

# Backpropagation

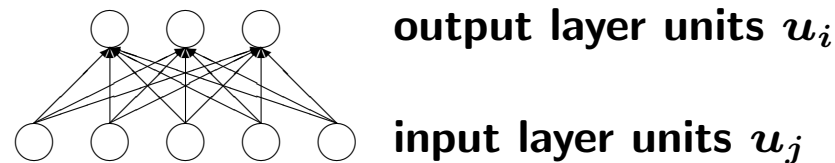
---

- ▶ Can we learn more complex functions like linearly separable ones?
- ▶ **Observation:** Exclusive-or can be computed by a 3-layer network.
- ▶ How can we train the weights in a multi-layer network?
  - ▷ Backpropagation
  - ▷ Rumelhart, Hinton, Williams: Learning Internal Representations by Error Propagation. In: Parallel Distributed Processing (Rumelhard, McClelland and the PDP Research Group, eds.), MIT Press, vol.1, ch.8: 1996.

# 2-Layer Networks of Linear Units

---

- ▶ 2-layer networks:



- ▶ Linear units:  $v_i = \sum_j w_{ij} v_j$
- ▶ Classification problem  $\{(\vec{i}^k, \vec{o}^k) \mid 1 \leq k \leq d\}$ .
- ▶ Ideally:  $v_i^k = \sum_j w_{ij} v_j^k = o_i^k$  for all  $1 \leq k \leq d$ .
- ▶ How should the weights look like?
- ▶ Can we learn such weights?
- ▶ How does a learning rule look like?

# Gradient Descent Learning

---

- ▶ Error measure or cost function:

$$E(\bar{w}) = \sum_k E^k(\bar{w}),$$

where

$$E^k(\bar{w}) = \frac{1}{2} \sum_i (o_i^k - v_i^k)^2 = \frac{1}{2} \sum_i (o_i^k - \sum_j w_{ij} v_j^k)^2.$$

- ▶  $E(\bar{w})$  and  $E^k(\bar{w})$  are differentiable functions of the weights.
- ▶ **Idea:** use gradient descent to minimize the error of the network.
- ▶ **Proposition:**  $\frac{dE^k}{dw_{ij}} = -(o_i^k - v_i^k)v_j^k$ .
- ▶  $\delta_i^k = (o_i^k - v_i^k)$  is the error of the  $i$ -th output unit wrt  $k$ -th pattern.
- ▶ Change weight  $\Delta^k w_{ij} \sim \delta_i^k v_j^k$ .

# Gradient Descent Learning Algorithm

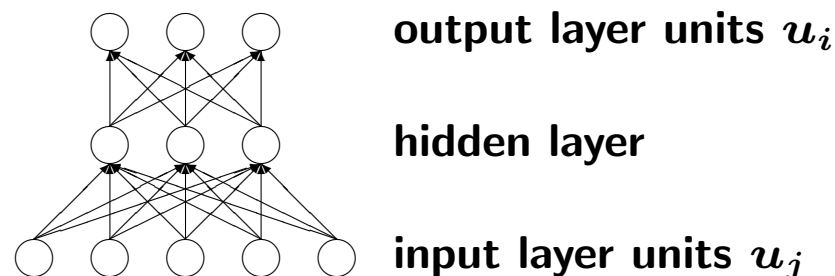
---

- ▶ initialize weights arbitrarily; select learning rate  $\eta$ .
- ▶ do until all patterns are correctly classified.
  - ▷ Select (fairly) and present input pattern  $\vec{i}^k$ .
  - ▷ Wait until the network produces an output.
  - ▷ Compare produced with desired output.
  - ▷ Change weights according to  $\Delta^k w_{ij} = \eta(o_i^k - v_i^k)v_j^k$ .

# Multi-layer networks

---

▶ 3-layer networks:



- ▶ **Exercise:** Show that multi-layer networks of linear units behave as 2-layer networks of linear units.
- ▶ We assume that output functions  $g$  are monotone increasing and differentiable.
- ▶ For simplicity we assume that all units in the hidden and output layer have the same output function.

# Backpropagation Algorithm

---

- ▶ initialize weights arbitrarily; select learning rate  $\eta$ .
- ▶ do until all patterns are correctly classified.
  - ▶ Select (fairly) and present input pattern  $\vec{i}^k$ .
  - ▶ Wait until the network produces an output.
  - ▶ Compare produced with desired output.
  - ▶ Change weights by  $\Delta^k w_{ij} = \eta \delta_i^k v_j^k$ , where

$$\delta_i^k = \begin{cases} g'(p_i^k) \times (o_i^k - v_i^k) & \text{if } u_i \text{ is an output unit} \\ g'(p_i^k) \times \sum_l \delta_l^k w_{li} & \text{if } u_i \text{ is a hidden unit} \end{cases}$$

# Output Function

---

- ▶ Sigmoidal function:

$$v_i = \frac{1}{1 + e^{-(\sum_j w_{ij}v_j + \theta_i)}}$$

- ▶ We find

$$\frac{dv_i}{d(\sum_j w_{ij}v_j - \theta_i)} = v_i(1 - v_i).$$

- ▶ Hence

$$\delta_i^k = \begin{cases} v_i^k(1 - v_i^k)(o_i^k - v_i^k) & \text{if } u_i \text{ is an output unit} \\ v_i^k(1 - v_i^k) \sum_l \delta_l^k w_{li} & \text{if } u_i \text{ is a hidden unit} \end{cases}$$

- ▶ Units are active if  $v_i \geq 0.9$  and passive if  $v_i \leq 0.1$ .

# Properties

---

▶ Learning rate  $\eta$ :

- ▶ If  $\eta$  is large, then system learns rapidly but may oscillate.
- ▶ If  $\eta$  is small, then system learns slowly but will not oscillate.
- ▶ In the ideal case  $\eta$  should be adapted during learning:

$$\Delta w_{ij}(t + 1) = \eta \delta_i(t) v_j(t) + \alpha \Delta w_{ij}(t)$$

where  $\alpha$  is a constant and  $\alpha \Delta w_{ij}(t)$  is called **momentum term**.

- ▶ Almost all functions can be learned.
- ▶ Learning is NP-hard.