsymbol{net}_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial \boldsymbol{net}_j}x_{ji}$$

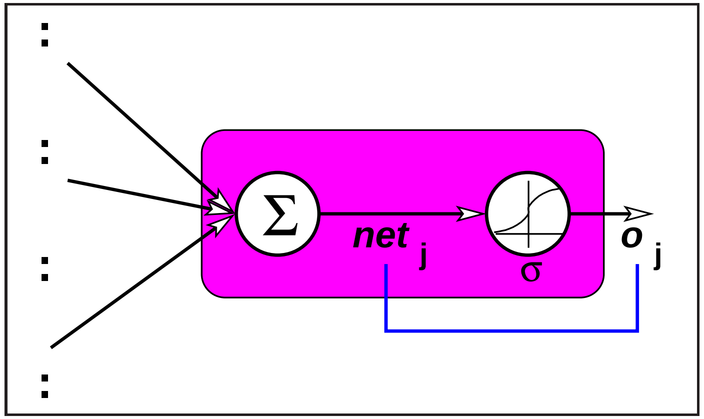
$$\Rightarrow \Delta w_{ji} \;\stackrel{def}{=}\; -\eta\frac{\partial E_d}{\partial w_{ji}} = -\eta\frac{\partial E_d}{\partial \boldsymbol{net}_j}x_{ji}$$



Note: In the sequel we will use the notation: $\delta_j = -\dfrac{\partial E_d}{\partial \boldsymbol{net}_j} \Rightarrow \Delta w_{ji} = \eta\delta_j x_{ji}$

Stage/Case 1: Computing the increments (Δ) for output unit weights

$$
\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j} \\
\frac{\partial o_j}{\partial net_j} &= \frac{\partial \sigma(net_j)}{\partial net_j} = o_j(1-o_j) \\
\frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k\in outputs}(t_k-o_k)^2 \\
&= \frac{\partial}{\partial o_j}\frac{1}{2}(t_j-o_j)^2 = \frac{1}{2}2(t_j-o_j)\frac{\partial(t_j-o_j)}{\partial o_j} \\
&= -(t_j-o_j)
\end{aligned}
$$



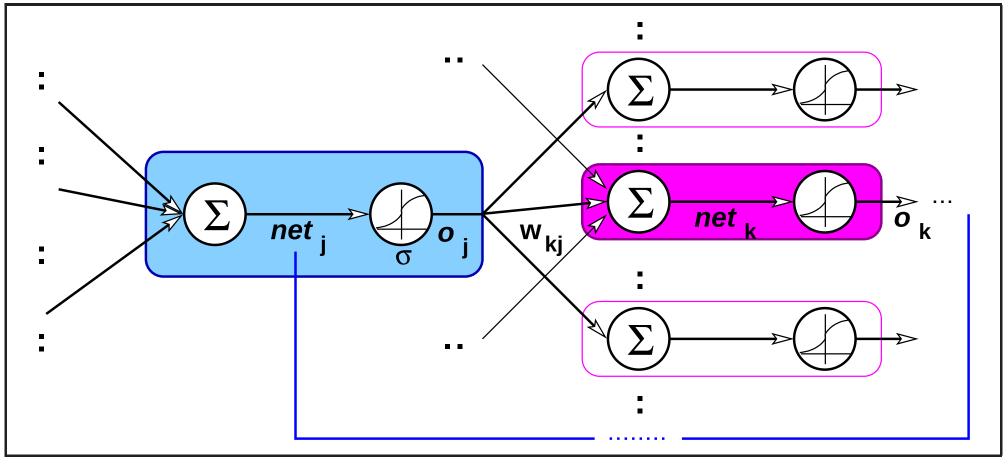
$$
\begin{aligned}
\Rightarrow \frac{\partial E_d}{\partial net_j} &= -(t_j-o_j)o_j(1-o_j) = -o_j(1-o_j)(t_j-o_j) \\
\Rightarrow \delta_j &\stackrel{not.}{=} -\frac{\partial E_d}{\partial net_j} = o_j(1-o_j)(t_j-o_j) \\
\Rightarrow \Delta w_{ji} &= \eta\delta_j x_{ji} = \eta o_j(1-o_j)(t_j-o_j)x_{ji}
\end{aligned}
$$

Stage/Case 2: Computing the increments (Δ) for hidden unit weights

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$



$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k w_{kj} o_j (1 - o_j)$$

Therefore:

$$\delta_j \stackrel{not}{=} -\frac{\partial E_d}{\partial net_j} = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

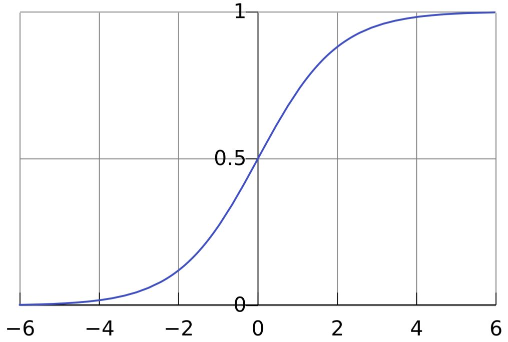
$$\Delta w_{ji} \stackrel{def}{=} -\eta \frac{\partial E_d}{\partial w_{ji}} = -\eta \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji} = \eta \delta_j x_{ji} = \eta \left[\ o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}\ \right] x_{ji}$$

Convergence of Backpropagation
for NNs of Sigmoid units

Nature of convergence

- **The weights are initialized near zero; therefore, initial decision surfaces are near-linear.**

  Explanation: o_j **is of the form** $\sigma(\vec{w} \cdot \vec{x})$, **therefore** $w_{ji} \approx 0$ **for all** i, j; **note that the graph of** σ **is approximately liniar in the vecinity of** 0.



- **Increasingly non-linear functions are possible as training progresses**

- **Will find a local, not necessarily global error minimum. In practice, often works well (can run multiple times).**

More on Backpropagation

- Easily generalized to arbitrary directed graphs
- Training can take thousands of iterations $\rightarrow$ slow!
- Often include weight momentum α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{ij} + \alpha \Delta w_{ij}(n-1)$$

  Effect:

  - speed up convergence (increase the step size in regions where the gradient is unchanging);
  - "keep the ball rolling" through local minima (or along flat regions) in the error surface

- Using network after training is very fast
- Minimizes error over *training* examples;
  Will it generalize well to subsequent examples?