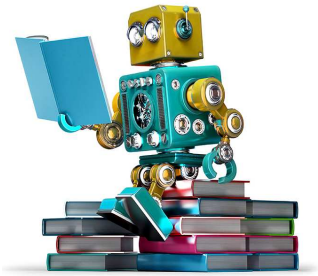
Module 3- Outline

MACHINE LEARNING

Artificial Neural Network

1. Biological Motivation
2. Neural Network Representation
3. Appropriate Problems for NN learning
4. Perceptions
5. **Multilayer Networks and Backpropagation Algorithm**
6. Remarks on Backpropagation Algorithm
7. Summary



Multilayer Networks (ANN)

- capable of learning **nonlinear decision surfaces**
- normally **directed** and **acyclic** ⇒ Feed-forward Network
- based on **sigmoid unit**
  - much like a perceptron
  - but based on a smoothed, **differentiable threshold function**

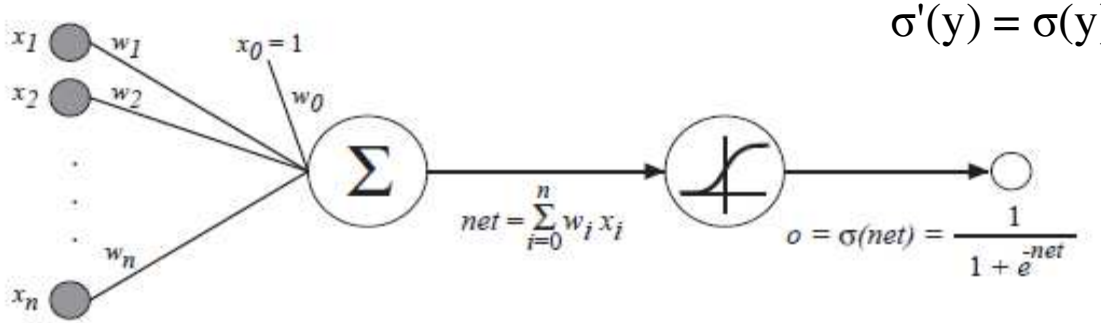
$$\sigma(net) = \frac{1}{1+e^{-net}}$$

$$\lim_{net \to +\infty} \sigma(net) = 1$$

$$\lim_{net \to -\infty} \sigma(net) = 0$$

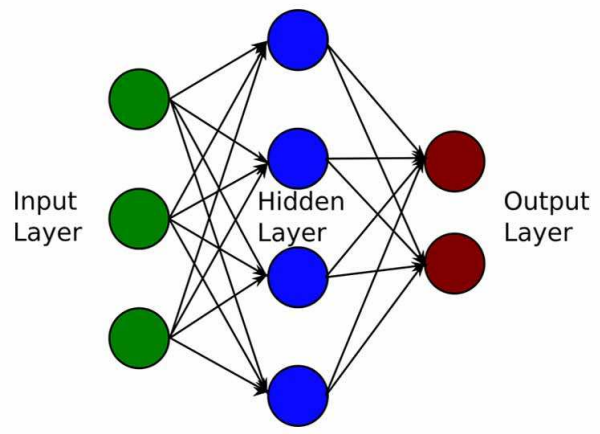
$$o = \sigma(\vec{w} \cdot \vec{x}) \quad \sigma(y) = \frac{1}{1+e^{-y}}$$

$$\sigma'(y) = \sigma(y)\,(1 - \sigma(y))$$



BACKPROPOGATION

- learns weights for a feed-forward multilayer network with a fixed set of neurons and interconnections
- employs gradient descent to minimize error
- redefinition of E
  - has to sum the errors over all units
  - $E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$
- **problem:** search through a large H defined over all possible weight values for all units in the network



$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

One major difference in the case of multilayer networks is - the error surface can have multiple local minima

Despite this obstacle, in practice Backpropagation Algorithm been found to produce excellent results in many real-world applications.

BACKPROPOGATION Algorithm

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

The input from unit i to unit j is denoted x_{ji} and the weight from unit i to unit j is denoted w_{ji}.

- create a feed-forward network with n_{in} inputs. n_{hidden} hidden untis, and n_{out} output units
- Initialize all network weights to small random numbers
- Until the **termination condition** is met, Do
  - For each $< \vec{x}, \vec{t} >$ in $training\_examples$, Do

    Propagate the input forward through the network:

    1. Input $\vec{x}$ to the network and compute o_u of every unit u

    Propagate the errors back trough the network:

    2. For each network **output unit** k, calculate its error term δ_k

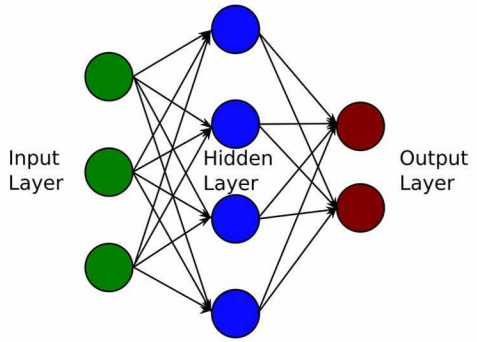
       $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$

    3. For each **hidden unit** h, calculate its error term δ_h

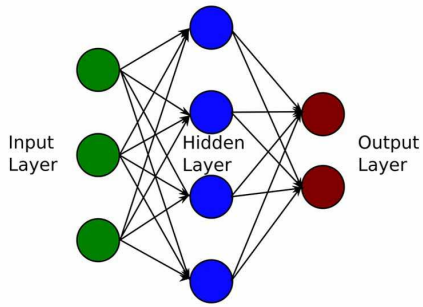
       $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$

    4. Update each weight w_{ji}

       $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$ where $\Delta w_{ji} = \eta \delta_j x_{ji}$



- This algorithm applies to feedforward networks
  - containing **two layers** of **sigmoid units**,
  - with units at each layer connected to all units from the preceding layer.



- This is the **incremental** or **stochastic**, gradient descent version of Backpropagation.
- The notation used here is the same as that used in earlier sections, with the following extensions:
  - An index (e.g., an integer) is assigned to each node in the network, where a "node" is either an input to the network or the output of some unit in the network.
  - x_{ji} denotes the input from node i to unit j, and w_{ji} denotes the corresponding weight.
  - δ_n denotes the error term associated with unit n. It plays a role analogous to the quantity $(t - o)$ in our earlier discussion of the delta training rule. As we shall see later, $\delta_n = -\frac{\partial E}{\partial net_n}$.

Termination Conditions

- fixed number of iterations
- error falls below some threshold
- error on a separate validation set falls below some threshold
- **important:**
  - too few iterations reduce error insufficiently
  - too many iterations can lead to overfitting the data

Adding Momentum

- one way to avoid local minima in the error surface or flat regions
- make the weight update in the n^{th} iteration depend on the update in the $(n-1)^{th}$ iteration

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1)$$

$$0 \leq \alpha \leq 1$$

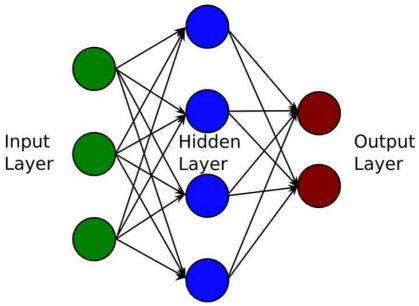
δ computation

- **δ_k** is simply the familiar **$(t_k - o_k)$** from the delta rule, multiplied by the factor **$o_k(1 - o_k)$,** which is the derivative of the sigmoid squashing function.
- However, since training examples provide target values t_k only for network outputs,
  - no target values are directly available to indicate the error of hidden units values.
- Instead, the error term for hidden unit h is calculated by
  - summing the error terms δ_k for each output unit influenced by h,
  - weighting each of the δ_k's by w_{kh}, the weight from hidden unit h to output unit k.
  - This weight characterizes the degree to which hidden unit h is "responsible for" the error in output unit k.

$$\delta_r = o_r (1 - o_r) \sum_{s \in layer\ m+1} w_{sr}\, \delta_s$$

Derivation of Backpropagation rule

- We have $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$ where $E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$



- x_{ji} = the ith input to unit j
- w_{ji} = the weight associated with the ith input to unit j
- $net_j = \sum_i w_{ji} x_{ji}$ (the weighted sum of inputs for unit j)
- o_j = the output computed by unit j
- t_j = the target output for unit j
- σ = the sigmoid function
- $outputs$ = 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

Derivation of Backpropagation rule

MACHINE LEARNING

- Let us derive the expression for stochastic gradient descent rule.

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$= \frac{\partial E_d}{\partial net_j} x_{ji}$$

- **Case 1: Training Rule for Output Unit Weights.**

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j} \qquad (4.23)$$

To begin, consider just the first term in Equation (4.23)

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

The derivatives $\frac{\partial}{\partial o_j}(t_k - o_k)^2$ will be zero for all output units k except when $k = j$. We therefore drop the summation over output units and simply set $k = j$.

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2 \qquad \frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

$$= \frac{1}{2}2(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j}$$

$$= -(t_j - o_j) \tag{4.24}$$

Next consider the second term in Equation (4.23). Since $o_j = \sigma(net_j)$, the derivative $\frac{\partial o_j}{\partial net_j}$ is just the derivative of the sigmoid function, which we have already noted is equal to $\sigma(net_j)(1 - \sigma(net_j))$. Therefore,

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$

$$= o_j(1 - o_j) \tag{4.25}$$

Substituting expressions (4.24) and (4.25) into (4.23), we obtain

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\ o_j(1 - o_j) \tag{4.26}$$

and combining this with Equations (4.21) and (4.22), we have the stochastic gradient descent rule for output units

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta\ (t_j - o_j)\ o_j(1 - o_j) x_{ji} \tag{4.27}$$

Case 2: Training Rule for Hidden Unit Weights.

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

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

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

Rearranging terms and using δ_j to denote $-\frac{\partial E_d}{\partial net_j}$, we have

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \, w_{kj}$$

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$

BACKPROPOGATION Algorithm

MACHINE LEARNING

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

The input from unit i to unit j is denoted x_{ji} and the weight from unit i to unit j is denoted w_{ji}.

- create a feed-forward network with n_{in} inputs. n_{hidden} hidden untis, and n_{out} output units
- Initialize all network weights to small random numbers
- Until the **termination condition** is met, Do
  - For each $< \vec{x}, \vec{t} >$ in $training\_examples$, Do

    Propagate the input forward through the network:

    1. Input $\vec{x}$ to the network and compute o_u of every unit u

    Propagate the errors back trough the network:

    2. For each network **output unit** k, calculate its error term δ_k

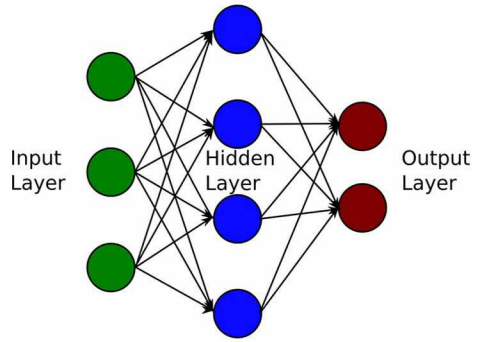
       $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$

    3. For each **hidden unit** h, calculate its error term δ_h

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

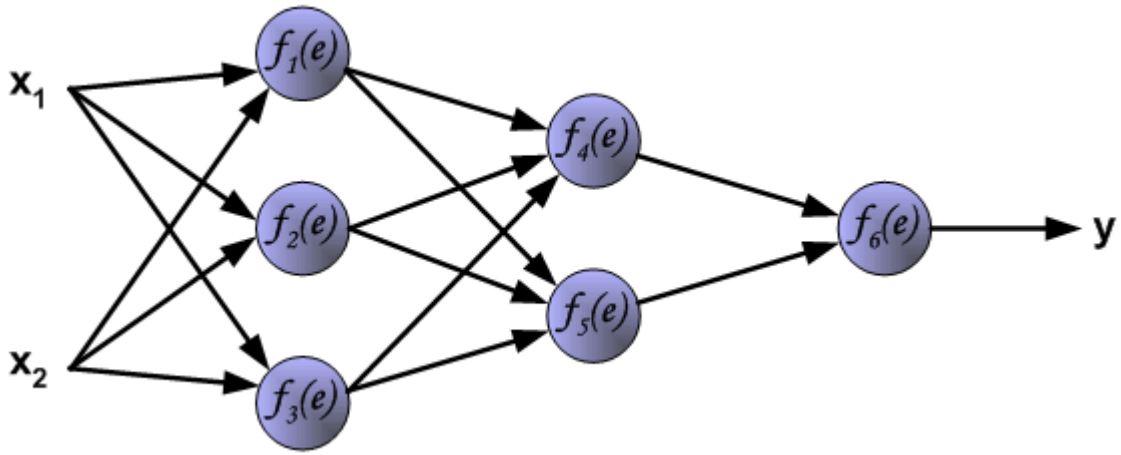
    4. Update each weight w_{ji}

       $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$ where $\Delta w_{ji} = \eta \delta_j x_{ji}$



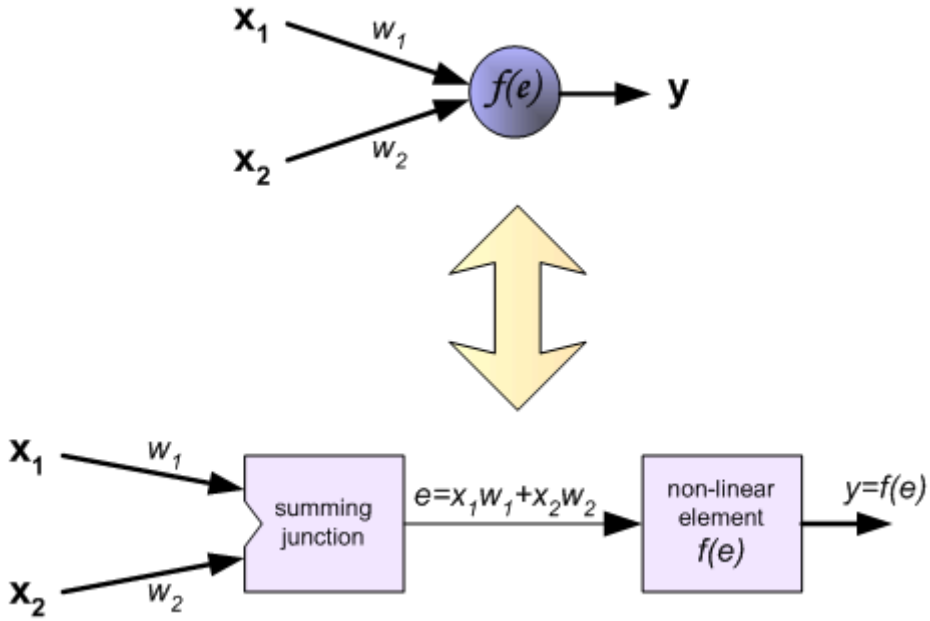
Learning Algorithm: Backpropagation

The following slides describes **teaching process** of multi-layer neural network employing **backpropagation** algorithm. To illustrate this process the three layer neural network with two inputs and one output,which is shown in the picture below, is used:



Learning Algorithm: Backpropagation

Each neuron is composed of two units. First unit adds products of weights coefficients and input signals. The second unit realise nonlinear function, called neuron transfer (activation) function. Signal *e* is adder output signal, and *y = f(e)* is output signal of nonlinear element. Signal *y* is also output signal of neuron.



Learning Algorithm: Backpropagation

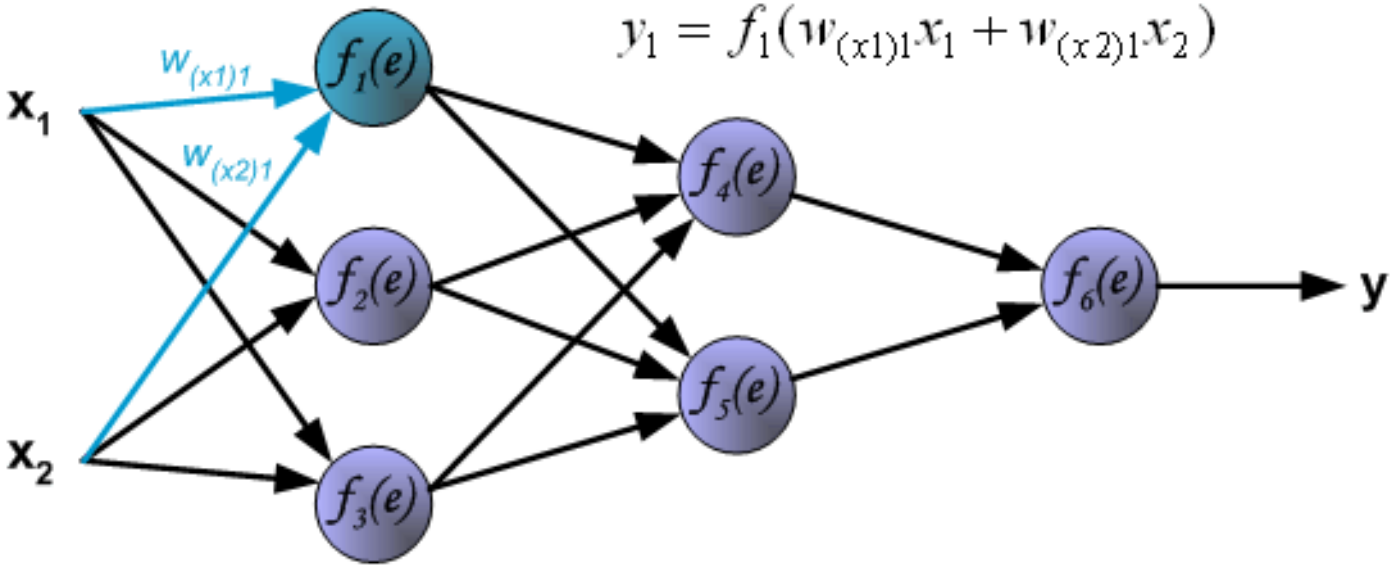
- To teach the neural network we need training data set. The training data set consists of input signals (x_1 and x_2) assigned with corresponding target (desired output) z.

- The network training is an iterative process. In each iteration weights coefficients of nodes are modified using new data from training data set. Modification is calculated using algorithm described below:

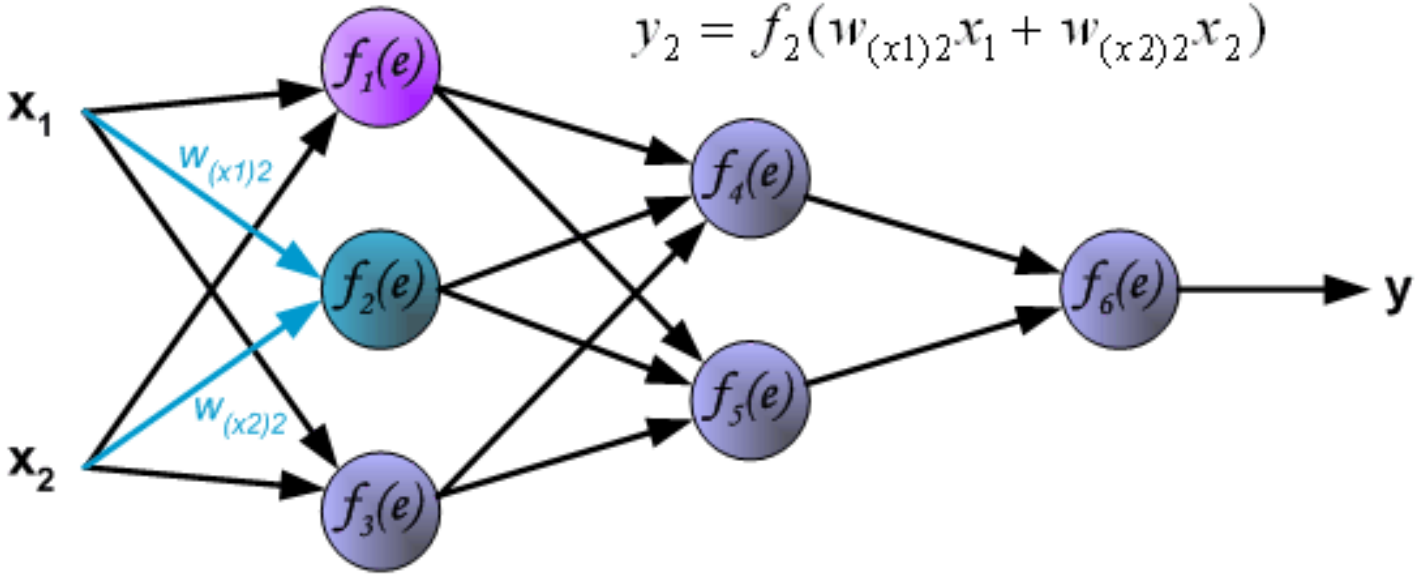
- Each teaching step starts with forcing both input signals from training set. After this stage we can determine output signals values for each neuron in each network layer.

Learning Algorithm: Backpropagation

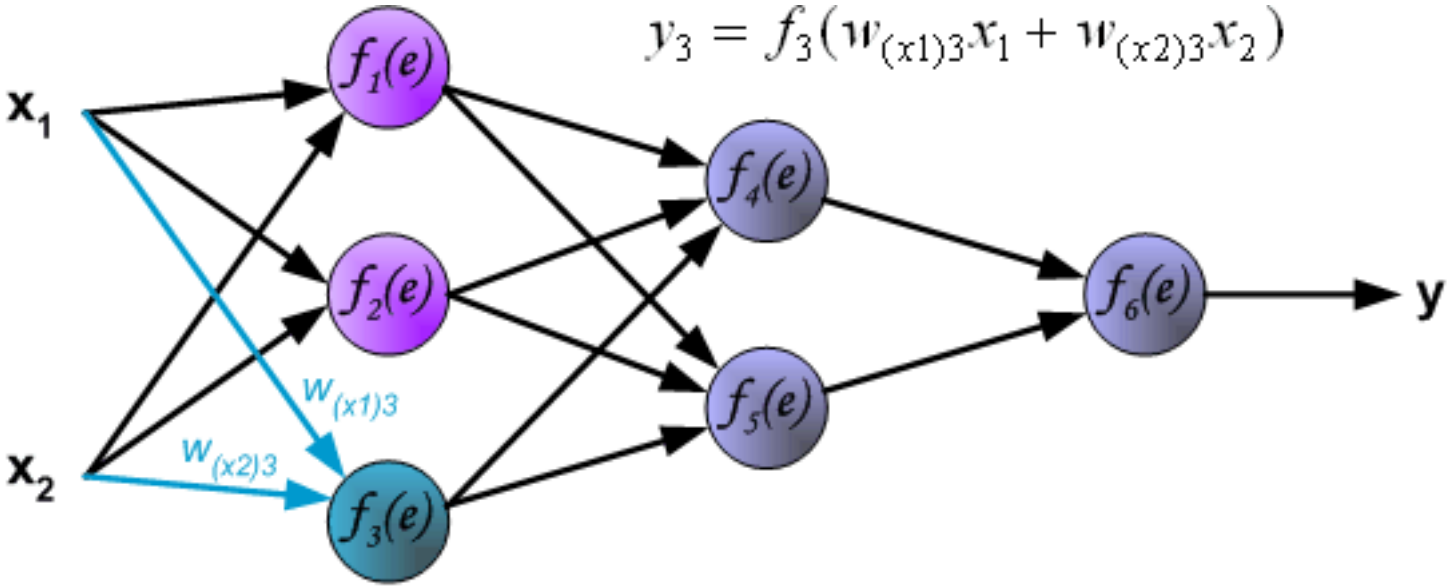
Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.



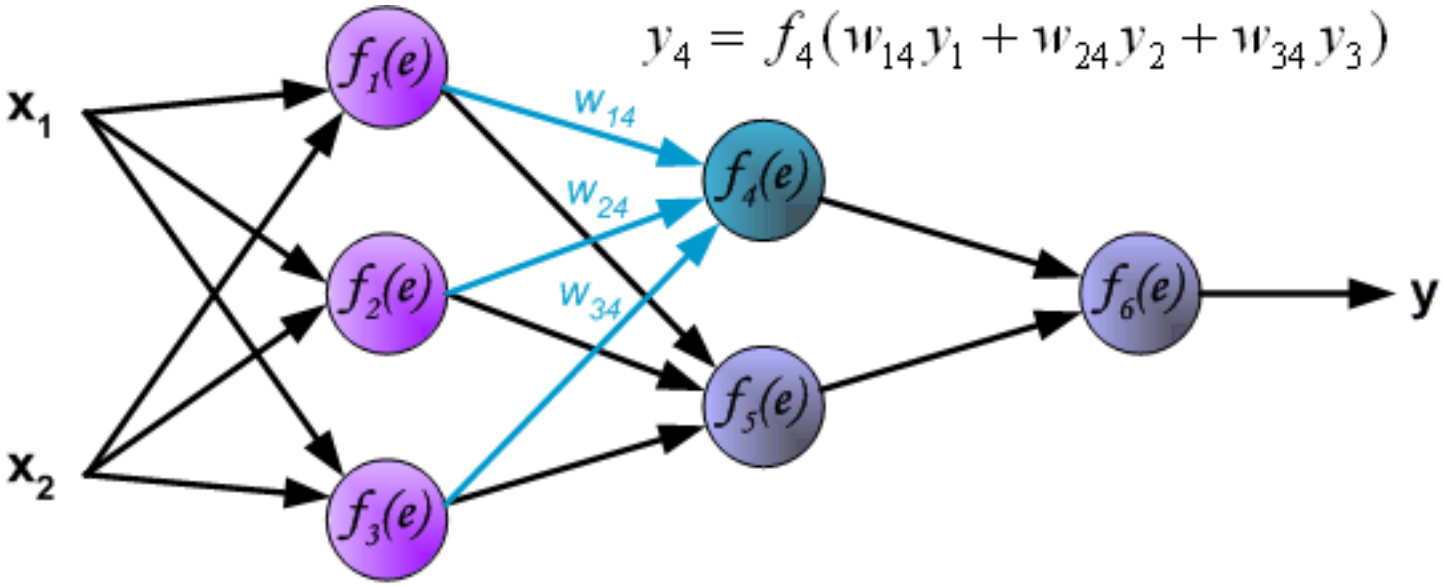
Learning Algorithm: Backpropagation



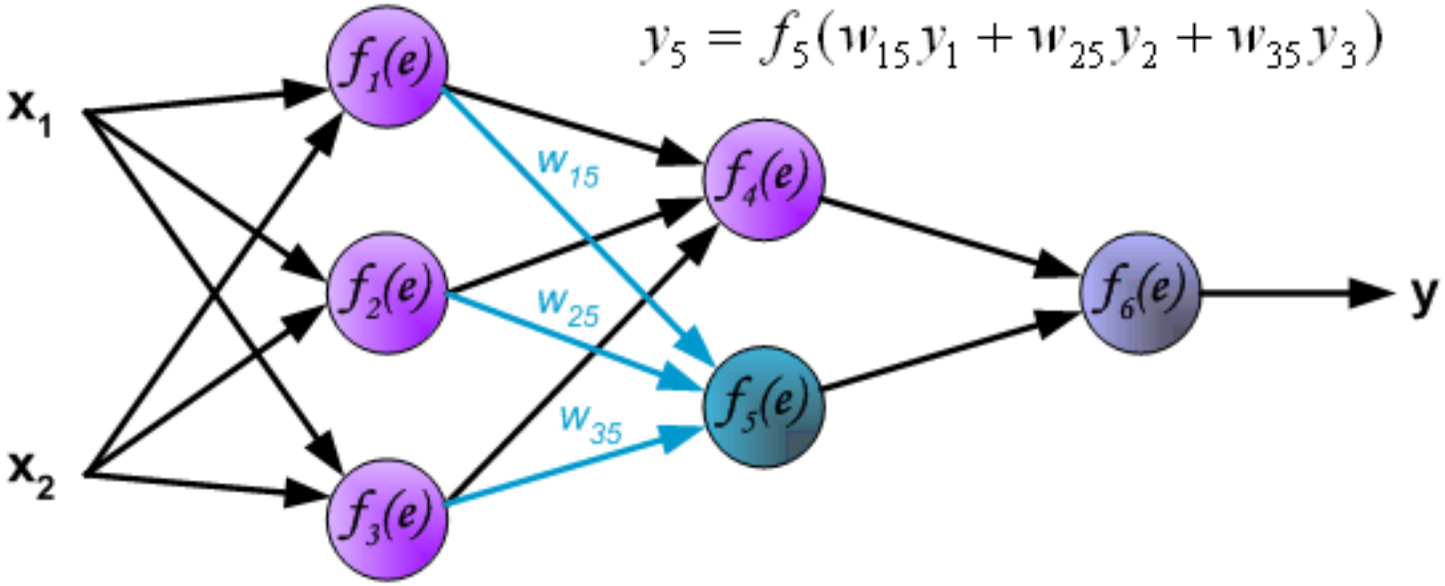
Learning Algorithm: Backpropagation



Learning Algorithm: Backpropagation



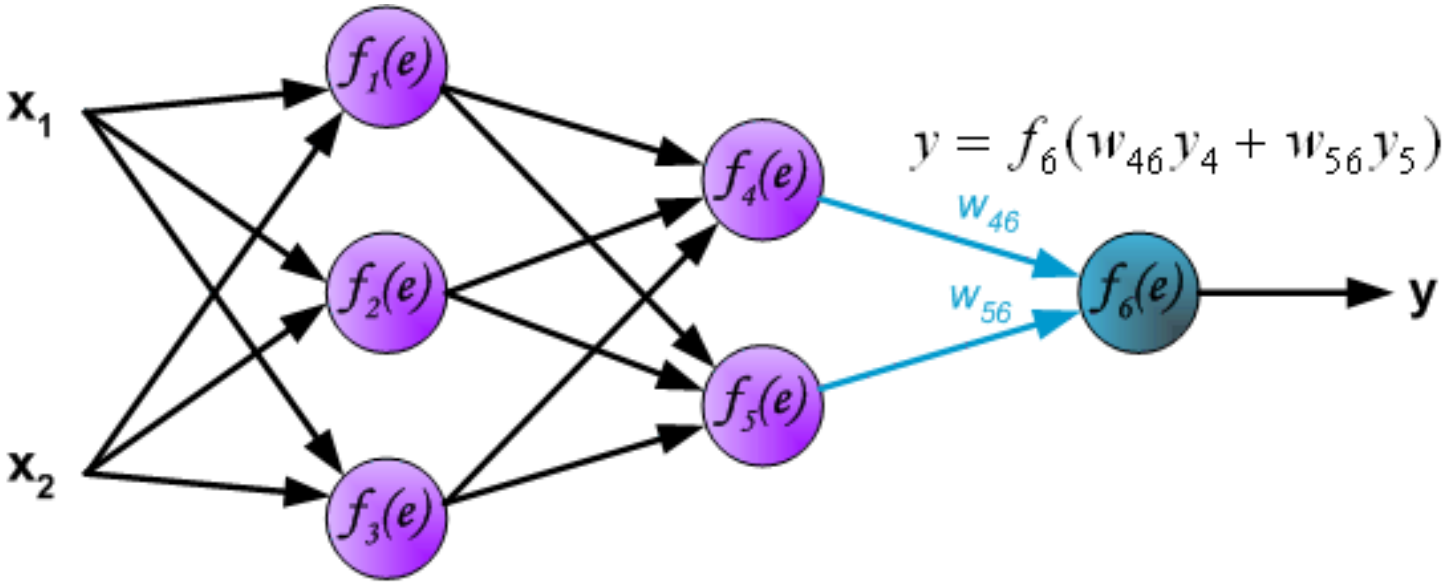
Learning Algorithm: Backpropagation



Learning Algorithm: Backpropagation

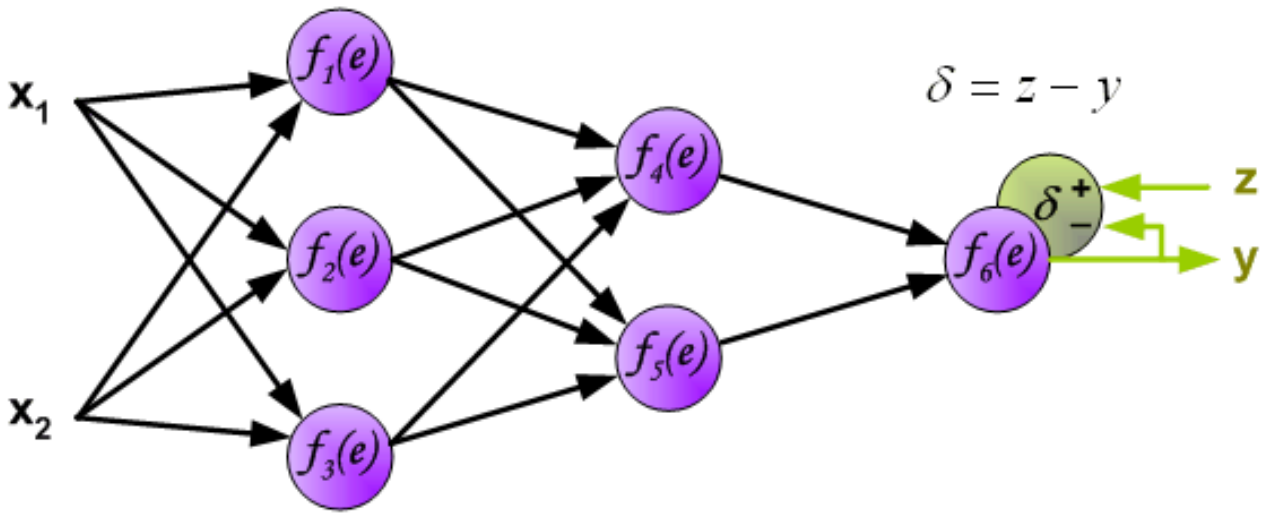
MACHINE LEARNING

Propagation of signals through the output layer.



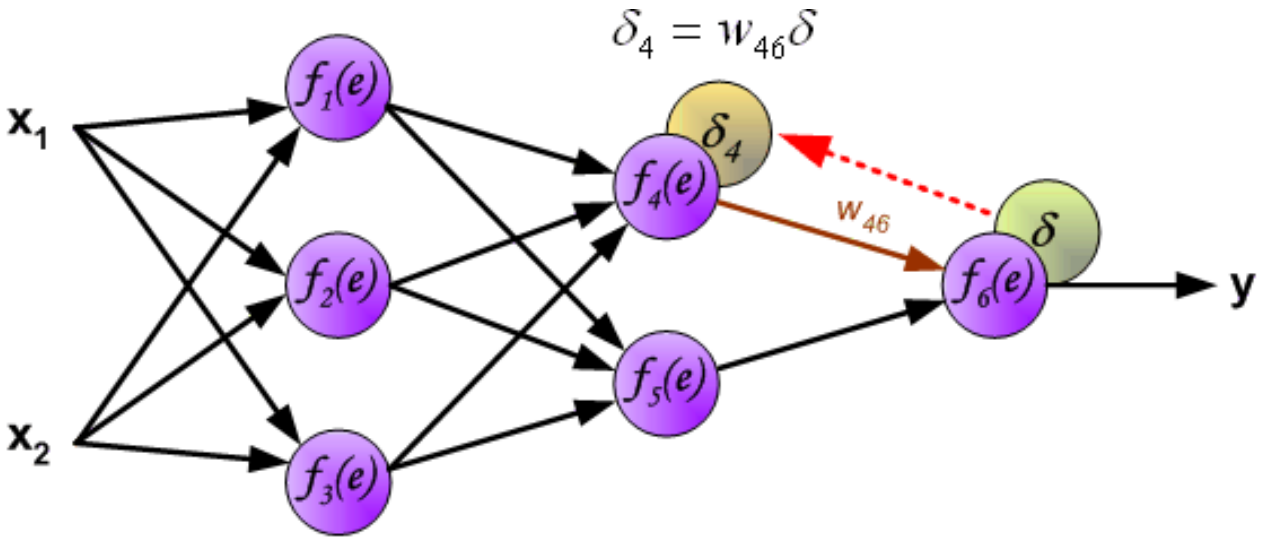
Learning Algorithm: Backpropagation

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



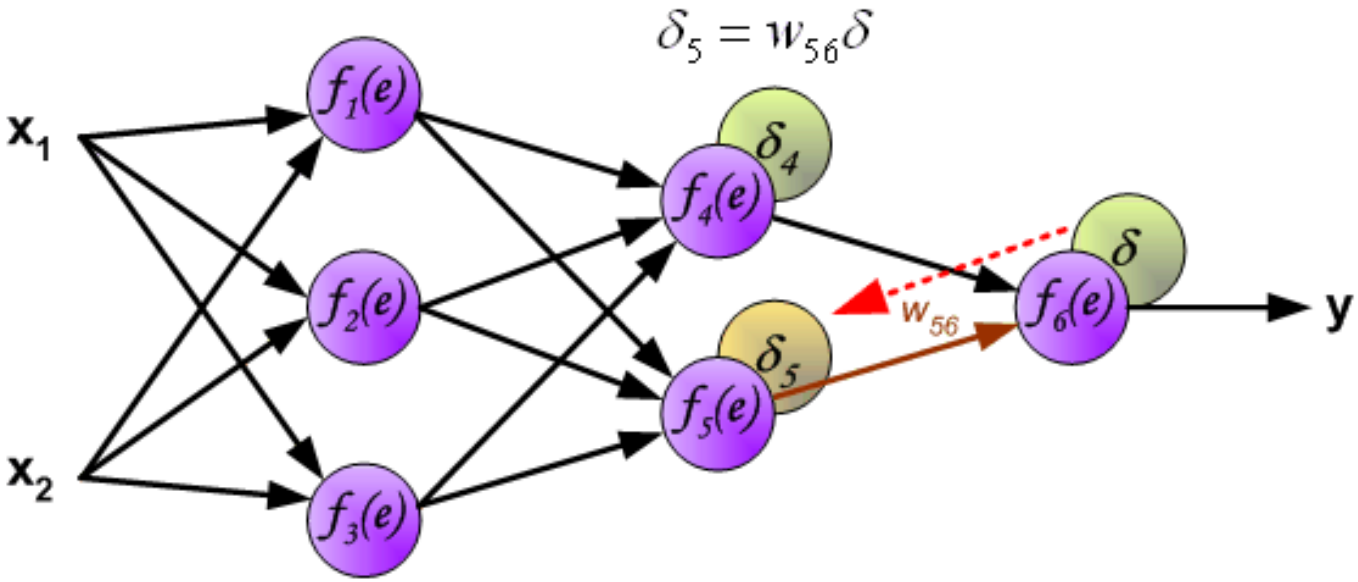
Learning Algorithm: Backpropagation

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.



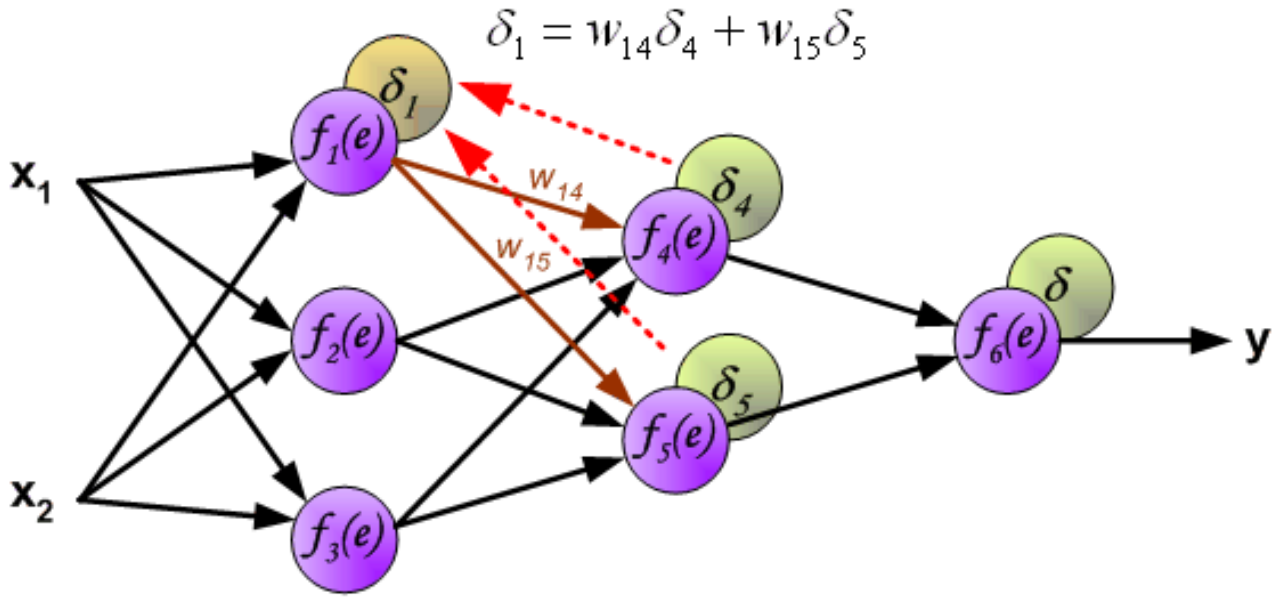
Learning Algorithm: Backpropagation

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:



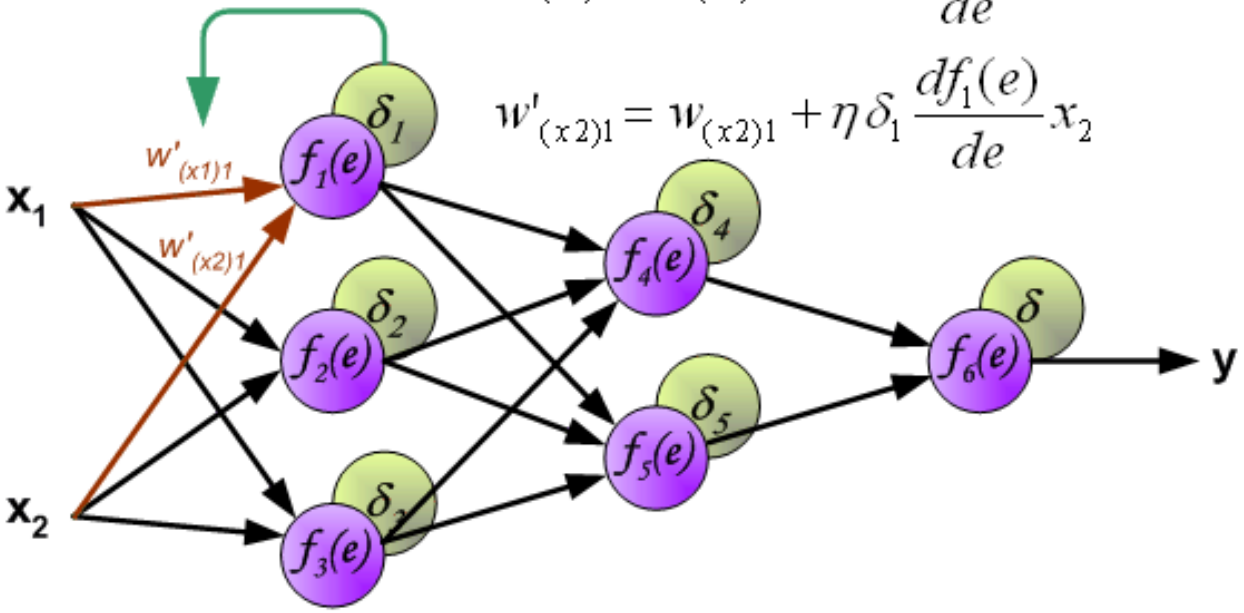
Learning Algorithm: Backpropagation

MACHINE LEARNING

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).

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

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



Learning Algorithm: Backpropagation

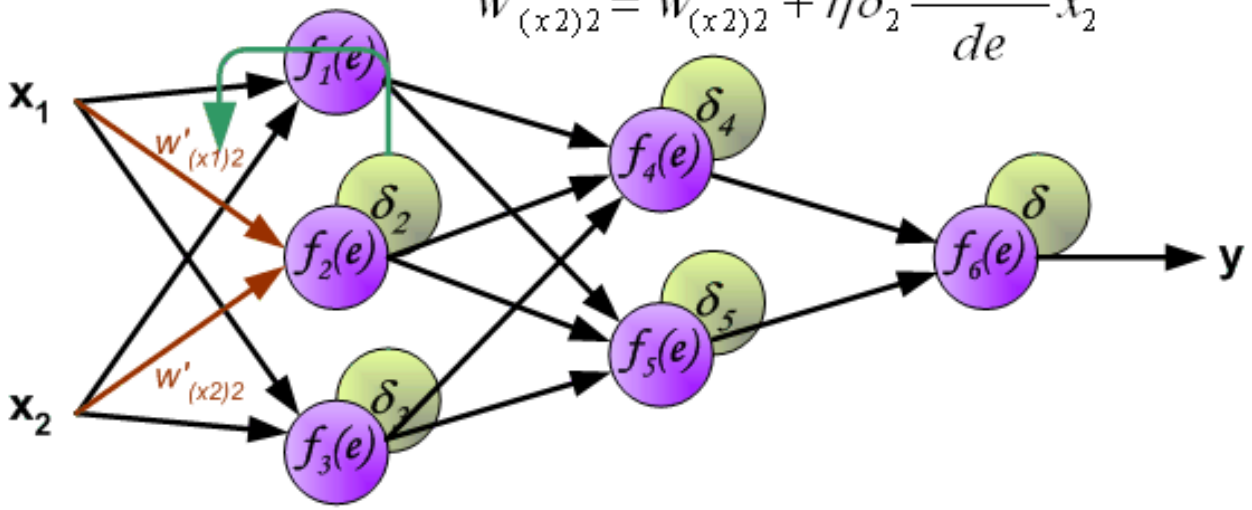
MACHINE LEARNING

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).

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$

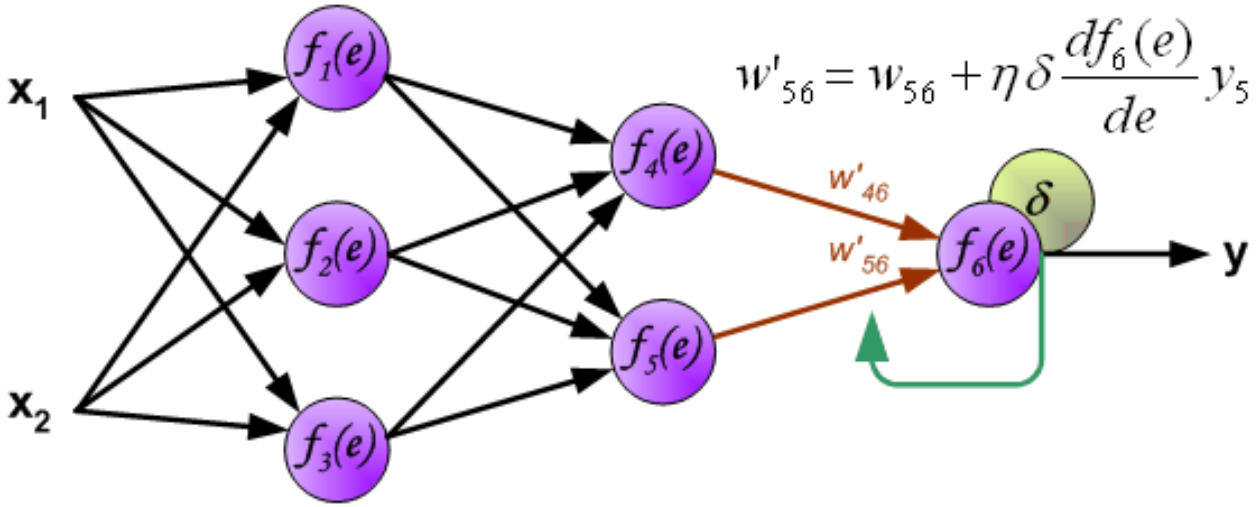


Learning Algorithm: Backpropagation

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).

$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

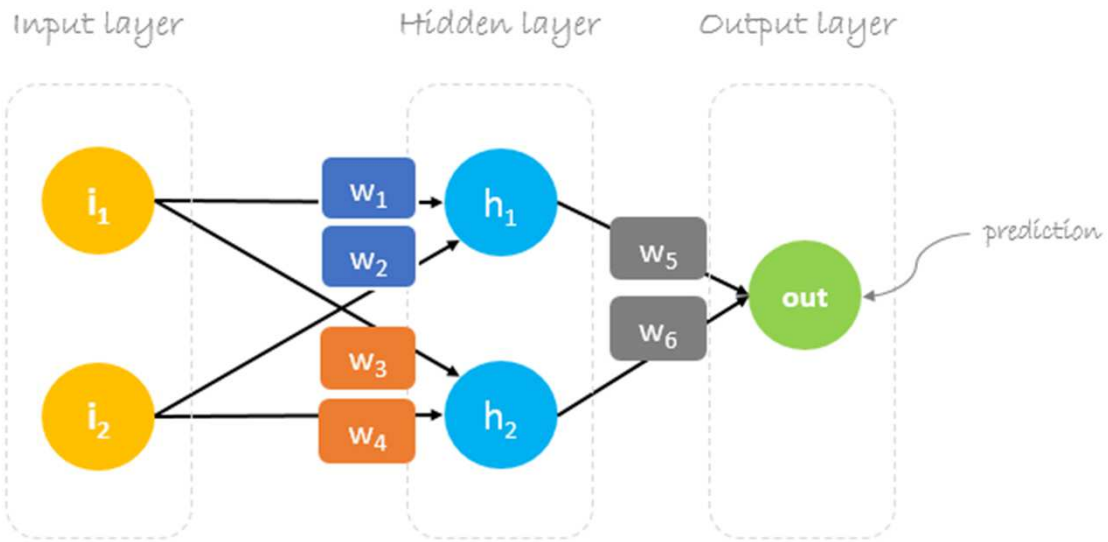
$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



Illustration

MACHINE LEARNING

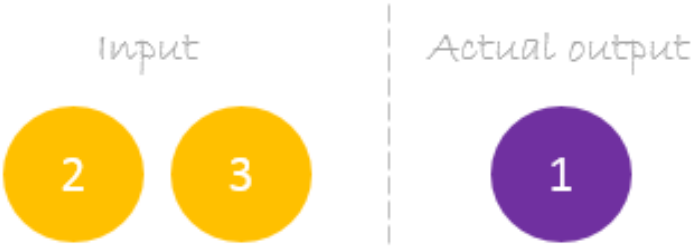
- we will build a neural network with three layers:
  - **Input** layer with two inputs neurons
  - One **hidden** layer with two neurons
  - **Output** layer with a single neuron



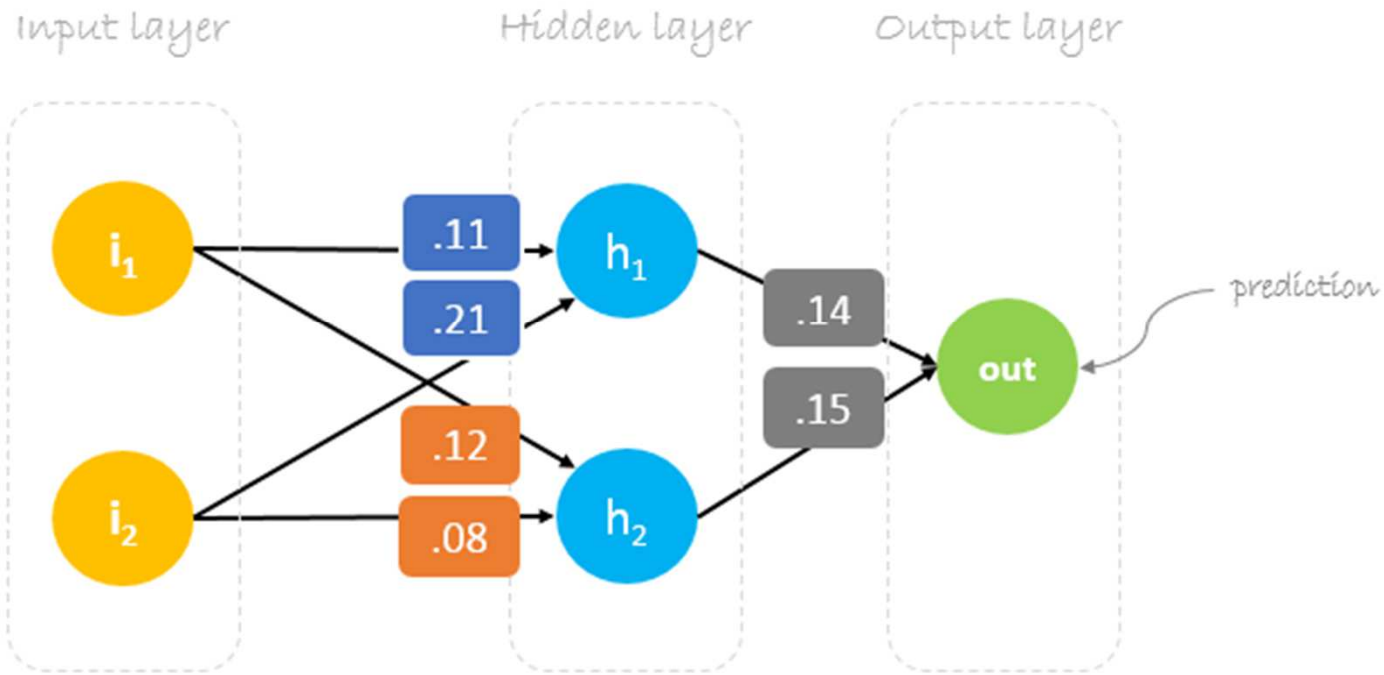
Dataset

Our dataset has one sample with two inputs and one output.

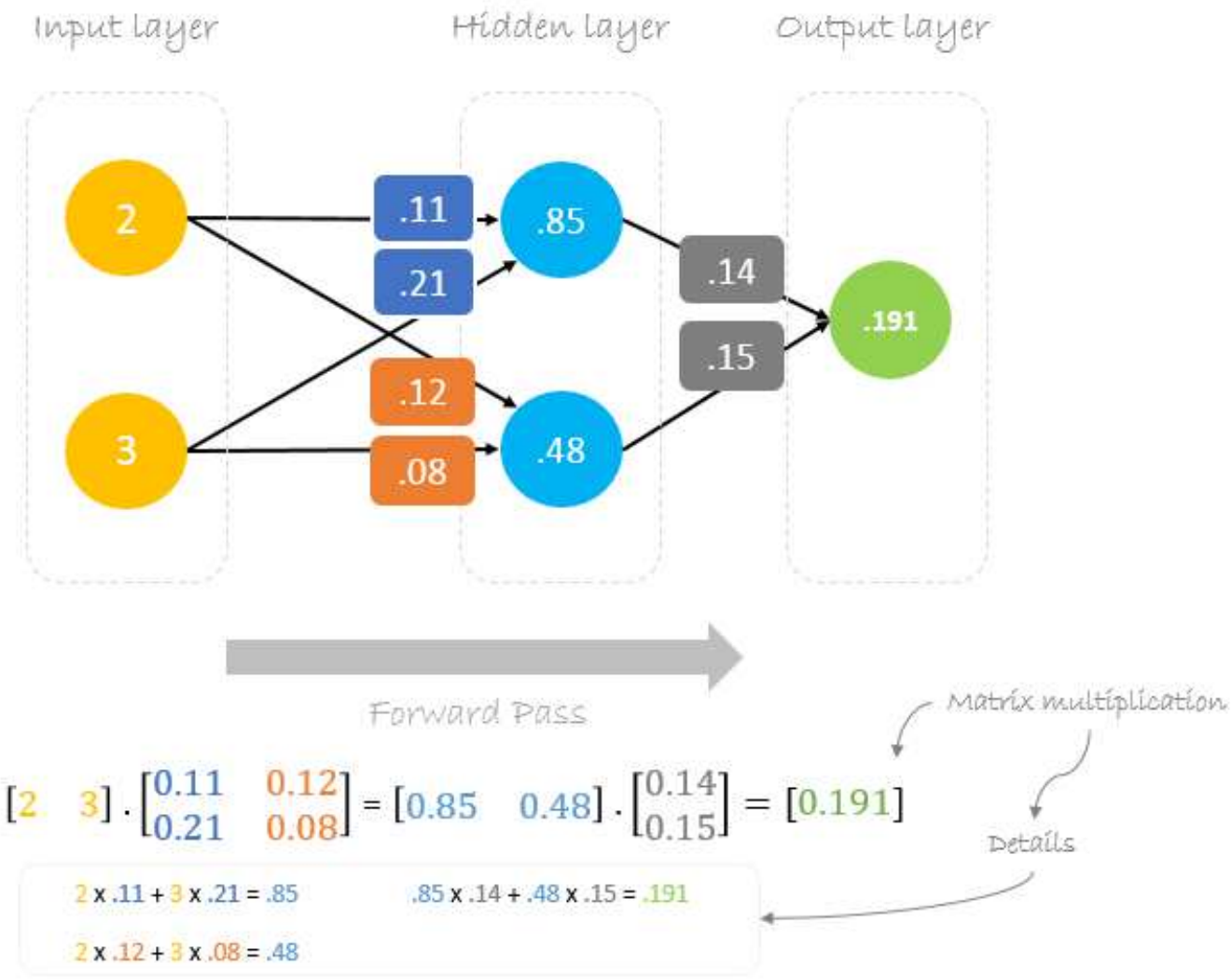
Our single sample is as following `inputs=[2, 3]` and `output=[1]`.



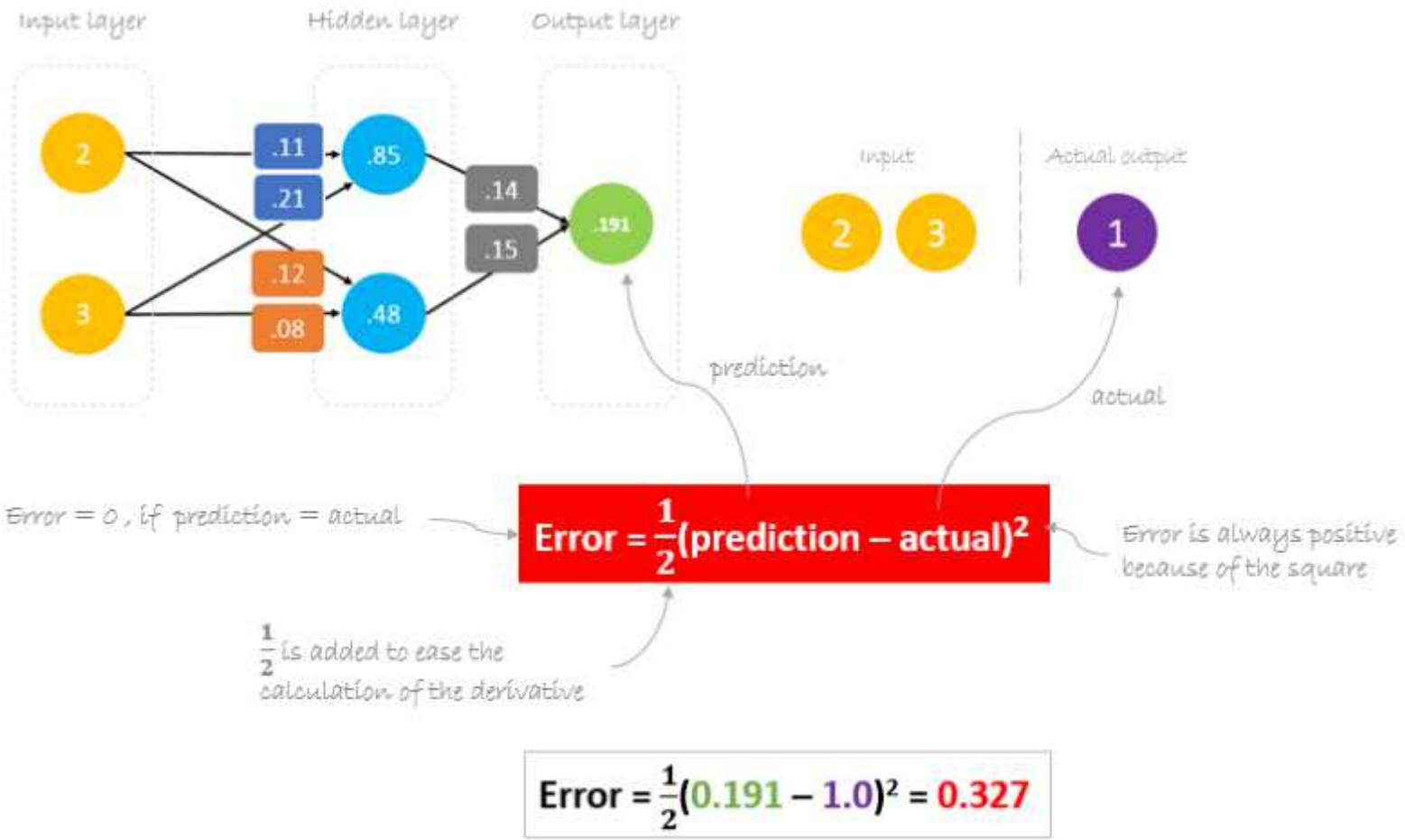
- Our initial weights will be as following: w1 = 0.11, w2 = 0.21, w3 = 0.12, w4 = 0.08, w5 = 0.14 and w6 = 0.15



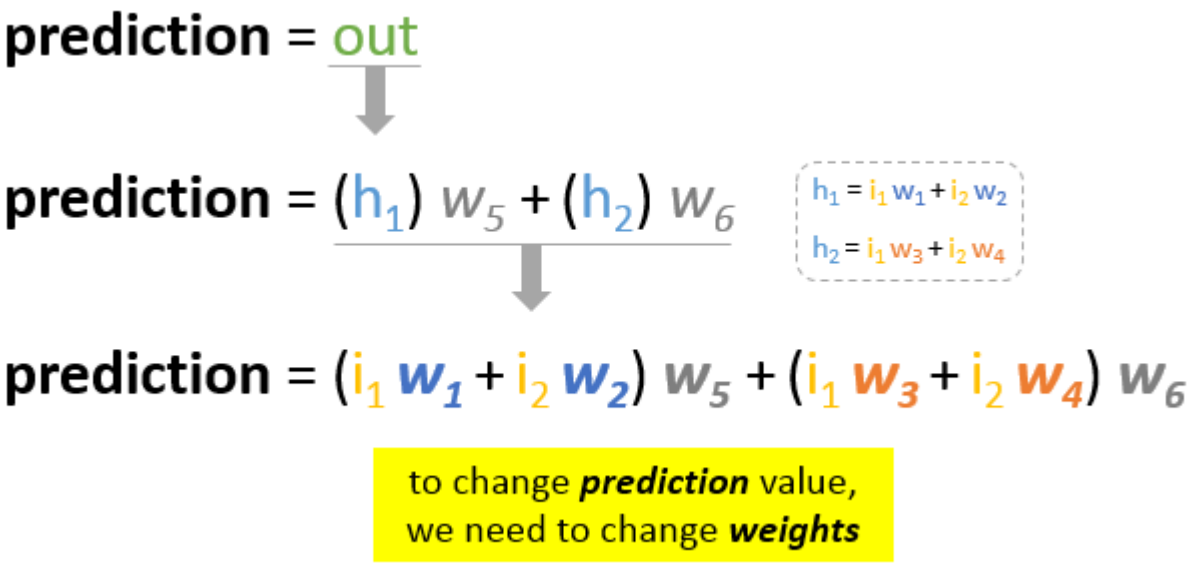
Forward Pass



Calculating Error



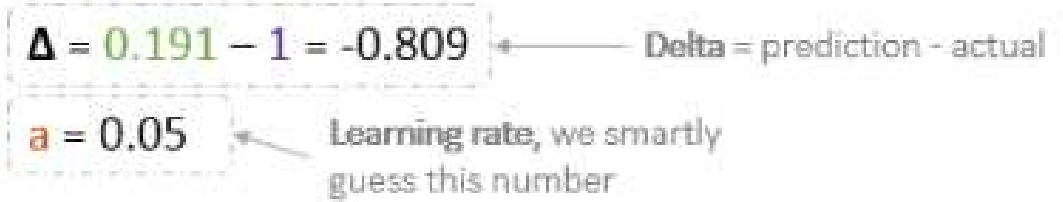
Reducing Error



The question now is **how to change\update the weights value so that the error is reduced?**

The answer is **Backpropagation!**

Backpropagation (of errors)

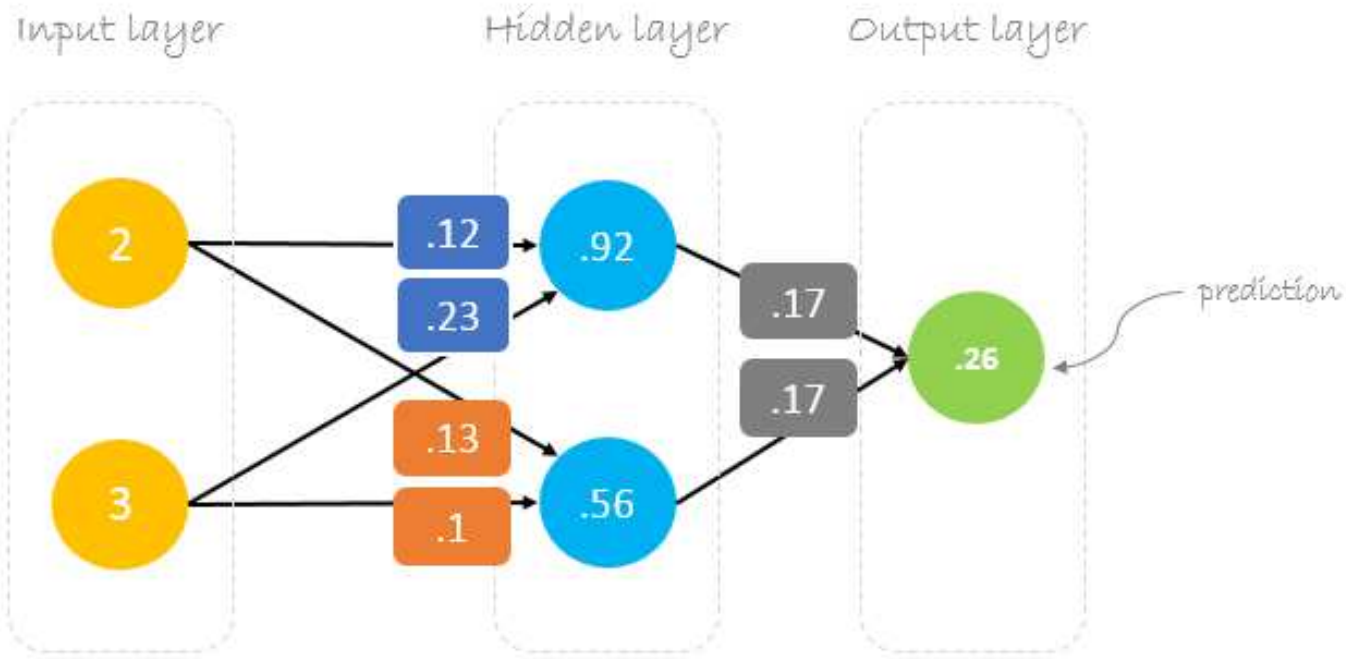


$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} . [0.14 \quad 0.15]$$

$$= \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

- Now, using the new **weights** we will repeat the forward pass through another training example



Representation Power

- *boolean functions:*
  - every boolean function can be representated by a two-layer network
- *continuous functions:*
  - every continuous function can be approximated with arbitrarily small error by a two-layer network (sigmoid units at the hidden layer and linear units at the output layer)
- *arbitrary functions:*
  - each arbitrary function can be approximated to arbitrary accuracy by a three-layer network

Inductive Bias

- every possible assignment of network weights represents a syntactically different hypothesis
  - $H = \{\vec{w} | \vec{w} \in \Re^{(n+1)}\}$
- **inductive bias:** smooth interpolation between data points

Illustration: Face Recognition

- **task:**
  - classifying camera image of faces of various people
  - images of 20 people were made, including approximately 32 different images per person
  - image resolution 120×128 with each pixel described by a greyscale intensity between 0 and 255
  - identifying the direction in which the persons are looking (i.e., left, right, up, ahead)

Illustration: Design Choice

- **input encoding:**
  - image encoded as a set of 30×32
  - pixel intensitiy values ranging from 0 to 255 linearly scaled from 0 to 1
  - $\Rightarrow$ reduces the number of inputs and network weights
  - $\Rightarrow$ reduces computational demands
- **output encoding:**
  - network must output one of four values indicating the face direction
  - *1-of-n* output encoding: 1 output unit for each direction
  - $\Rightarrow$ more degrees of freedom
  - $\Rightarrow$ difference between highest and second-highest output can be used as a measure of classification confidence

Illustration: Design Choice

- **network graph structure:**
  - BACKPROPAGATION works with any DAG of sigmoid units
  - question of how many units and how to interconnect them
  - using *standard design*: hidden layer and output layer where every unit in the hidden layer is connected with every unit in the output layer

  $\Rightarrow$ 30 hidden units

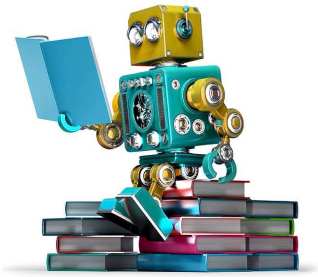
  $\Rightarrow$ test accuracy of 90%

Module 3- Outline

MACHINE LEARNING

Artificial Neural Network

1. Biological Motivation
2. Neural Network Representation
3. Appropriate Problems for NN learning
4. Perceptions
5. Multilayer Networks and Backpropagation Algorithm
6. **Remarks on Backpropagation Algorithm**
7. Summary



Convergence and Local Minima

- Convergence to global minima is not guaranteed
- Still, backpropagation is a highly effective function approximation method in practice.
- Despite the comments, gradient descent over the complex error surfaces represented by ANNs is still poorly understood,
  - no methods are known to predict with certainty when local minima will cause difficulties.
- Common heuristics to attempt to alleviate the problem of local minima include:
  - Add a momentum term to the weight-update rule
  - Use stochastic gradient descent rather than true gradient descent.
  - Train multiple networks using the same data, but initializing each network with different random weights.

Representational Power of Feedforward Networks

MACHINE LEARNING

- What set of functions can be represented by feedforward networks?
- Although much is still unknown about which function classes can be described by which types of networks, three quite general results are known:
  - Every Boolean functions.
    - number of hidden units required grows exponentially in the worst case with the number of network inputs.
  - Every bounded Continuous functions.
    - The networks that use sigmoid units at the hidden layer and (unthresholded) linear units at the output layer will achieve this.
    - The number of hidden units required depends on the function to be approximated.
  - Arbitrary functions.
    - Any function can be approximated to arbitrary accuracy by a network with three layers of units.

Hypothesis Space Search and Inductive Bias

MACHINE LEARNING

- Hypothesis space is the n-dimensional Euclidean space of the n network weights.
- Hypothesis space is continuous

- It is difficult to characterize precisely the inductive bias of backpropagation
  - Smooth interpolation between the data points

Hidden layer representation

- One intriguing property of Backpropagation is its ability to discover useful intermediate representations at the hidden unit layers inside the network
- Backpropagation to define hidden layer features that are not explicit in the input representation,
  - but which capture properties of the input instances that are most relevant to learning the target function.

Generalization, Overfitting, and Stopping Criterion

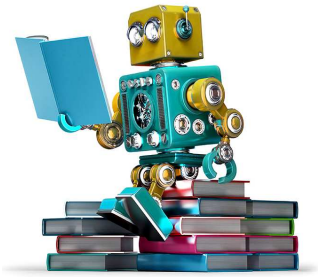
- Unspecified in the algorithm
- One obvious choice is to continue training until the error E on the training examples falls below some predetermined threshold.
  - This is a poor strategy because Backpropagation is susceptible to overfitting the training examples at the cost of decreasing generalization accuracy over other unseen examples.

Module 3- Outline

MACHINE LEARNING

Artificial Neural Network

1. Biological Motivation
2. Neural Network Representation
3. Appropriate Problems for NN learning
4. Perceptions
5. Multilayer Networks and Backpropagation Algorithm
6. Remarks on Backpropagation Algorithm
7. **Summary**



Summary

- able to learn discrete-, real- and vector-valued target functions
- noise in the data is allowed
- perceptrons learn hyperplane decision surfaces (linear separability)
- multilayer networks even learn nonlinear decision surfaces
- **BACKPROPAGATION** works on arbitrary feed-forward networks and uses gradient-descent to minimize the squared error over the set of training examples
- an arbitrary function can be approximated to arbitrary accuracy by a three-layer network
- **Inductive Bias**: smooth interpolation between data points

Advanced Topics

- hidden layer representations
- alternative error functions
- recurrent networks
- dynamically modifying network structure

Thank You