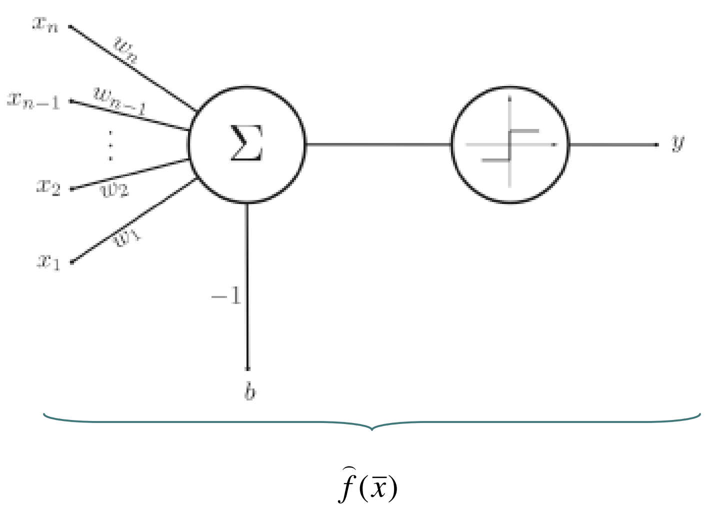
Training using Errors

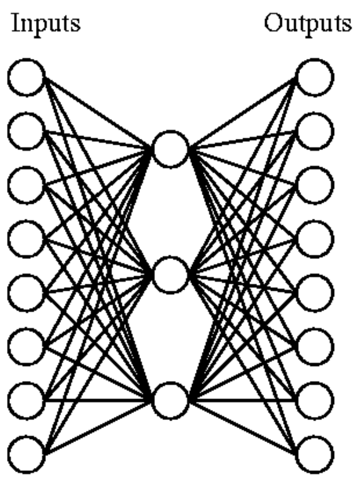
- In training a neural network error is very important
- Only errors allow us to refine the network weights
- We continue to refine the weights until the network classifies perfectly or with an acceptable error margin

Errors in the Perceptron



Initialize $\overline{w}$ and b to random values.
repeat
  for each $(\overline{x}_i, y_i) \in D$ **do**
    if $\hat{f}(\overline{x}_i) \neq y_i$ **then**
      $\overline{w} \leftarrow \overline{w} + \Delta\overline{w}$
      $b \leftarrow b + \Delta b$
    end if
  end for
until D is perfectly classified.
return $\overline{w}$ and b

- Single layer -- we can update the weights directly!



Backpropagation(*training_examples*, η, *in, out, hidden*)

Each training example is a pair of the form (**x**, **y**), where **x** is the vector of network input values and **y** is the vector of target network output values.
η is the learning rate, *in* is the number of network inputs, *out* is the number of output units and *hidden* is the number of units in the hidden layer.
The output from unit i is denoted o_i, and the weight from unit i to unit j is denoted w_{ij}.

- Create a feed-forward network with *in* inputs, *hidden* hidden units, and *out* output units.
- Initialize all network weights to small random numbers (-.05 ~ .05).
- Until termination condition is met, do
  - For each (**x**, **y**) in *training_examples*, do

    Propagate the input forward through the network:
    - Input the instance **x** to the network and compute the output o_u of every unit u in the network.

    Propagate the errors backward through the network:
    - For each network output unit k, calculate its error term δ_k

    $$\delta_k \leftarrow o_k(1-o_k)(\mathbf{y}_k - o_k)$$

    - For each hidden unit h, calculate its error term δ_h

    $$\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{hk}\delta_k$$

    - Update each network weight w_{ji}

    $$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}, \qquad \text{with } \Delta w_{ij} = -\eta \delta_j o_i$$

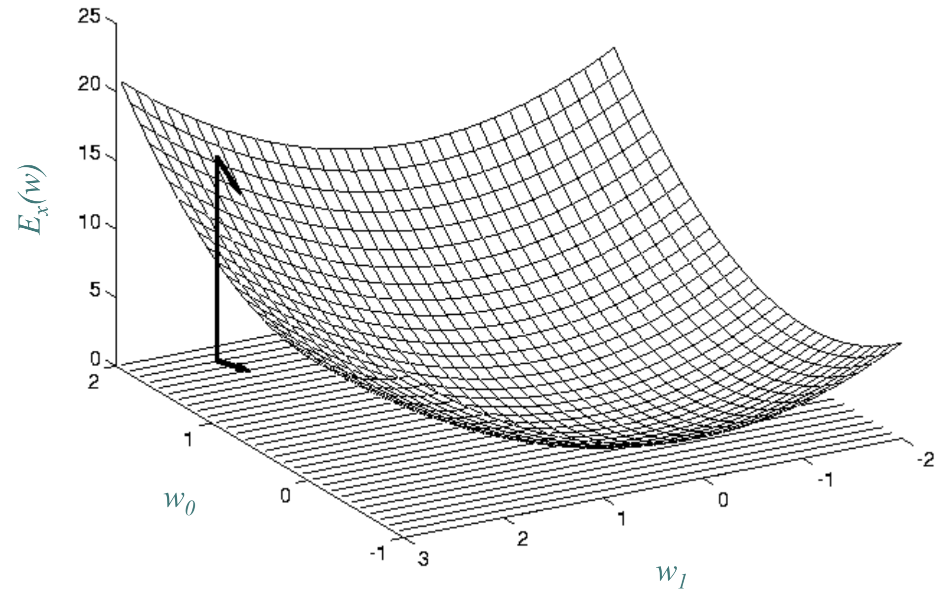
Neural Network Learning

- Define the network error as

$$E_x = \sum_k (\mathbf{y}_k - o_k)^2$$

  for some $x \in X$, where i is an index over the output units.
- Let $E_x(w)$ be the error E_x as a function of the weights w.
- Use the gradient (slope) of the error surface to guide the search towards appropriate weights:

$$\Delta w_{ij} = -\eta \frac{\partial E_x}{\partial w_{ij}}$$



Neural Network Learning

This is utilized during backpropagation:

- The error terms in Δw_{ji} are based on derivatives of the transfer function,

$$\Delta w_{ij} = -\eta \frac{\partial E_x}{\partial w_{ij}} = -\eta \delta_j o_i .$$

- Backpropagation converges on a set of weights that minimize the value of the error surface (possible local minima!)