CS 412

FEB 27TH – BACKPROPAGATION

Perceptron

What is the problem with the simple perceptron?

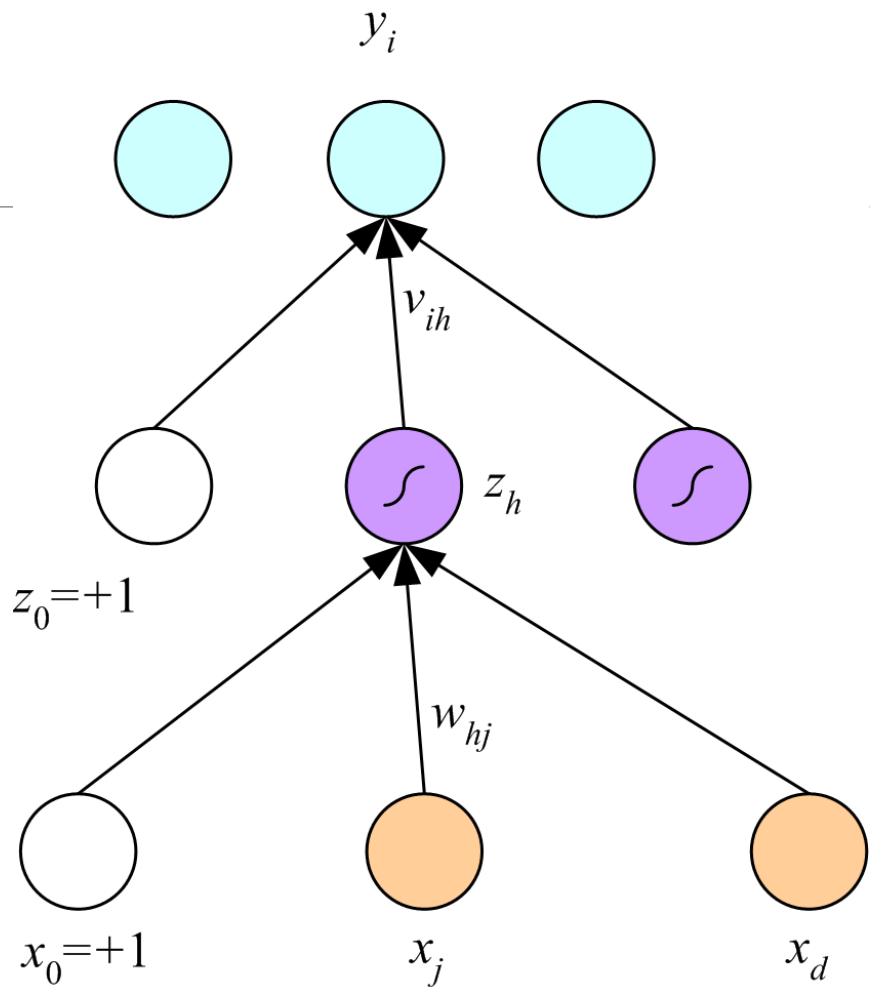
- It can't model non-linear data

How do we fix this?

- SVM fixed this by using the kernel methods
- Can the perceptron? **Yes**, but that's not what the neural network does

Let's add multiple layers to the perceptron

- At each level we have a **regression** model defined by the activation function and **always** a constant w_0



How to train?

We (hopefully) have some idea of how a particular set of weights causes the neural network to make a decision

- We have some vector of inputs X_d that are all fed as parameters to some number of nodes
- Each of these nodes outputs a sigmoid function to the next hidden layer
- This process eventually leads to the final layer, which makes the final prediction

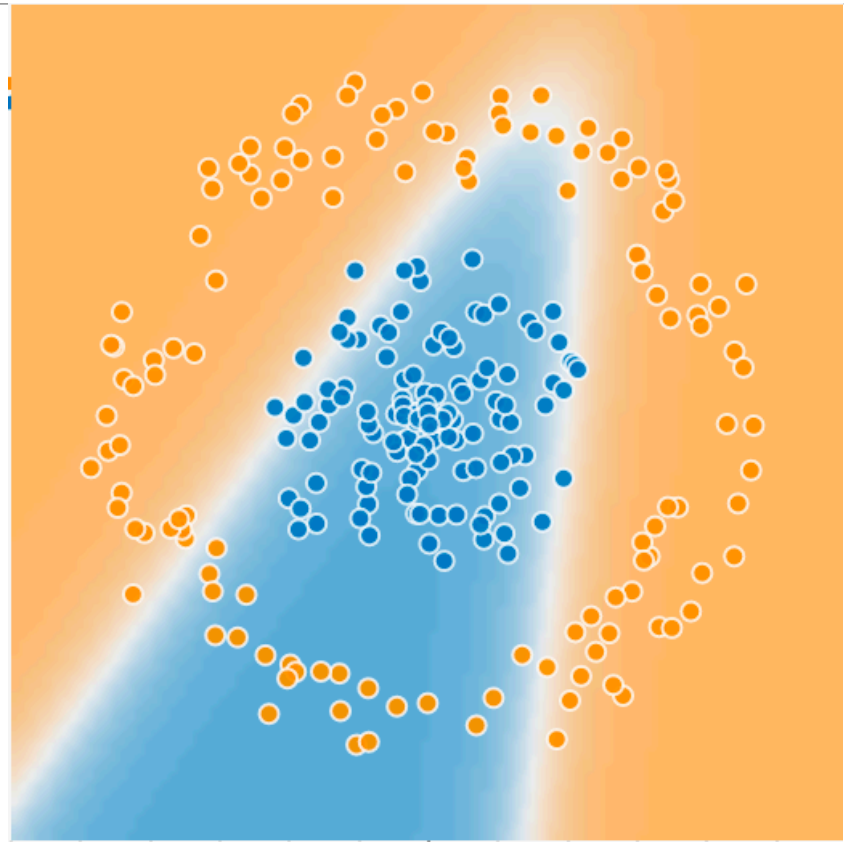
How to train?

What is the structure of the neural network that produced this decision region?

- Blue is positive, orange is negative
- What do the white regions represent?

Two lines from two middle “hidden nodes” with sigmoid behavior

What might the weights look like for each of these nodes?



Algorithm 8.4: Perceptron algorithm

Input: linearly separable data set $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \{-1, +1\}$ for $i = 1 : N$;
Initialize $\boldsymbol{\theta}_0$;
$k \leftarrow 0$;
repeat
    $k \leftarrow k + 1$;
    $i \leftarrow k \bmod N$;
    if $\hat{y}_i \neq y_i$ **then**
        $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + y_i \mathbf{x}_i$
    else
        no-op
until *converged*;

What's wrong with this approach?

It's an easy way to separate linear data. But what's wrong?

- Data isn't always linearly separable
- Nodes of the neural network work in conjunction with each other
  - This approach would mean that all internal nodes convert to the same point!

How do we get around this?

- Random weights
- Feedback between nodes

How to Train

- Training to minimize sum-squared error

$$\ell(W) = \frac{1}{2}\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]^2$$

$$\frac{\partial \ell(W)}{\partial w_k} = -\sum_j [y^j - g(w_0 + \sum_i w_i x_i^j)]\ x_k^j\ g'(w_0 + \sum_i w_i x_i^j)$$

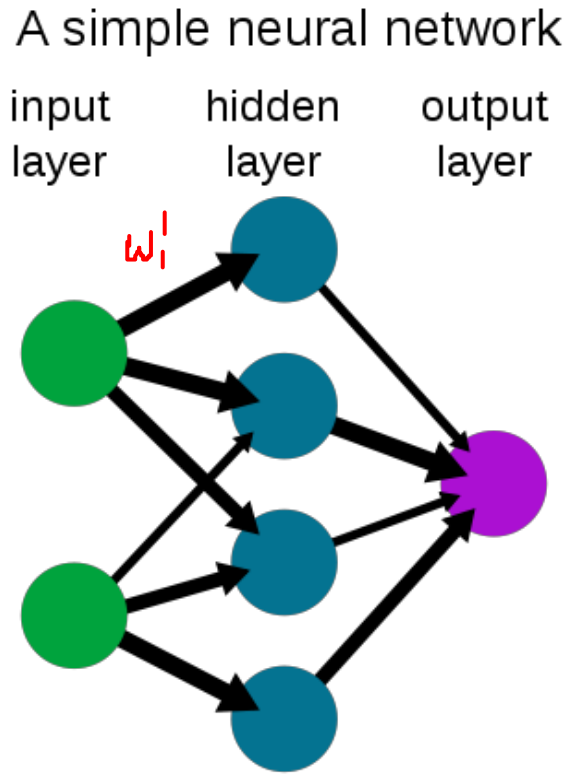
Gradient descent for 1-hidden layer – Back-propagation: Computing $\frac{\partial \ell(W)}{\partial w_i^k}$

$$\ell(W) = \frac{1}{2}\sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g\left(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})\right)$$

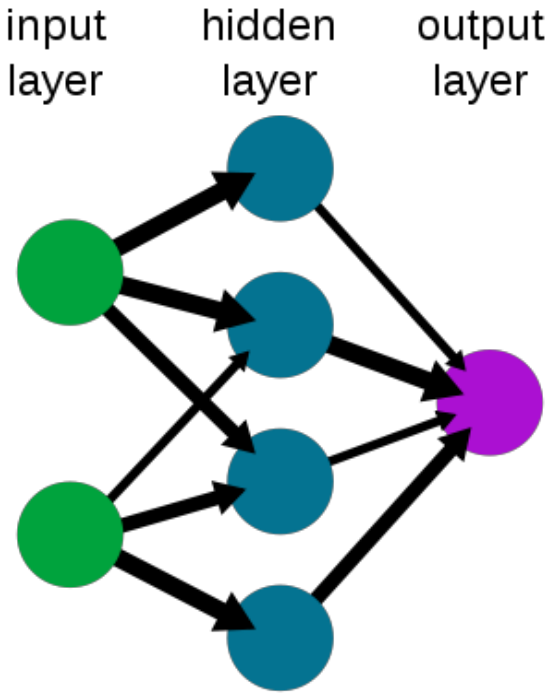
$$\frac{\partial \ell(W)}{\partial w_i^k} = \sum_{j=1}^{m} -[y - out(\mathbf{x}^j)]\frac{\partial out(\mathbf{x}^j)}{\partial w_i^k}$$

Dropped w_0 to make derivation simpler



Back-propagation Algorithm

A simple neural network



$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k g(w_0^k + \sum_i w_i^k x_i)\right)$$

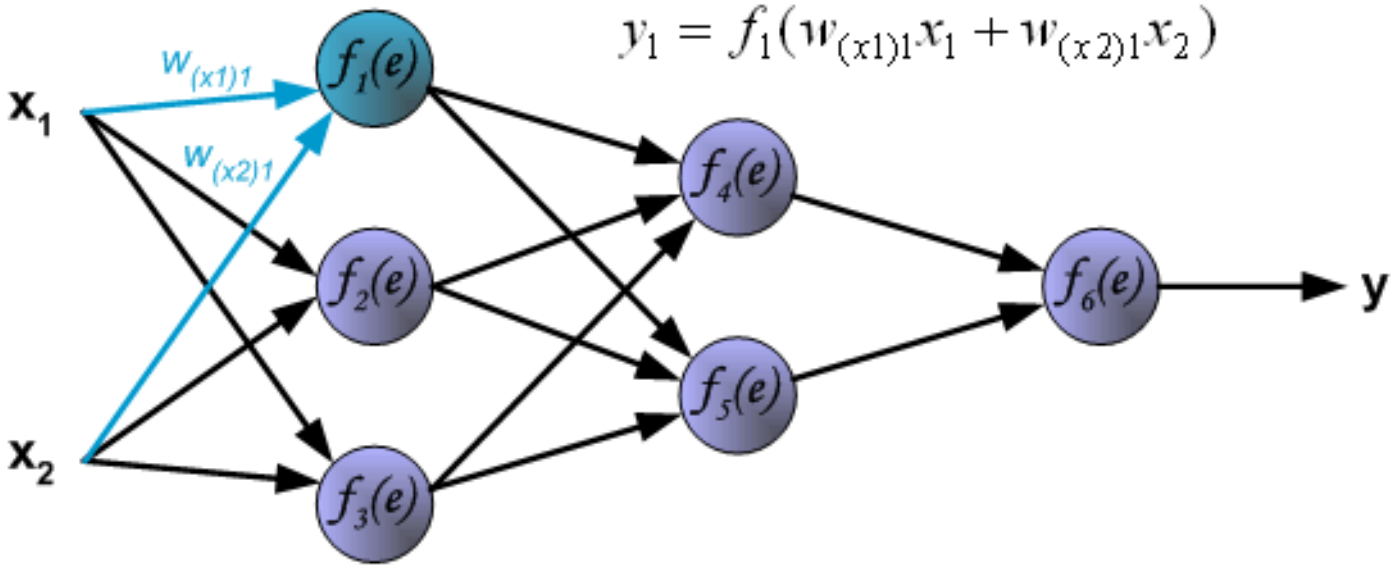
Sigmoid function: $g(z) = \frac{1}{1 + exp(-z)}$

(Rummelhart et al. 1986)

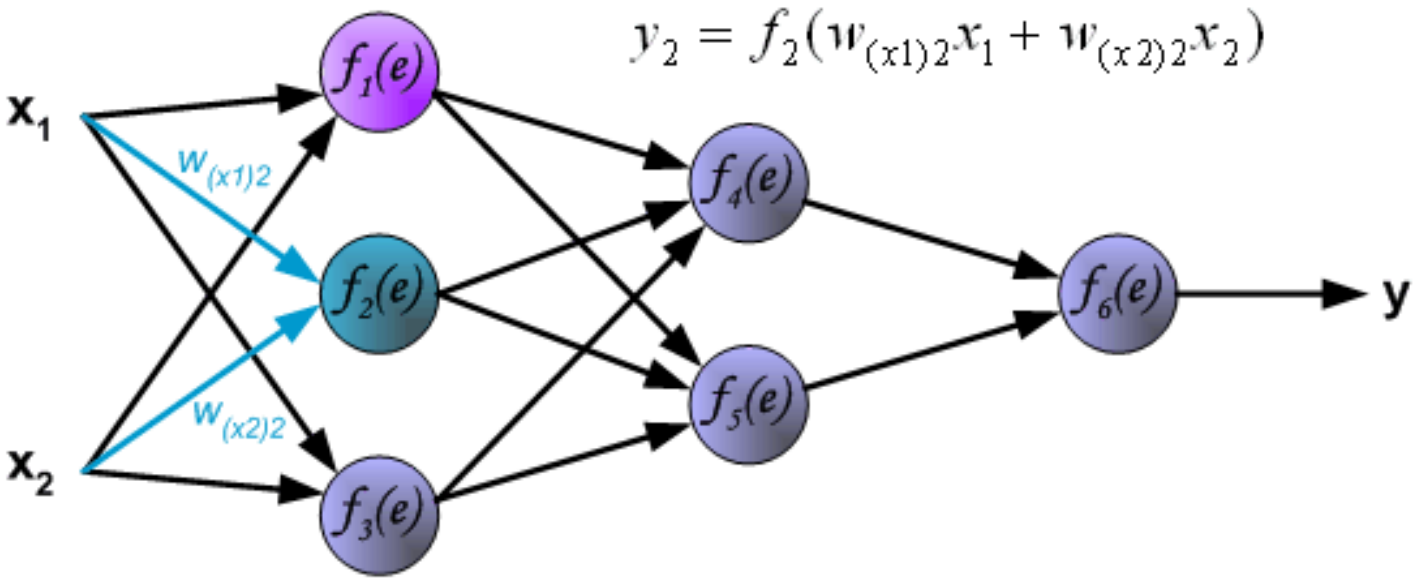
Non-convex function of '**w**'s

Back-prop: find local optima

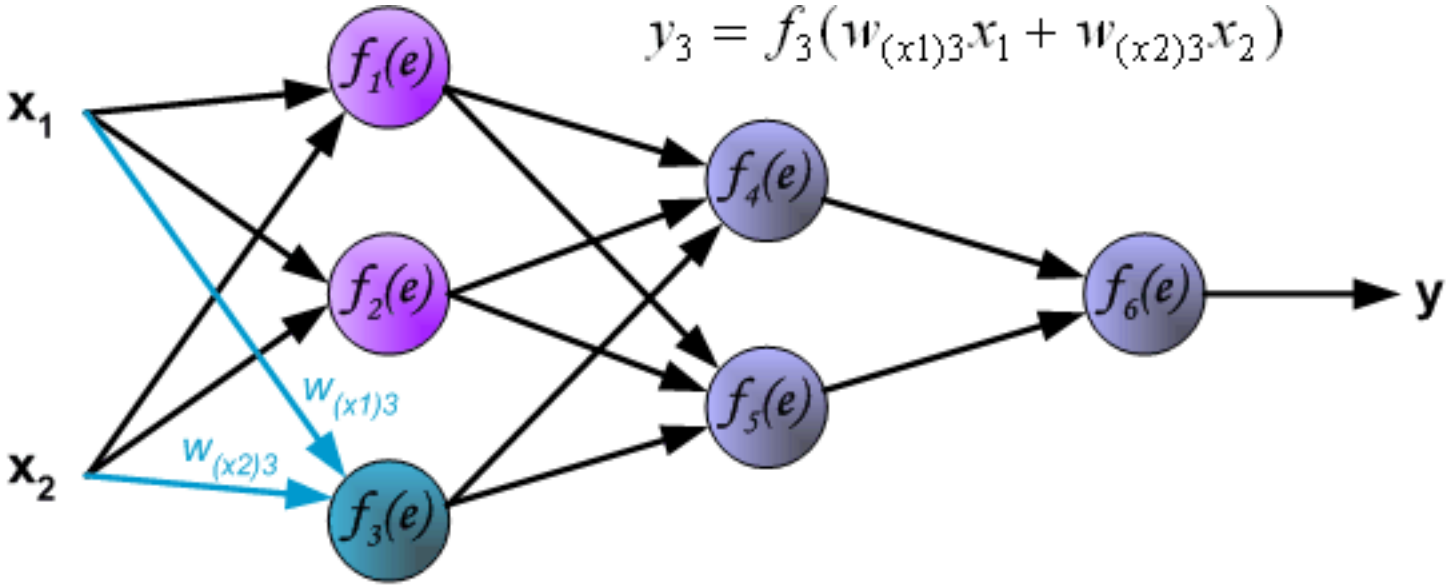
Learning Algorithm: Backpropagation



Learning Algorithm: Backpropagation

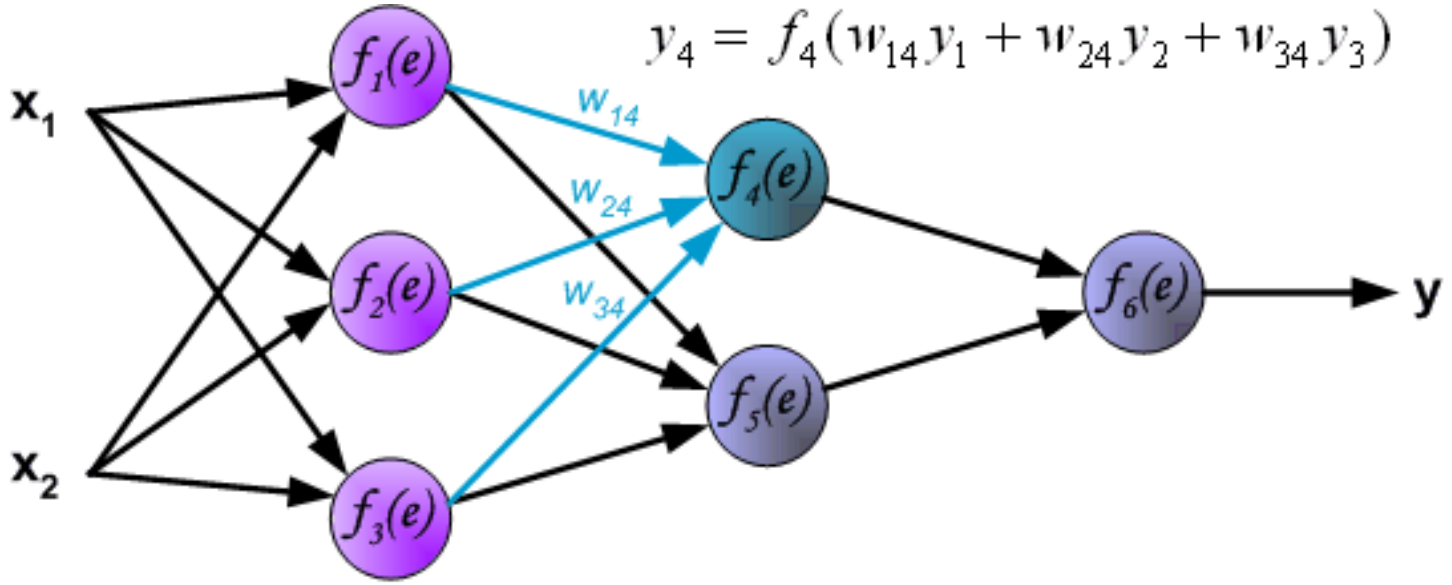


Learning Algorithm: Backpropagation

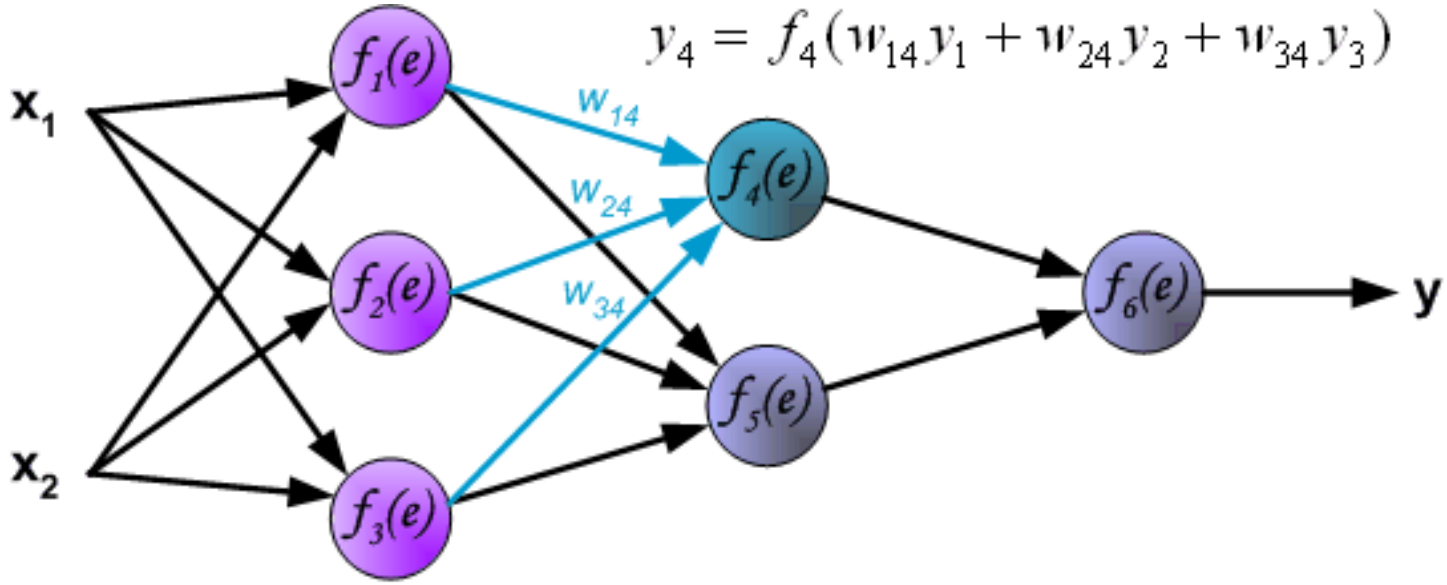


Learning Algorithm: Backpropagation

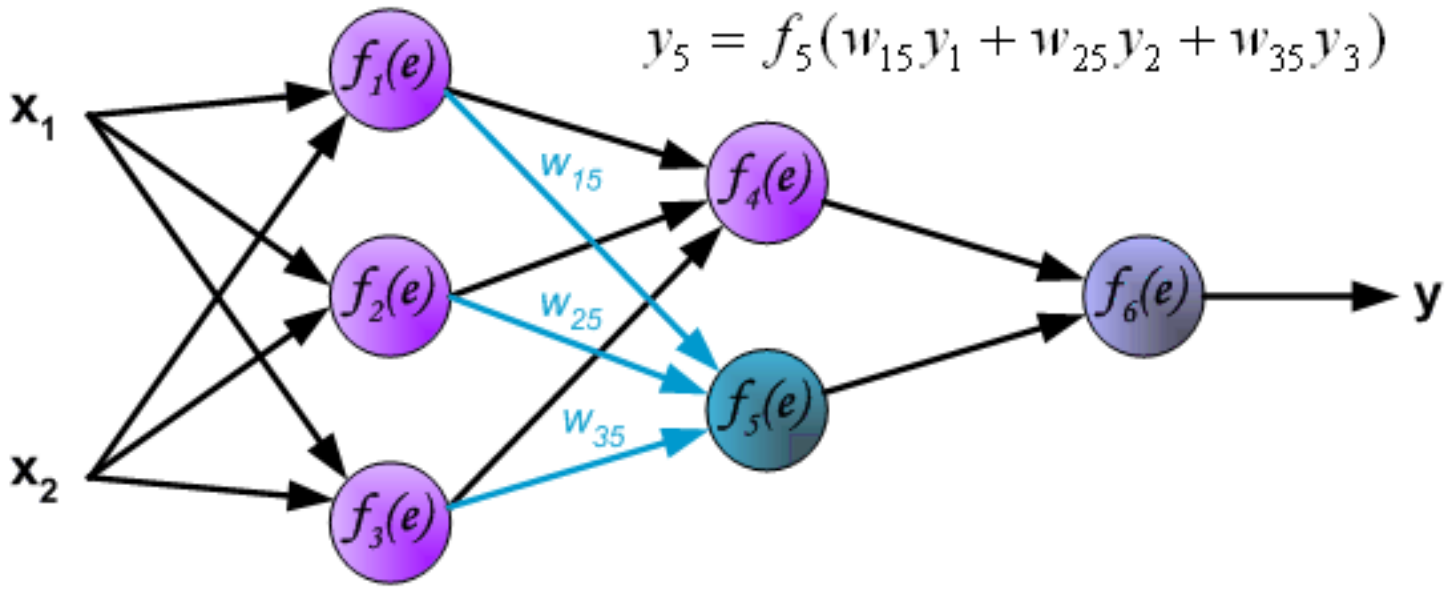
Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron *m* and input of neuron *n* in the next layer.



Learning Algorithm: Backpropagation

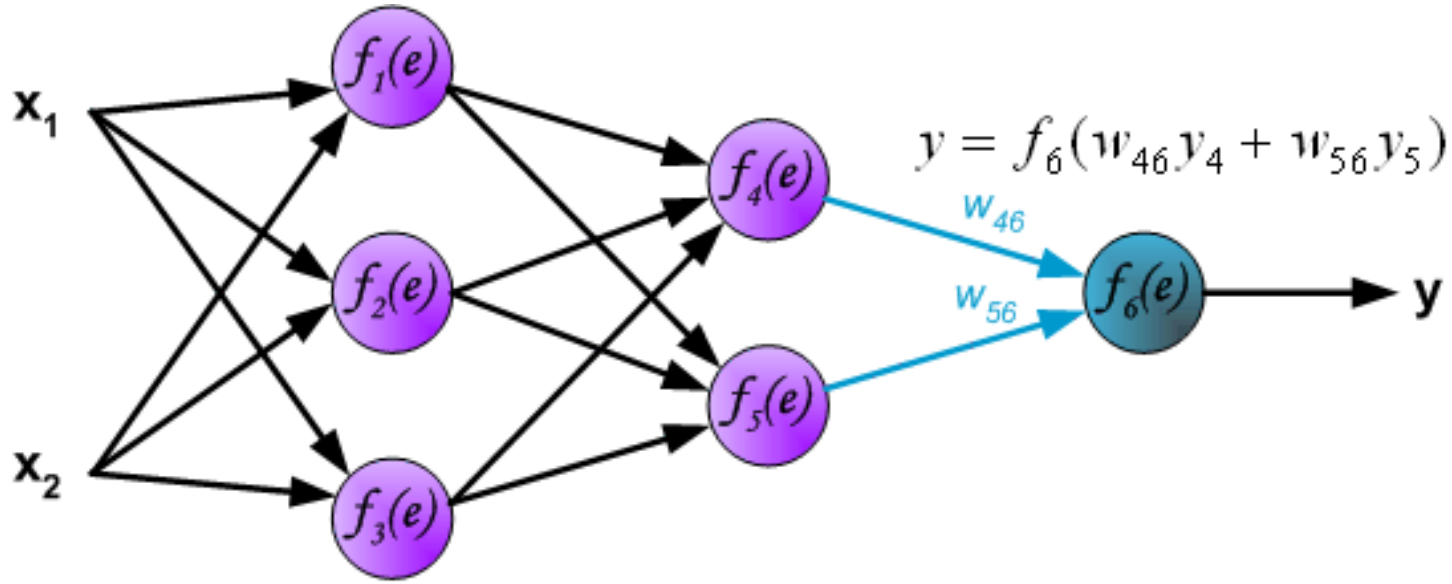


Learning Algorithm: Backpropagation



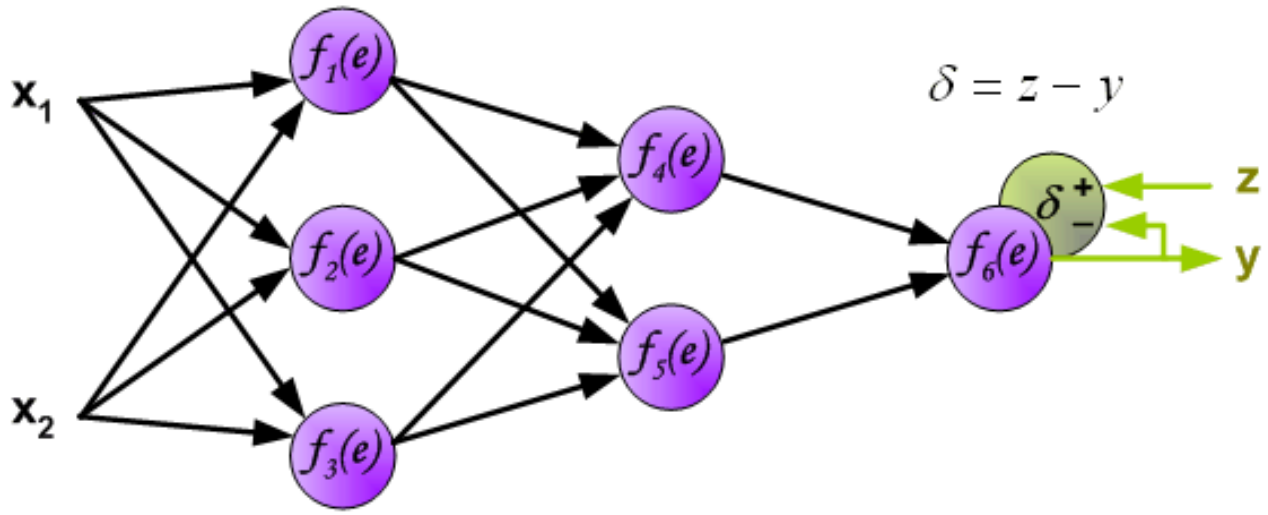
Learning Algorithm: Backpropagation

Propagation of signals through the output layer.



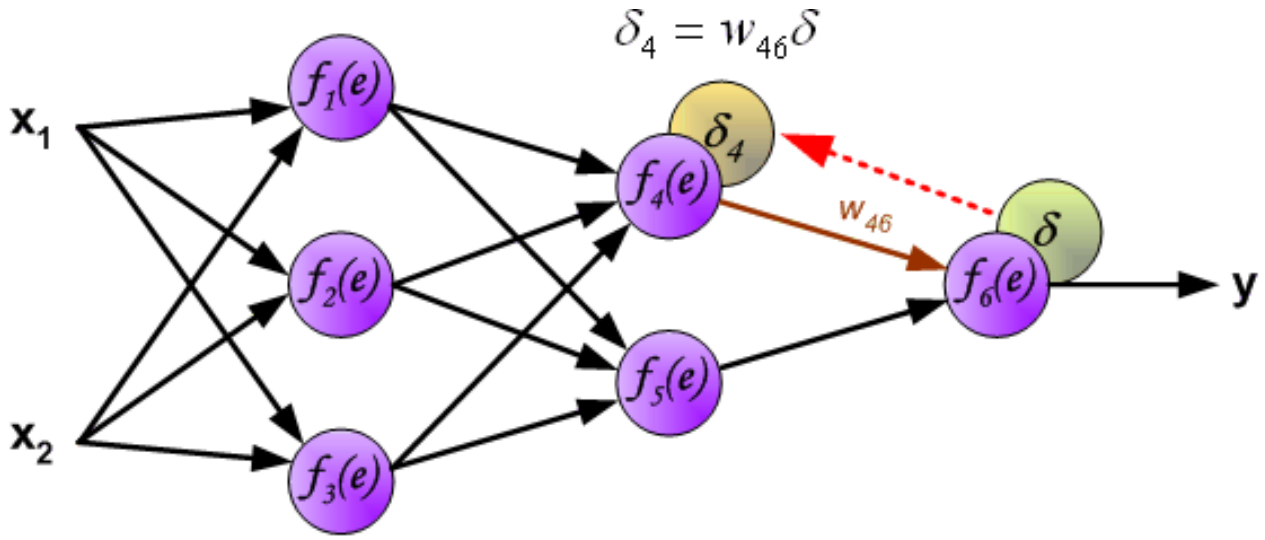
Learning Algorithm: Backpropagation

In the next algorithm step the output signal of the network *y* is compared with the desired output value (the target), which is found in training data set. The difference is called error signal *d* of output layer neuron



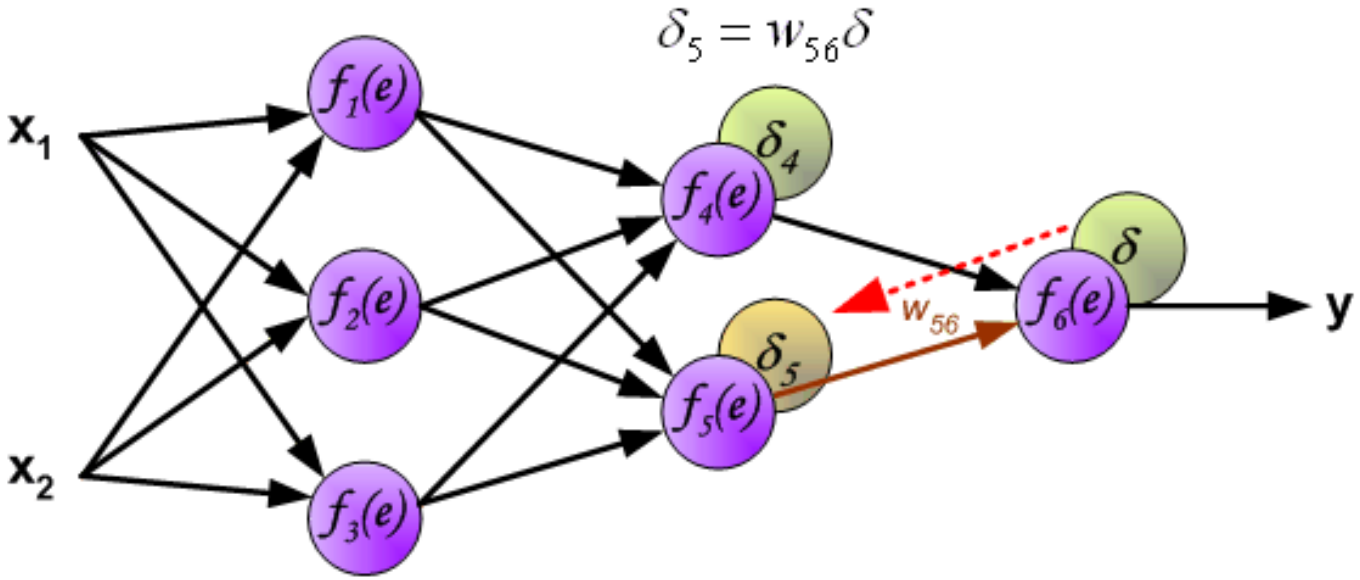
Learning Algorithm: Backpropagation

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



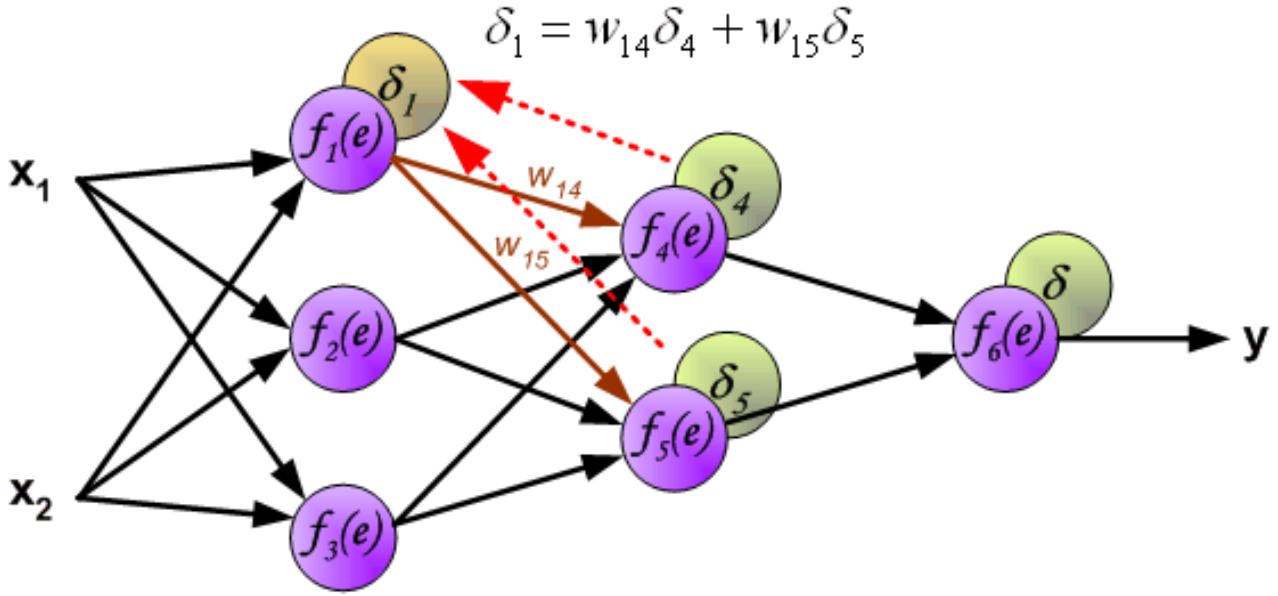
Learning Algorithm: Backpropagation

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



Learning Algorithm: Backpropagation

The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:

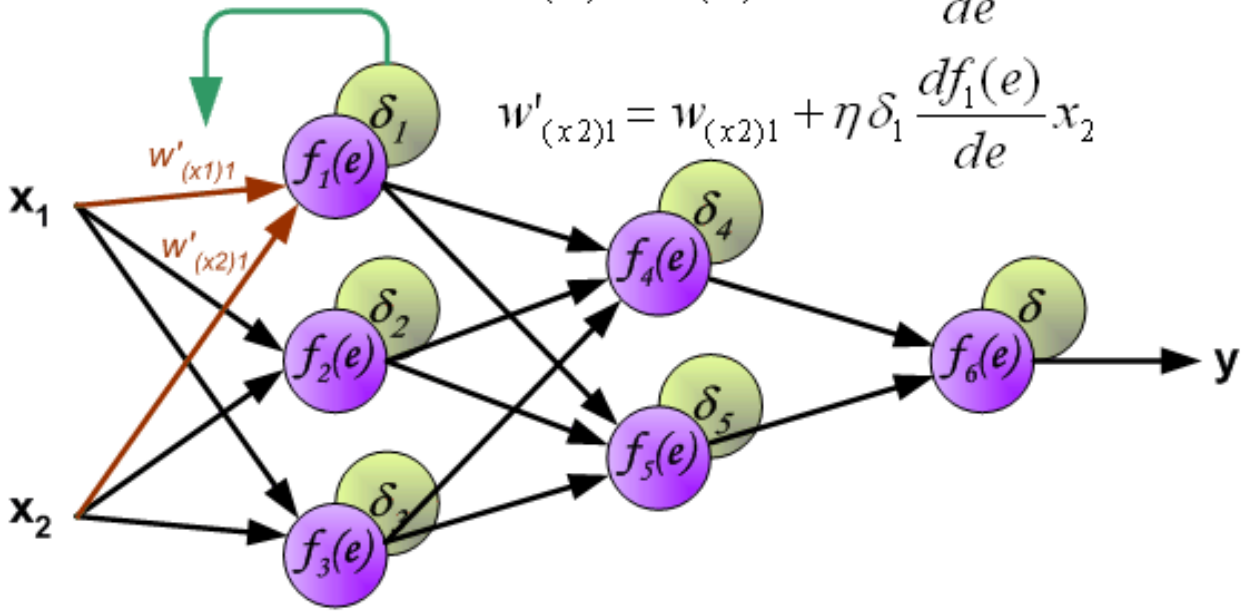


Learning Algorithm: Backpropagation

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified)

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



Convergence of backprop

Perceptron leads to convex optimization

- Gradient descent reaches **global minima**

Multilayer neural nets **not convex**

- Gradient descent could get stuck in local minima
- Hard to set learning rate
- Selecting number of hidden units and layers = fuzzy process
- Nonetheless, neural nets are one of the most used ML approaches

Definitions

For each neuron (j), the output of that neuron is

$$o_j = g(\sum_{k=1}^{n} w_{kj} o_k)$$

For the first layer of neurons, o_k is just the input variables x_k

Definitions

For each neuron (j), the output of that neuron is

$$o_j = g(\sum_{k=1}^{n} w_{kj} o_k)$$

To find the correct direction for our gradient descent, we need to find the partial derivative of the error with respect to each weight.

- This will depend on our activation function
- By selecting the sigmoid as our activation function g(x), we get the following derivative

$$\frac{dg(x)}{dx} = g(x)(1 - g(x))$$

Finding the direction of descent

Then, to find how each weight impacts the final error, given by the squared error function for each sample:

$$E = \frac{1}{2}(y - f(x)^2)$$

We then need to find $\frac{\partial E}{\partial w_{ij}}$ in order to determine the direction of descent. We can a calculate this by:

$$\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

Finding the direction of descent

What are each of these terms?

$$\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

Finding the direction of descent

What are each of these terms?

$$\frac{\partial E}{\partial o_j} = \sum_{\ell \in L} \left(\frac{\partial E}{\partial \text{net}_\ell} \frac{\partial \text{net}_\ell}{\partial o_j} \right) = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \frac{\partial o_\ell}{\partial \text{net}_\ell} w_{j\ell} \right)$$

$$\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

Finding the direction of descent

What are each of these terms?

$$\frac{\partial E}{\partial o_j} = \sum_{\ell \in L} \left(\frac{\partial E}{\partial \text{net}_\ell} \frac{\partial \text{net}_\ell}{\partial o_j} \right) = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \frac{\partial o_\ell}{\partial \text{net}_\ell} w_{j\ell} \right)$$

$$\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

This is for the arbitrary neuron *j*, yet it depends on other neurons *l*
This is the back propagation! Think about the error for the output neuron

Finding the direction of descent

What are each of these terms?

$$\frac{\partial E}{\partial o_j} = \sum_{\ell \in L} \left(\frac{\partial E}{\partial \text{net}_\ell} \frac{\partial \text{net}_\ell}{\partial o_j} \right) = \sum_{\ell \in L} \left(\frac{\partial E}{\partial o_\ell} \frac{\partial o_\ell}{\partial \text{net}_\ell} w_{j\ell} \right)$$

$$\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

This is for the arbitrary neuron *j*, yet it depends on other neurons *l*
This is the back propagation! Think about the error for the output neuron

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2} (t - y)^2 = y - t$$

Finding the direction of descent

What are each of these terms?

$$\frac{\partial o_j}{\partial net_j} =$$

$$\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

Finding the direction of descent

What are each of these terms?

$$\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j} g(net_j) = g(net_j)(1 - g(net_j))$$

When g(x) is the sigmoid (a big advantage of using the sigmoid!)

Finding the direction of descent

What are each of these terms?

$$\frac{\partial \text{net}_j}{\partial w_{ij}} =$$

$$\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

Finding the direction of descent

What are each of these terms?

$$\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

$$\frac{\partial \mathrm{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj} o_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i.$$

A nice and straightforward derivative here

Bringing it all together, we get

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i \qquad \delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j}$$

,

Finding the direction of descent

What are each of these terms?

$$\frac{\partial \text{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=1}^{n} w_{kj} o_k \right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i .$$

$$\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$net_j = \sum_{k=1}^{n} w_{kj} o_k$$

A nice and straightforward derivative here

Bringing it all together, we get

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i \qquad \delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} (o_j - t_j) o_j (1 - o_j) & \text{if } j \text{ is an output neuron,} \\ (\sum_{\ell \in L} w_{j\ell} \delta_\ell) o_j (1 - o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

Finding the direction of descent

What should the final change in our weights be then?

Finding the direction of descent

What should the final change in our weights be then?

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

Finding the direction of descent

What should the final change in our weights be then?

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

Where:

- η : learning rate
- δ_j : the back-propogation factor for neuron j
- o_i : the output of the neuron *i* (which is input into j with weight w_{ij})

Back propagation

This is considered one learning iteration: we adjust all of our weights in one iteration

What are some strengths?

Back propagation

This is considered one learning iteration: we adjust all of our weights in one iteration

What are some strengths?

- Sigmoid gives us some advantages as far as differentials
- We adjust based on new feedback
  - We can actually adjust partially based on new points (no need to retrain the whole network with new data)
- With random start weights, hopefully each neuron finds its own novel trend in the data
-

Back propagation

This is considered one learning iteration: we adjust all of our weights in one iteration

What are some strengths?

- Sigmoid gives us some advantages as far as differentials
- We adjust based on new feedback
  - We can actually adjust partially based on new points (no need to retrain the whole network with new data)
- With random start weights, hopefully each neuron finds its own novel trend in the data

Weaknesses?

-

Back propagation

This is considered one learning iteration: we adjust all of our weights in one iteration

What are some strengths?

- Sigmoid gives us some advantages as far as differentials
- We adjust based on new feedback
  - We can actually adjust partially based on new points (no need to retrain the whole network with new data)
- With random start weights, hopefully each neuron finds its own novel trend in the data

Weaknesses?

- Gradient descent finds local minima
  - May want to train several neural networks with random start weights
  - Keep the best (or some k best)
- Has difficulty with plateaus (regions where there is little change in the output)
-