Deep Neural Network MLP

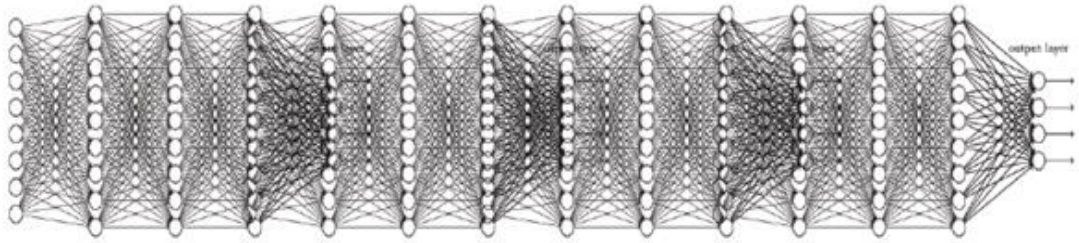
Multi-Layer Perceptron Structure

Multi-Layer Perceptron Functions

Multi-Layer Perceptron Training

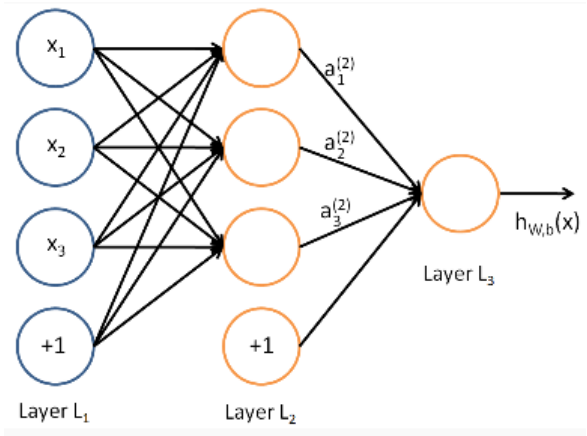
Contents

- Architecture
- Capability
- Learning

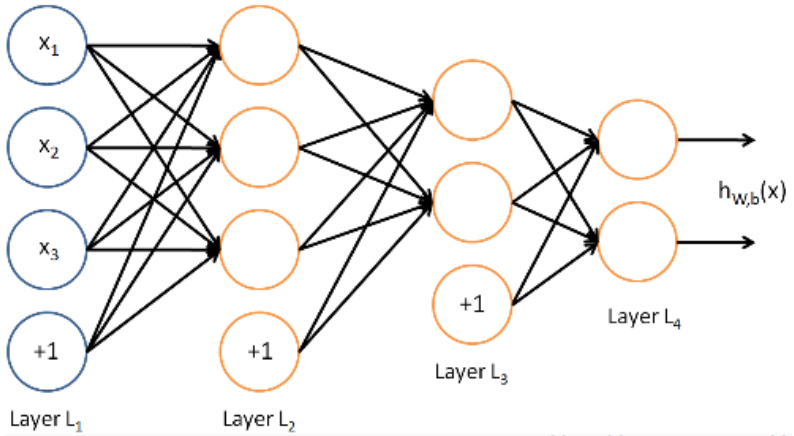


MLP – Architecture

- **Multi-Layer Perceptron**
  - **input layer, hidden layer(s), output layer**
  - **feed-forward**
  - **fully-connected**



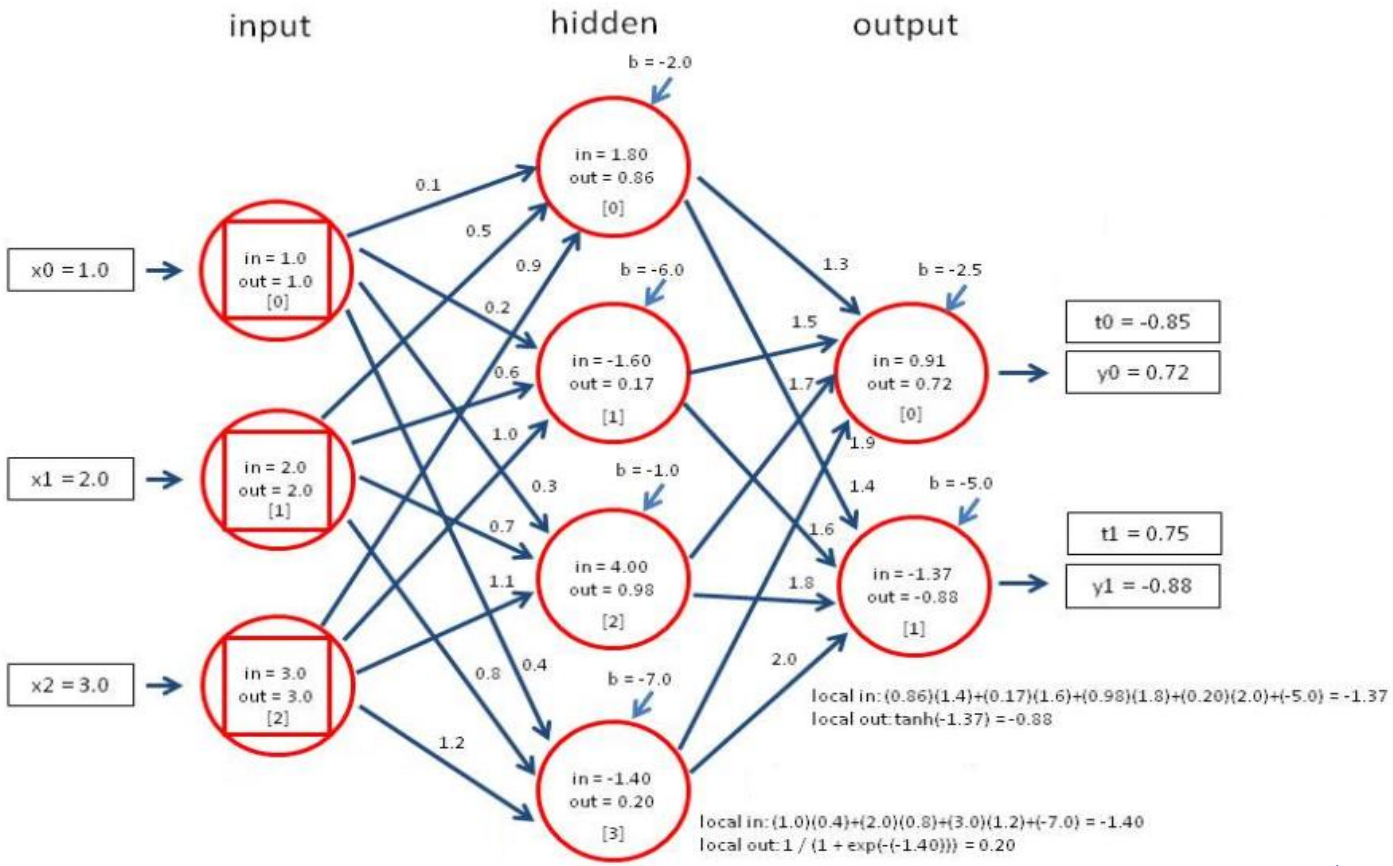
3×3×1 MLP



3×3×2×2 MLP

MLP – Architecture

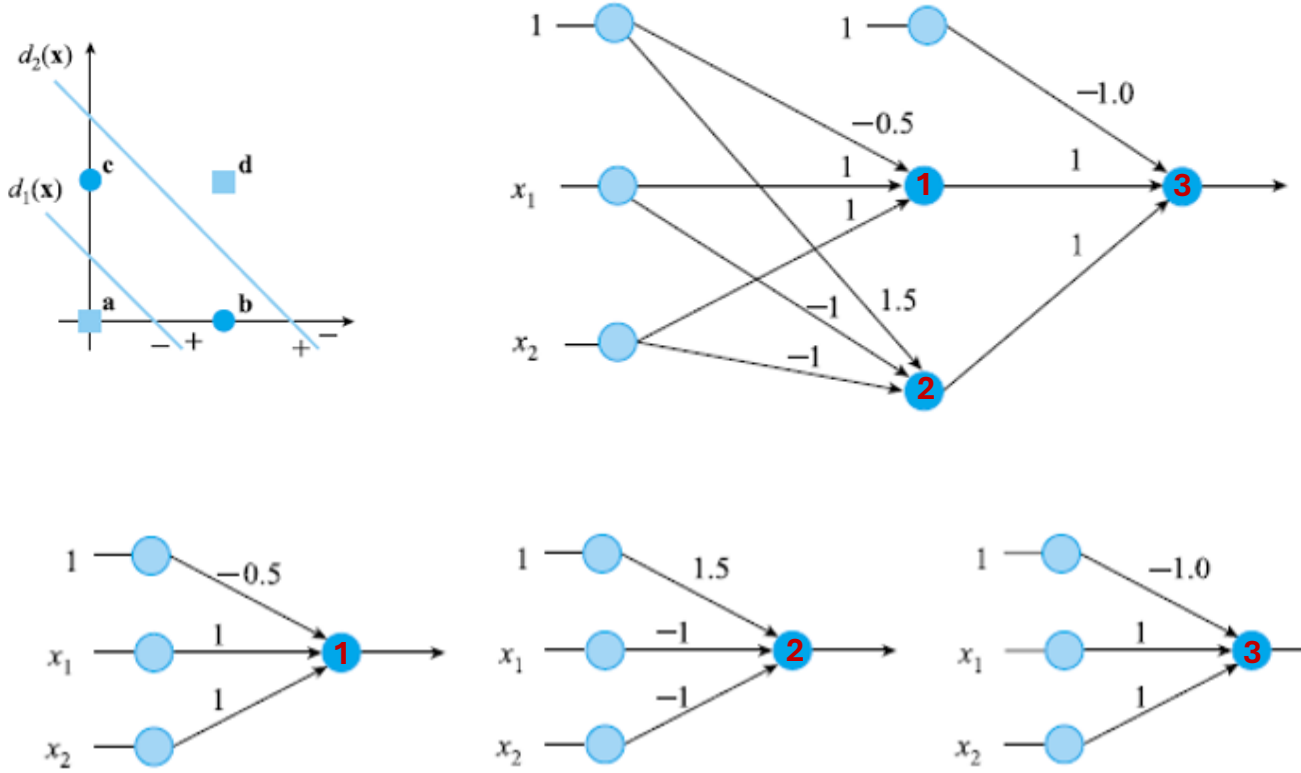
- **Forward calculations**



(*) assume $f = f_s(z) = \frac{1}{1+e^{-z}}$

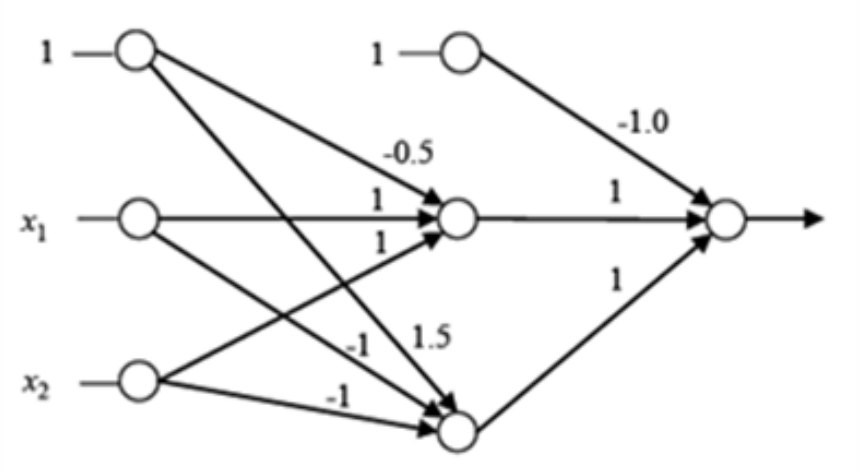
MLP – Capability

- **XOR function (sign activation, f_g)**



Exercise

Consider the following Multi-Layer Perceptron.



$$\tau(s) = \begin{cases} +1, s \geq 0 \\ 0, s < 0 \end{cases}$$

(1) Assuming that the activation function in each perceptron is a step function $\tau(s)$, what is the final output of this MLP for the pattern $x=(0.5, 0.5)^T$? Describe the processing sequence and specify the final output.

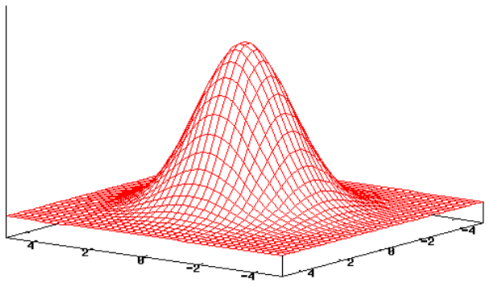
(2) Show the area of point P in $x_1 - x_2$ space where the MLP output for the P is -1.

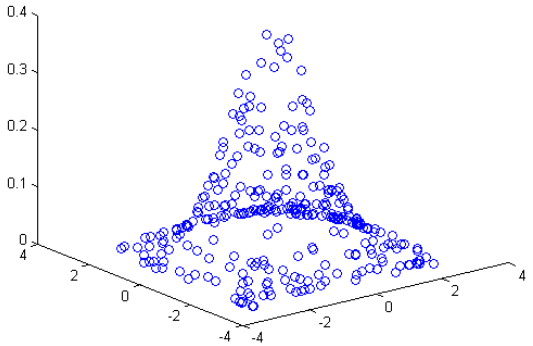
MLP – Capability

- **Approximation for a 2D Gaussian (sigmoid activation, f_S)**

$$f_X(x) = \frac{1}{(2\pi)^{\frac{n}{2}}\left[\det(C_X)\right]^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_X)' C_X^{-1}(x-\mu_X)\right)$$

$$\mu_X = \begin{bmatrix}0\\0\end{bmatrix};\ C_X = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$$

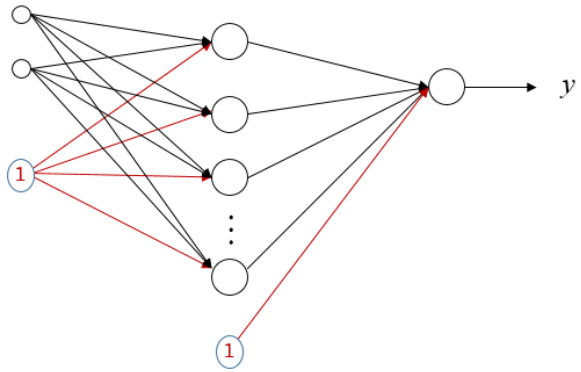




Number of training samples = 160
Number of hidden nodes = 8
Number of output nodes = 1
weights at Input layer

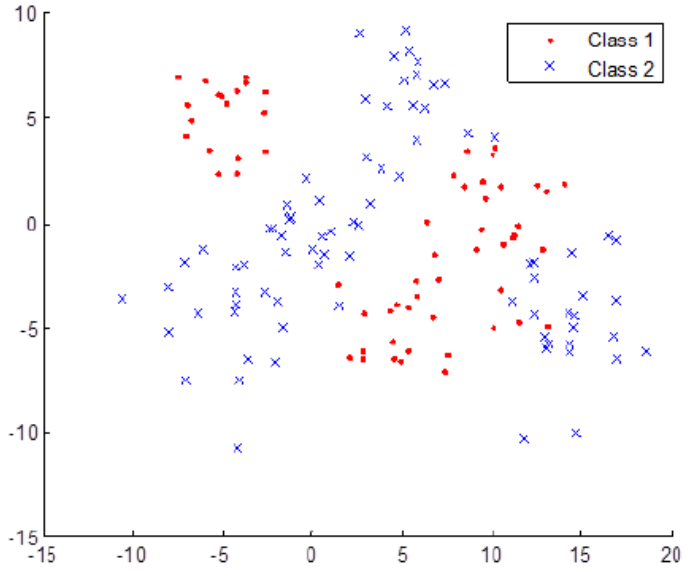
3.0945	2.2547
3.1498	2.5907
2.0944	-1.0050
0.1958	0.7448
-1.2822	4.1165
1.4074	3.4815
-1.9996	-0.5480
1.4700	-2.7861

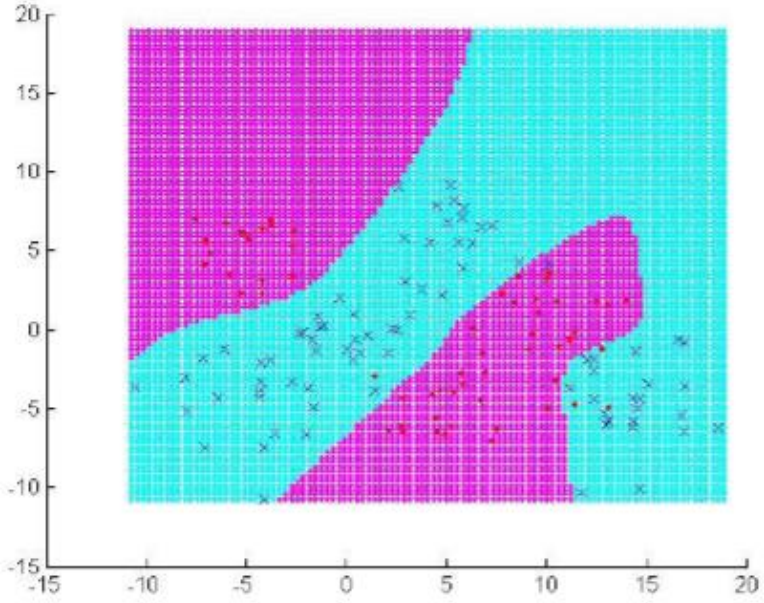
weights at output layer
[1.6538 0.9026 -1.7103 -7.4777 0.4493 0.6933 -2.0152 -1.5972]
Bias at output node
-2.2320
Bias at hidden nodes
[-4.4199 -2.9651 -0.8074 -0.6321 1.1862 0.9070 -0.6155 3.0402]



MLP – Capability

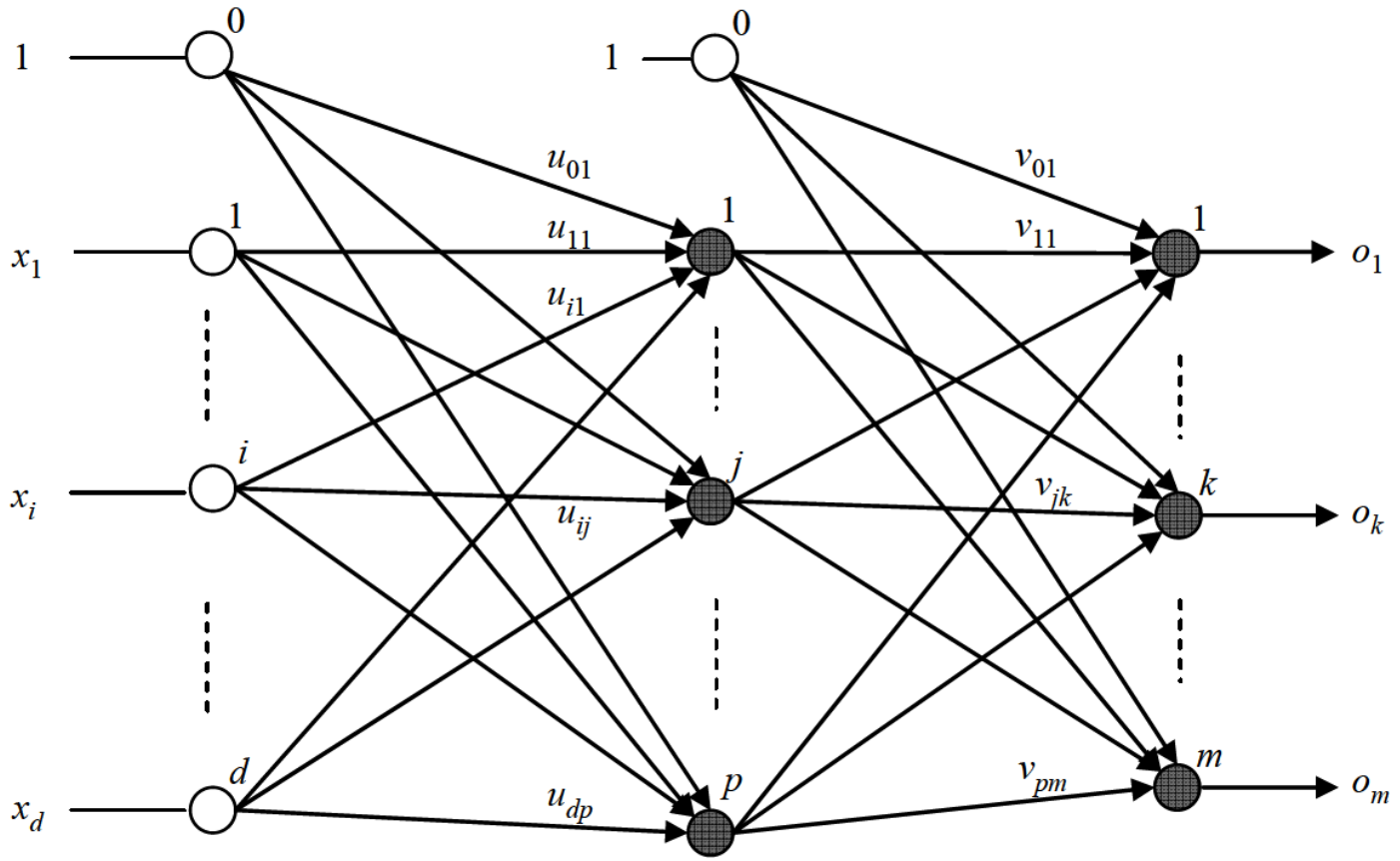
- **Nonlinear decision boundaries**
  - 2×4×1 MLP with sigmoid activation
  - learning rate = 0.01
  - max #-iterations = 9,000





MLP Learning: 2-layer MLP

- **Weight parameters in a $d \times p \times m$ MLP [오일석, 2008]**



MLP – Learning

- **Problem definition: given a set of N training samples,**

  $\mathbf{X}=\{ (\mathbf{x}_1, \mathbf{t}_1), (\mathbf{x}_2, \mathbf{t}_2), \ldots, (\mathbf{x}_N, \mathbf{t}_N) \}$,

  where $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]^T$ and $\mathbf{t}_i = [t_{i,1}, t_{i,2}, \ldots, t_{i,m}]^T$

☞ **find *a possible* u and v which map $\mathbf{x}_i$ to the corresponding $\mathbf{t}_i$.**

- **Cost function for the goodness of $\theta = \{\mathbf{u}, \mathbf{v}\}$**

$$E(\theta) = \frac{1}{2}\sum_{k=1}^{m}(o_k - t_k)^2$$

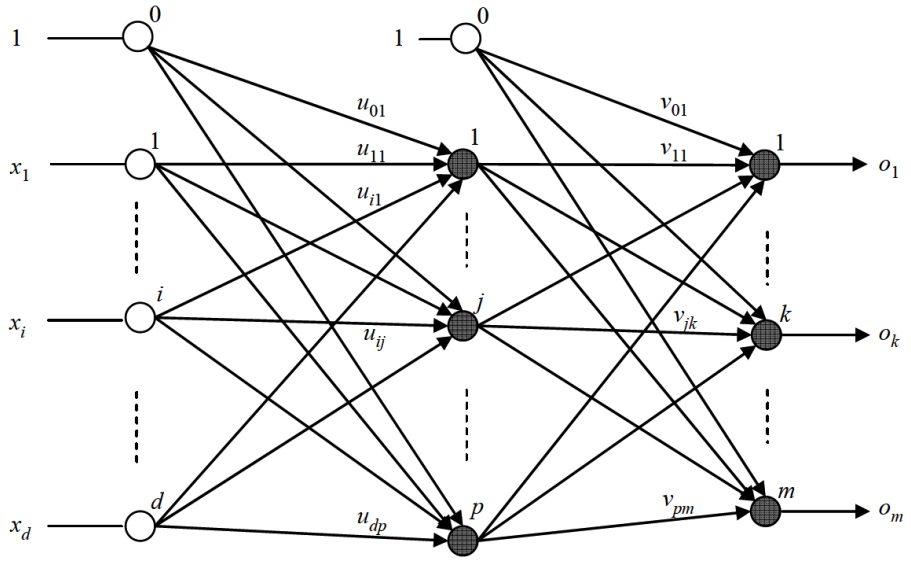
where m is the number of output nodes

MLP - Learning

- **Step 1: Forward calculations with a training sample x**
  - **input/output of j-th hidden node**

$$z\_sum_j = \sum_{i=1}^{d} x_i\, u_{ij} + u_{0j} = \sum_{i=0}^{d} x_i\, u_{ij}$$

$$z_j = \tau_1(z\_sum_j) = \tau_1\left(\sum_{i=0}^{d} x_i\, u_{ij}\right)$$



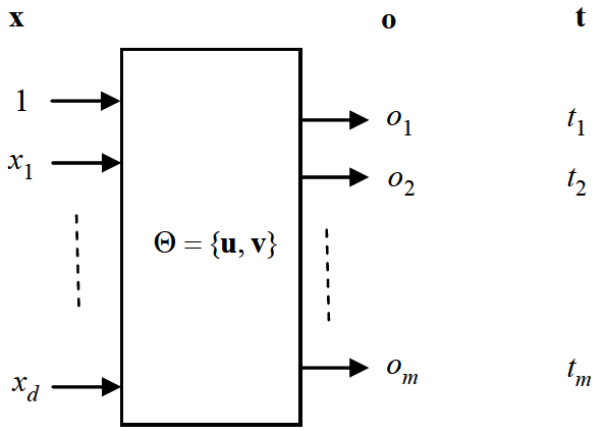
  - **input/output of k-th output node**

$$o\_sum_k = \sum_{j=1}^{p} z_j\, v_{jk} + v_{0k} = \sum_{j=0}^{p} z_j\, v_{jk}$$

$$o_k = \tau_2(o\_sum_k) = \tau_2\left(\sum_{j=0}^{p} z_j\, v_{jk}\right) = \tau_2\left(\sum_{j=0}^{p} \tau_1(z\_sum_j)\, v_{jk}\right) = \tau_2\left(\sum_{j=0}^{p} \tau_1\left(\sum_{i=0}^{d} x_i\, u_{ij}\right) v_{jk}\right)$$

MLP - Learning

- **Step 2: Error computation**



$$o_k = \tau_2\left(\sum_{j=0}^{p} z_j\, v_{jk}\right) = \tau_2\left(\sum_{j=0}^{p} \tau_1\left(\sum_{i=0}^{d} x_i\, u_{ij}\right) v_{jk}\right)$$

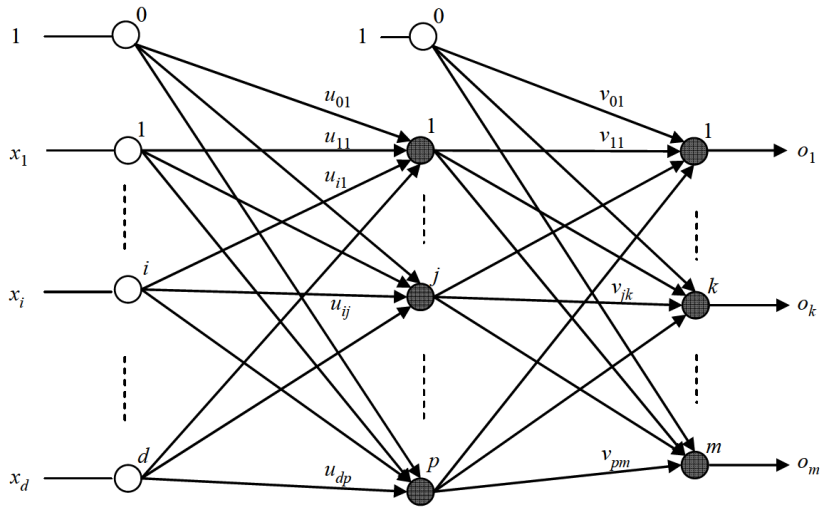
$$E = \frac{1}{2}\sum_{k=1}^{m}(o_k - t_k)^2 = \frac{1}{2}\sum_{k=1}^{m}\left(\tau_2\left(\sum_{j=0}^{p} z_j\, v_{jk}\right) - t_k\right)^2 = \frac{1}{2}\sum_{k=1}^{m}\left(\tau_2\left(\sum_{j=0}^{p} \tau_1\left(\sum_{i=0}^{d} x_i\, u_{ij}\right) v_{jk}\right) - t_k\right)^2$$

MLP - Learning

- **Step 3: weight updates by gradient descent method**

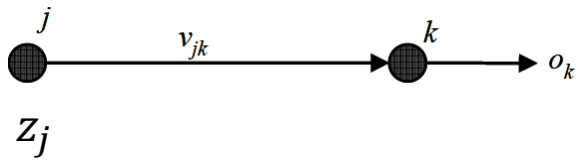
  - **Step 3-1:** $$\mathrm{v}(h+1) = \mathrm{v}(h) - \rho \frac{\partial E}{\partial \mathrm{v}}, \quad v_{jk}(h+1) = v_{jk}(h) - \rho \frac{\partial E}{\partial v_{jk}}$$

  - **Step 3-2:** $$\mathrm{u}(h+1) = \mathrm{u}(h) - \rho \frac{\partial E}{\partial \mathrm{u}}, \quad u_{ij}(h+1) = u_{ij}(h) - \rho \frac{\partial E}{\partial u_{ij}}$$



MLP - Learning

- $\frac{\partial E}{\partial v_{jk}}$ **for** v_{jk} **between hidden & output layer**



$$o\_sum_k = f_1(v_{jk}) = \sum_{j=0}^{p} z_j \, v_{jk}$$

$$o_k = f_2(o\_sum_k) = \tau_2(o\_sum_k)$$

$$E = f_3(o_k) = \frac{1}{2}\sum_{k=1}^{m}(o_k - t_k)^2$$

$$\rightarrow \; E = f_3(f_2(f_1(v_{jk})))$$

$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial f_3}{\partial v_{jk}} = \frac{\partial f_3}{\partial o_k} \times \frac{\partial f_2}{\partial o\_sum_k} \times \frac{\partial f_1}{\partial v_{jk}}$$

$$= (o_k - t_k) \times \tau'(o\_sum_k) \times z_j$$

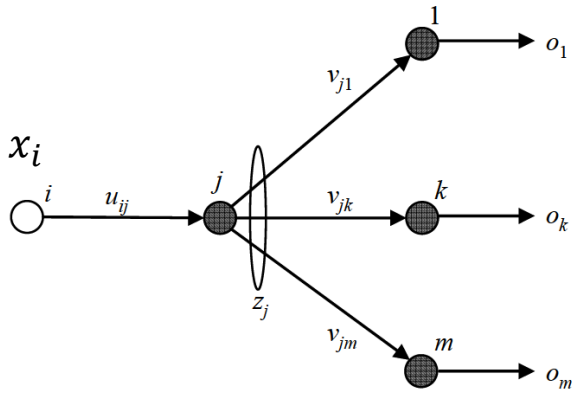
$$= \frac{\partial E}{\partial o_k} \times \tau'(o\_sum_k) \times z_j$$

$$= \delta_k^{(2)} z_j$$

$$\delta_k^{(2)} = \frac{\partial E}{\partial o_k} \times \tau_2'(o\_sum_k) = \frac{\partial E}{\partial o\_sum_k}$$

MLP - Learning

- $\frac{\partial E}{\partial u_{ij}}$ **for** u_{ij} **between input & hidden layer**



$$z\_sum_j = f_1(u_{ij}) = \sum_{i=0}^{d} x_i\, u_{ij}$$

$$z_j = f_2(z\_sum_j) = \tau_1(z\_sum_j)$$

$$o\_sum_k = f_3(z_j) = \sum_{j=0}^{p} z_j\, v_{jk}$$

$$o_k = f_4(o\_sum_k) = \tau_2(o\_sum_k)$$

$$E = f_5(o_k) = \frac{1}{2}\sum_{k=1}^{m}(o_k - t_k)^2$$

$$\rightarrow\ E = f_5(f_4(f_3(f_2(f_1(u_{ij})))))$$

$$\frac{\partial E}{\partial u_{ij}} = \frac{\partial f_5}{\partial o_k} \times \frac{\partial f_4}{\partial o\_sum_k} \times \frac{\partial f_3}{\partial z_j} \times \frac{\partial f_2}{\partial z\_sum_j} \times \frac{\partial f_1}{\partial u_{ij}}$$

$$= \sum_{k=1}^{m}(o_k - t_k) \times \tau'(o\_sum_k) \times v_{jk} \times \tau'(z\_sum_j) \times x_i$$

$$= \sum_{k=1}^{m} \delta_k^{(2)} \times v_{jk} \times \tau'(z\_sum_j) \times x_i$$

$$= \frac{\partial E}{\partial z_j} \times \tau'(z\_sum_j) \times x_i = \delta_j^{(1)} x_i$$

MLP - Learning

Algorithm [4.5]

Error Backpropagation Algorithm for Multi-Layer Perceptron (MLP) Learning (Pattern Mode)

Input: Training set $X = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$, learning rate ρ
Output: Weights u and v

Algorithm:

// Initialization

1. Initialize u and v.
2. $x_0 = z_0 = 1$; // Bias
3. **repeat** {
   4. **for** each sample X {
   5. Represent the current sample as $x = (x_1, x_2, \ldots, x_d)^T$ and $t = (t_1, t_2, \ldots, t_m)^T$.

   // Forward Computation

   6. **for** $j = 1$ to p { $z\_sum_j = \sum_{i=0}^{d} x_i u_{ij}$; $z_j = \tau(z\_sum_j)$; } // (4.12)
   7. **for** $k = 1$ to m { $o\_sum_k = \sum_{j=0}^{p} z_j v_{jk}$; $o_k = \tau(o\_sum_k)$; } // (4.13)

   // Error Backpropagation

   8. **for** $k = 1$ to m { $\delta_k = (t_k - o_k)\tau'(o\_sum_k)$; } // (4.18)
   9. **for** all $v_{jk}, 0 \le j \le p, 1 \le k \le m$ { $\Delta v_{jk} = \rho \delta_k z_j$; } // (4.19)
   10. **for** $j = 1$ to p { $\eta_j = \tau'(z\_sum_j) \sum_{k=1}^{m} \delta_k v_{jk}$; } // (4.20)
   11. **for** all $u_{ij}, 0 \le i \le d, 1 \le j \le p$ { $\Delta u_{ij} = \rho \eta_j x_i$; } // (4.21)

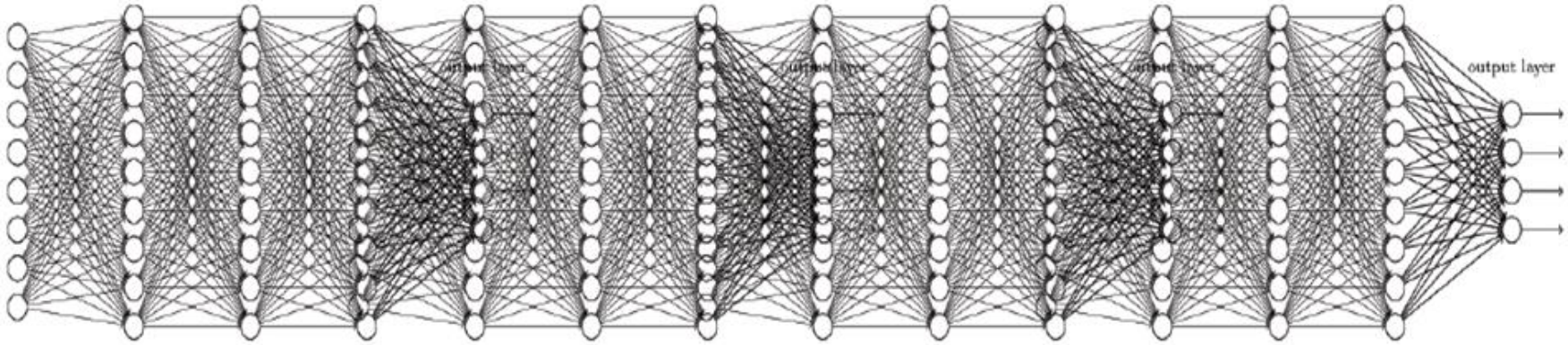
   // Weight Update

   12. **for** all $v_{jk}, 0 \le j \le p, 1 \le k \le m$ { $v_{jk} = v_{jk} + \Delta v_{jk}$; } // (4.17)
   13. **for** all $u_{ij}, 0 \le i \le d, 1 \le j \le p$ { $u_{ij} = u_{ij} + \Delta u_{ij}$; } // (4.17)

   }
4. **until** (stop-condition);
5. Save u and v.

MLP – Learning: Deep MLP

- **Deep MLP**



layer 1 *layer l* *layer L*

- x_i : i-th input value of layer 1
- $u_{ij}^{(l)}$: weight from i-th node of layer l to j-th node of layer $l+1$
- $z_j^{(l)}$: output from j-th node of layer l
- $o_k(t_k)$: k-th output(desired output) value in layer L

MLP – Learning: Deep MLP

- **Problem definition: given a set of N training samples,**

  $\mathbf{X}=\{ (\mathbf{x}_1, \mathbf{t}_1), (\mathbf{x}_2, \mathbf{t}_2), \ldots, (\mathbf{x}_N, \mathbf{t}_N) \}$,

  where $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]^T$ and $\mathbf{t}_i = [t_{i,1}, t_{i,2}, \ldots, t_{i,m}]^T$

☞ **find *a possible* set of all weights $u_{ij}^{(l)}$ which map $\mathbf{x}_i$ to $\mathbf{t}_i$.**

- **Cost function for the goodness of $\theta = \left\{u_{ij}^{(l)}\right\}$**

$$E(\theta) = \frac{1}{2}\sum_{k=1}^{m}(o_k - t_k)^2$$

where m is the number of output nodes

BP for Deep MLP (1)

- **Step 1: Forward calculations with a training sample x**

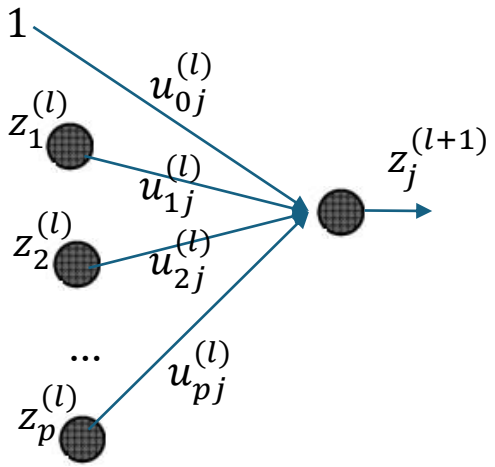
$$net_j^{(l+1)} = \sum_i z_i^{(l)} u_{ij}^{(l)}$$

$$z_j^{(l+1)} = \tau\left(net_j^{(l+1)}\right)$$

...

$$net_j^{(L)} = \sum_i z_i^{(L-1)} u_{ij}^{(L-1)}$$

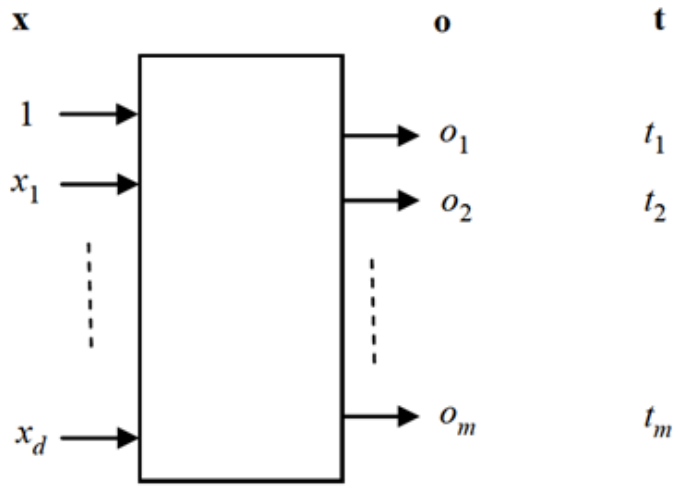
$$o_j = z_j^{(L)} = \tau\left(net_j^{(L)}\right)$$



BP for Deep MLP (2)

- **Step 2: Error computation**

$$E = \frac{1}{2}\sum_{k=1}^{m}(o_k - t_k)^2$$

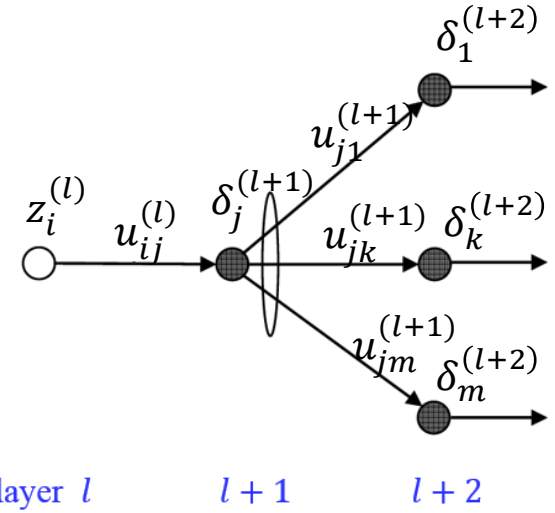


BP for Deep MLP (3)

- **Step 3: weight updates by error back-propagation**

$$u_{ij}^{(l)} = u_{ij}^{(l)} - \rho \frac{\partial E}{\partial u_{ij}^{(l)}}, \qquad l = L-1, L-2, \ldots, 1$$

$$\frac{\partial E}{\partial u_{ij}^{(l)}} = \frac{\partial E}{\partial net_j^{(l+1)}} \times \frac{\partial net_j^{(l+1)}}{\partial u_{ij}^{(l)}} = \delta_j^{(l+1)} \times z_i^{(l)}$$



- **Error Back-propagation**

$$\delta_j^{(L)} = \tau'\left(net_j^{(L)}\right)\left(o_j - t_j\right) \qquad output\ layer$$

$$\delta_j^{(l+1)} = \tau'\left(net_j^{(l+1)}\right)\sum_k u_{jk}^{(l+1)}\delta_k^{(l+2)} \qquad non-output\ layer$$

REVIEW

doi:10.1038/nature14539

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Machine-learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones. Machine-learning systems are used to identify objects in images, transcribe speech into text, match news items, posts or products with users' interests, and select relevant results of search. Increasingly, these applications make use of a class of techniques called deep learning.

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government. In addition to beating records in image recognition[1–4] and speech recognition[5–7], it has beaten other machine-learning techniques at predicting the activity of potential drug molecules[8], analysing particle accelerator data[9,10], reconstructing brain circuits[11], and predicting the effects of mutations in non-coding DNA on gene expression and disease[12,13]. Perhaps more surprisingly, deep learning has produced extremely promising results for various tasks in natural language understanding[14], particularly topic classification, sentiment analysis, question answering[15] and language translation[16,17].

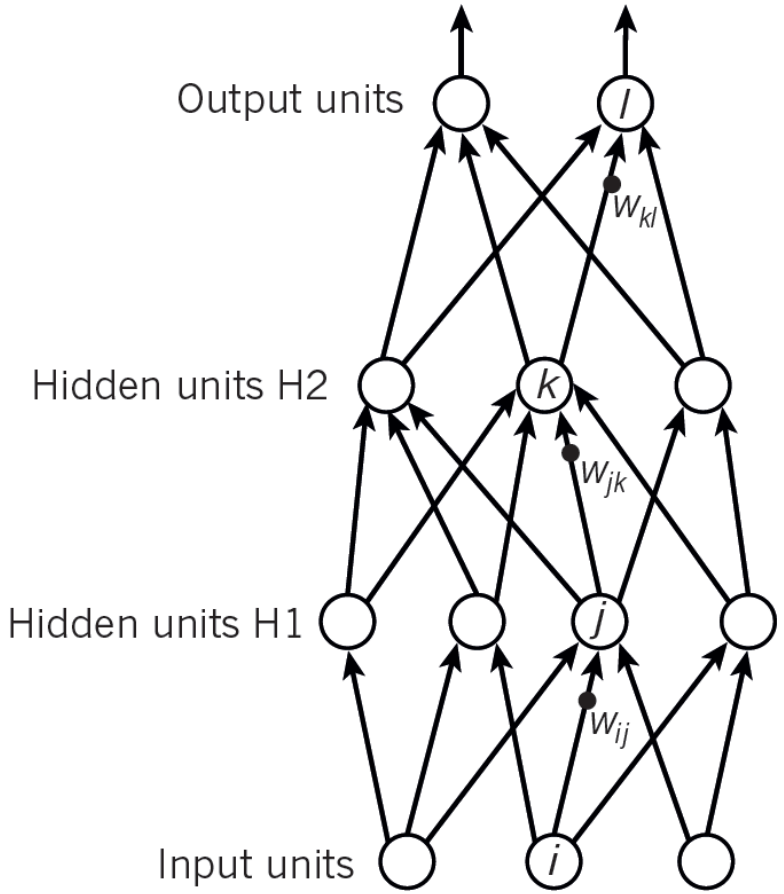
We think that deep learning will have many more successes in the near future because it requires very little engineering by hand, so it can easily take advantage of increases in the amount of available computation and data. New learning algorithms and architectures that are currently being developed for deep neural networks will only accelerate this progress.

Supervised learning

The most common form of machine learning, deep or not, is supervised learning. Imagine that we want to build a system that can classify images as containing, say, a house, a car, a person or a pet. We first

MLP Deep Learning

- **Forward calculations [Nature, 2015]**



$$y_l = f(z_l)$$

$$z_l = \sum_{k \,\varepsilon\, \text{H2}} w_{kl}\, y_k$$

$$y_k = f(z_k)$$

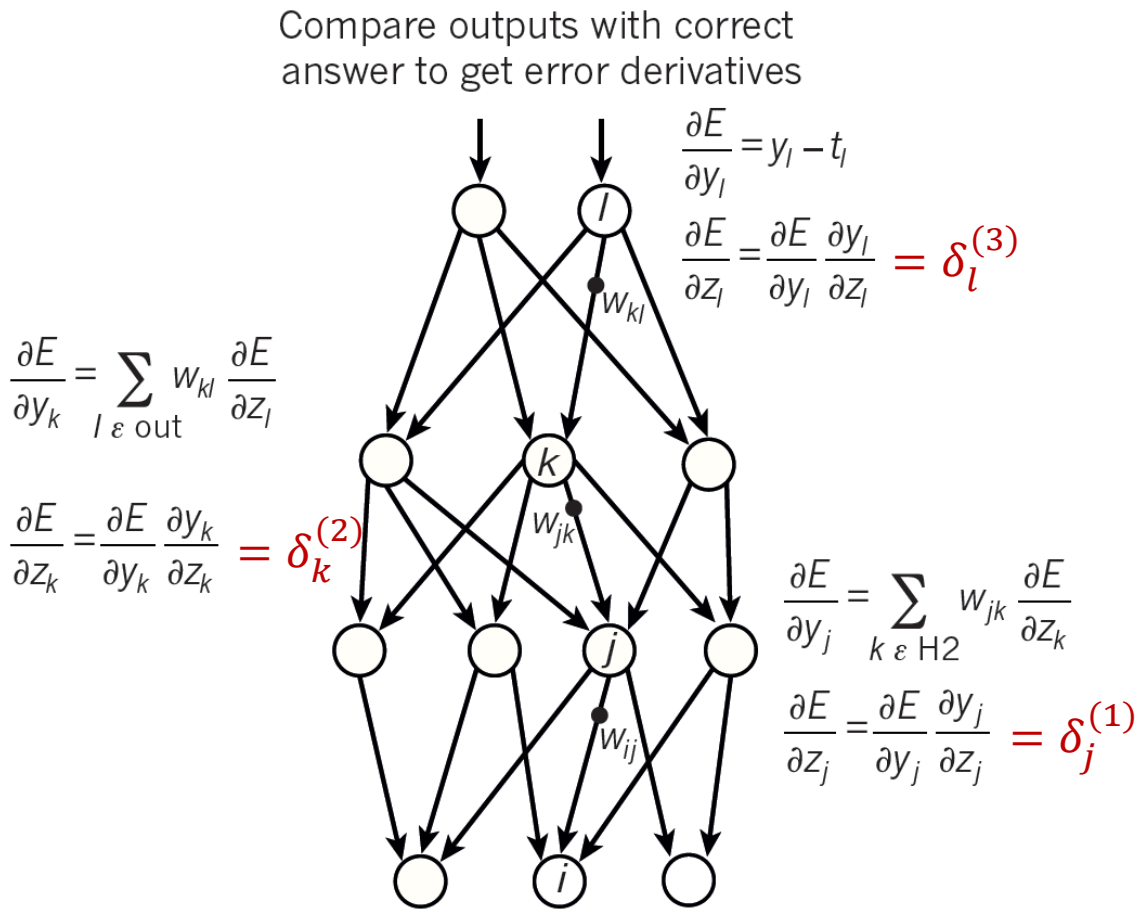
$$z_k = \sum_{j \,\varepsilon\, \text{H1}} w_{jk}\, y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \,\varepsilon\, \text{Input}} w_{ij}\, x_i$$

MLP Deep Learning

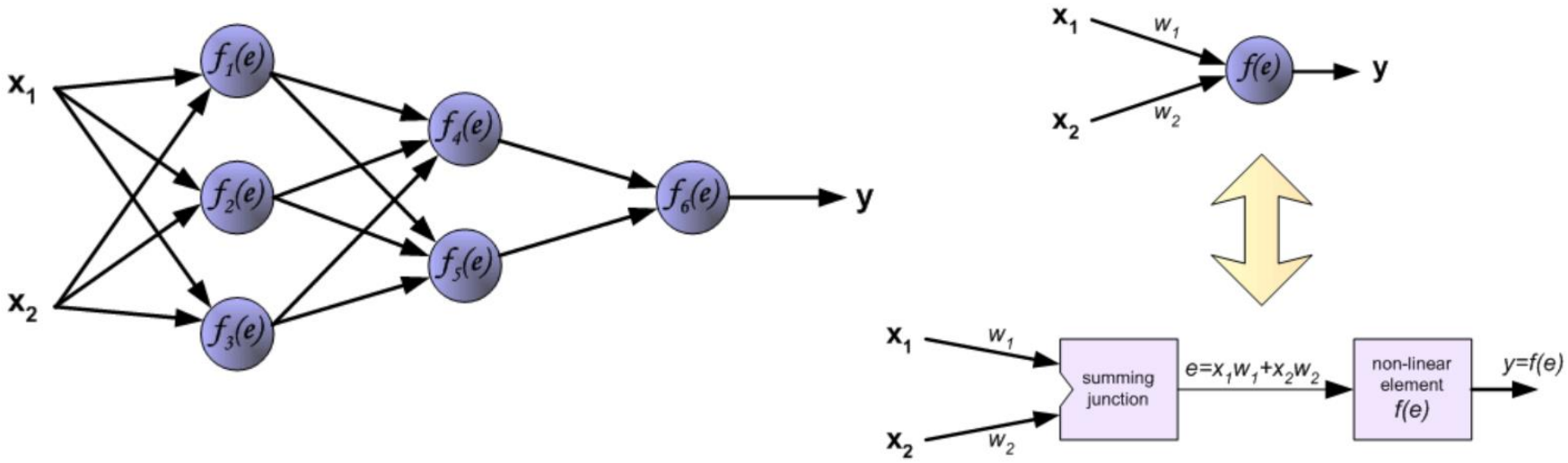
- **Backward error propagations [Nature, 2015]**



MLP - Learning

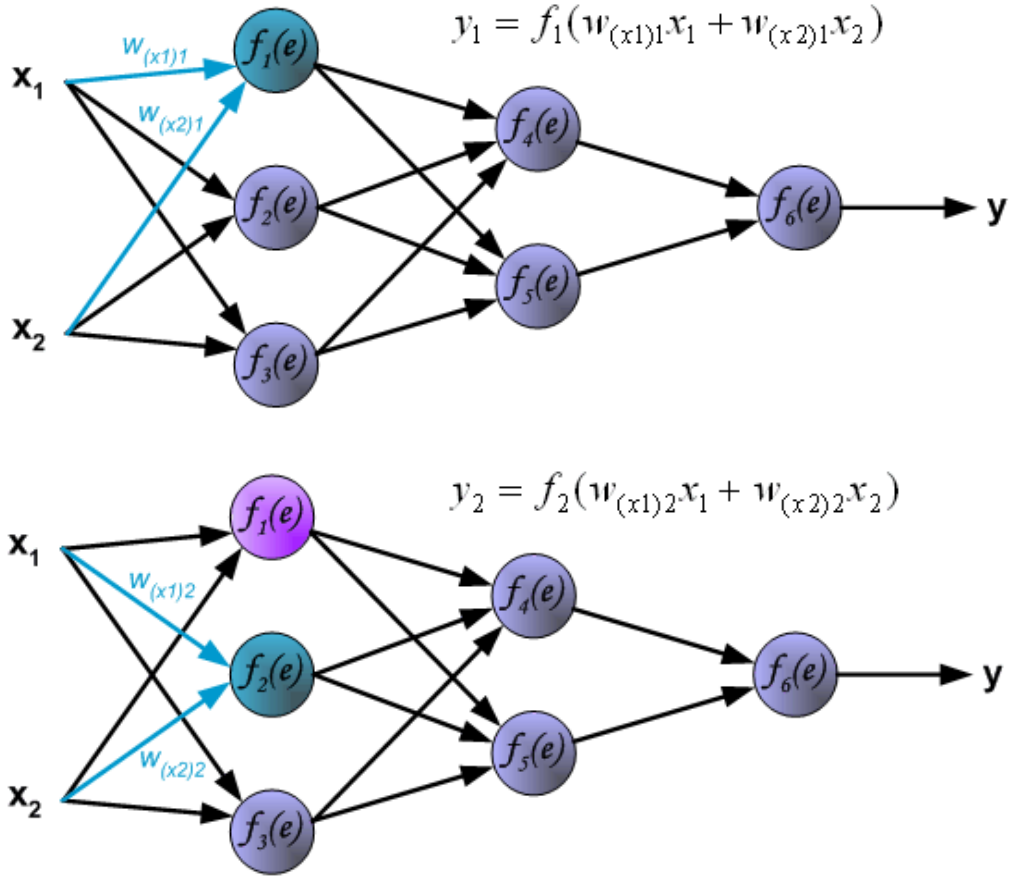
- **Visual understanding of BP algorithm**

  (http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html)



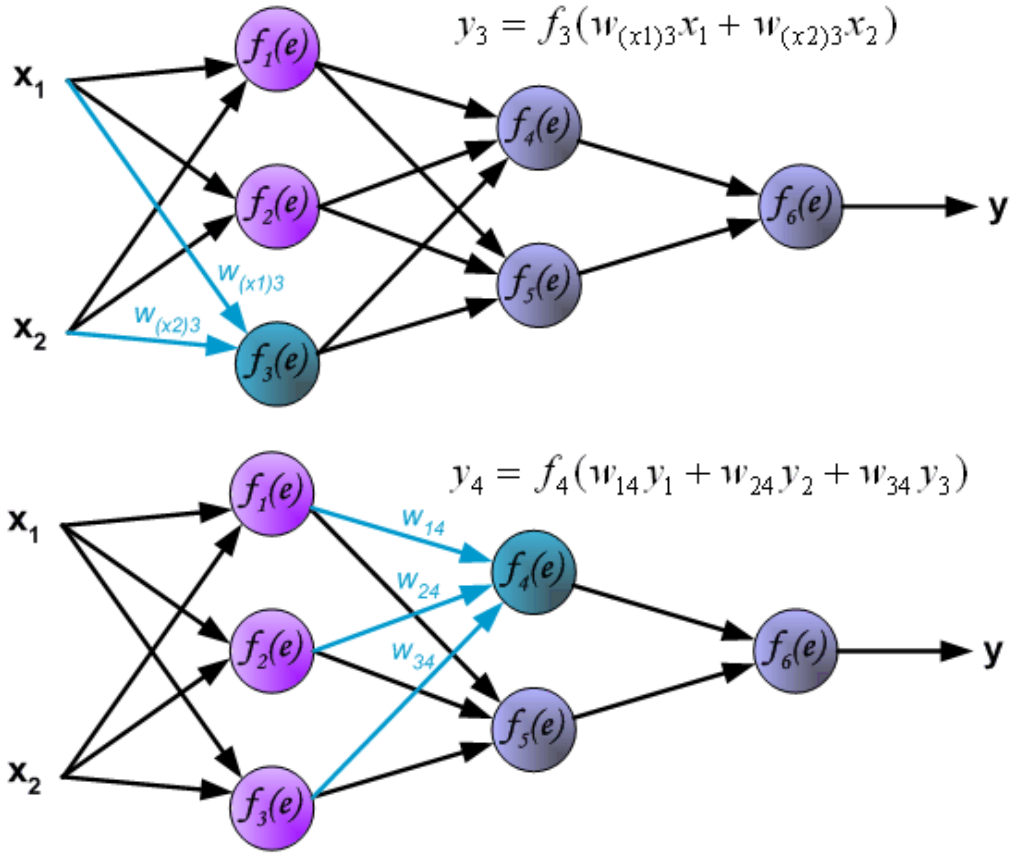
MLP - Learning

- **forward calculations(1)**



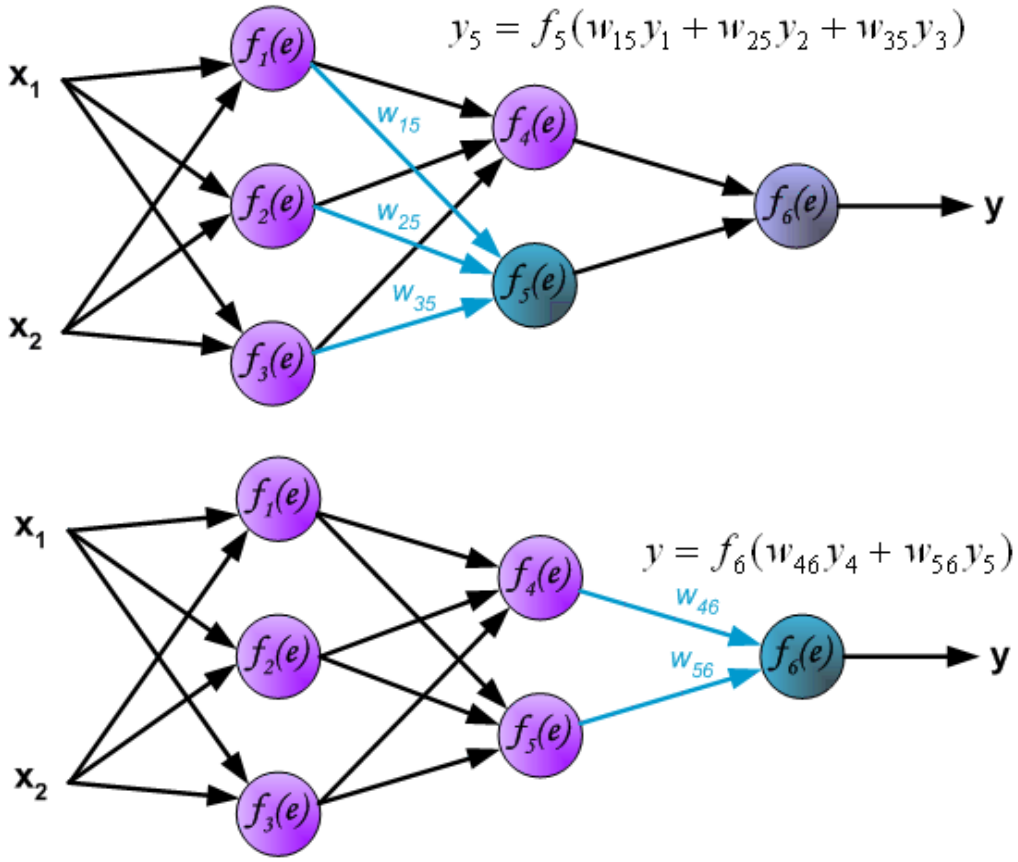
MLP - Learning

- **forward calculations(2)**



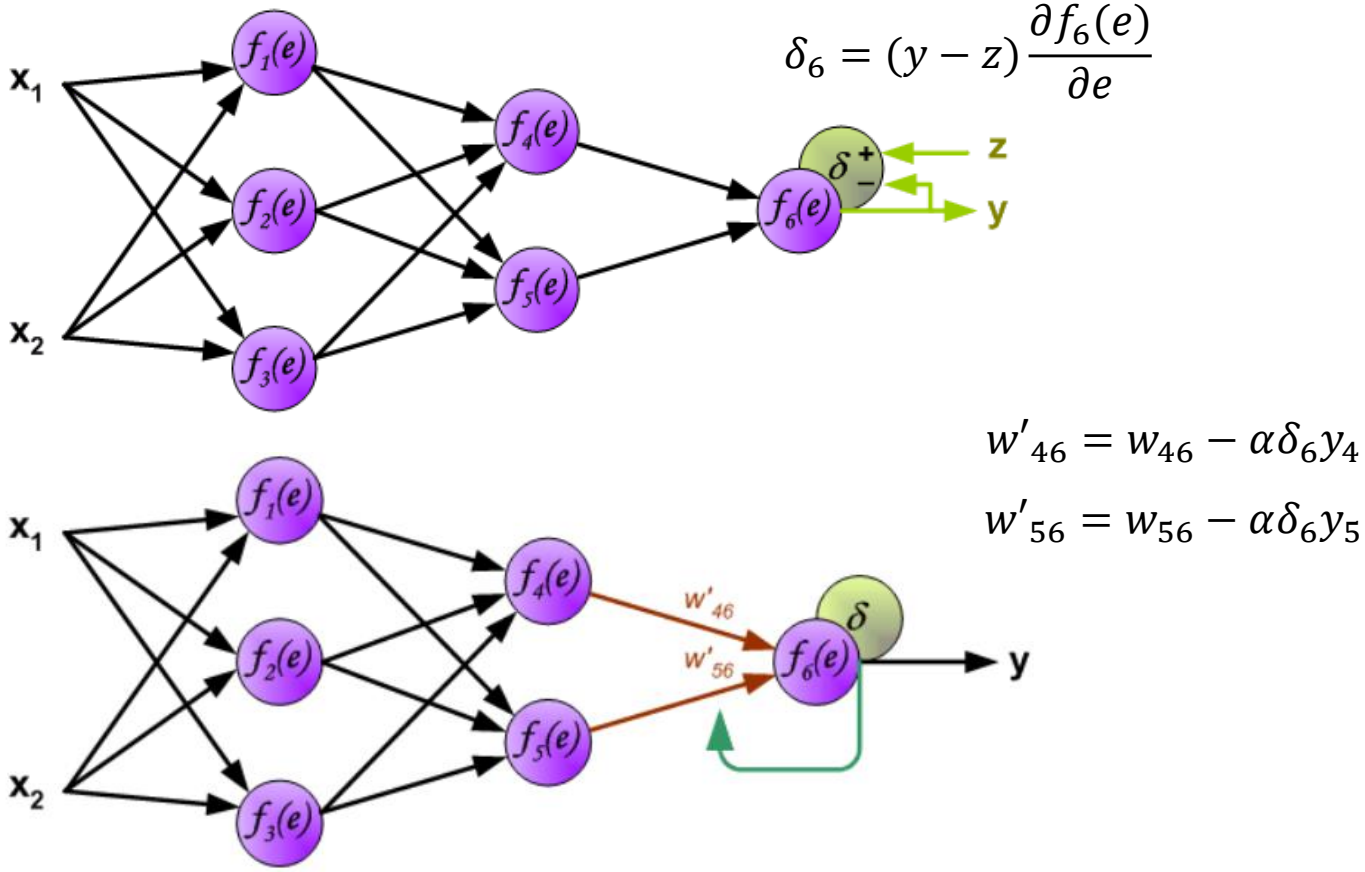
MLP - Learning

- **forward calculations(3)**



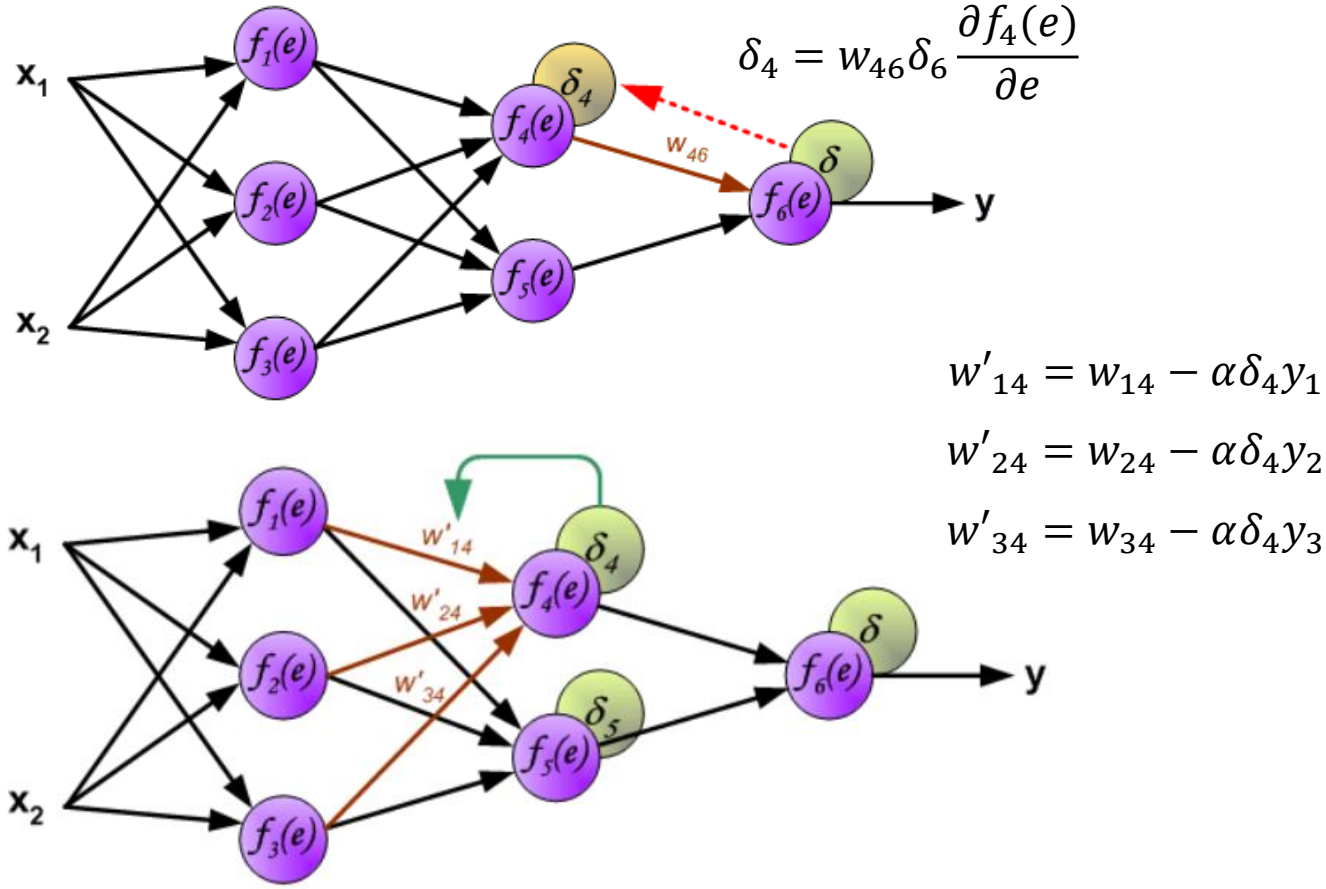
MLP - Learning

- **weight update with error back-propagation(1)**



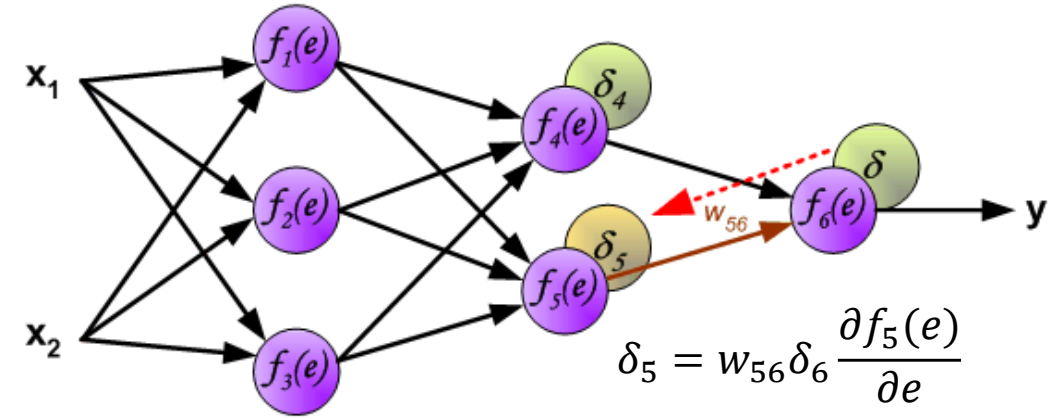
MLP - Learning

- **weight update with error back-propagation(2)**

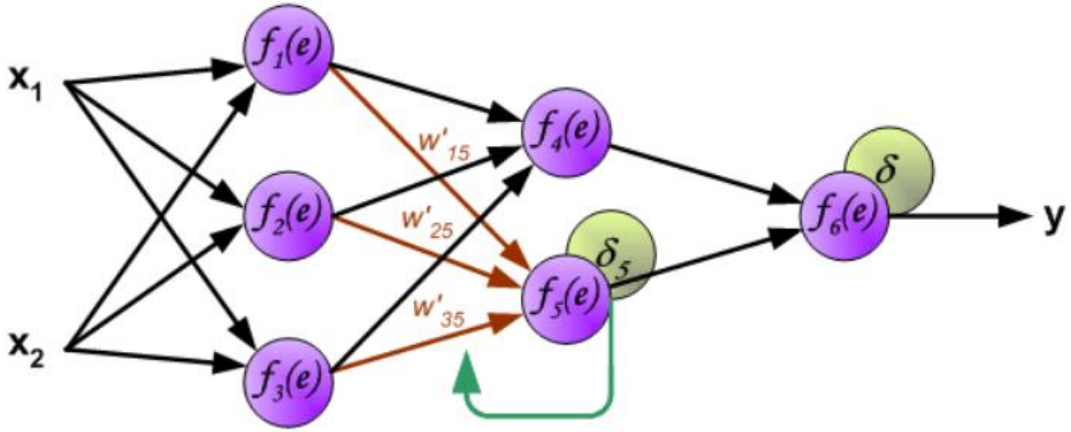


MLP - Learning

- **weight update with error back-propagation(3)**



$$\delta_5 = w_{56}\delta_6 \frac{\partial f_5(e)}{\partial e}$$



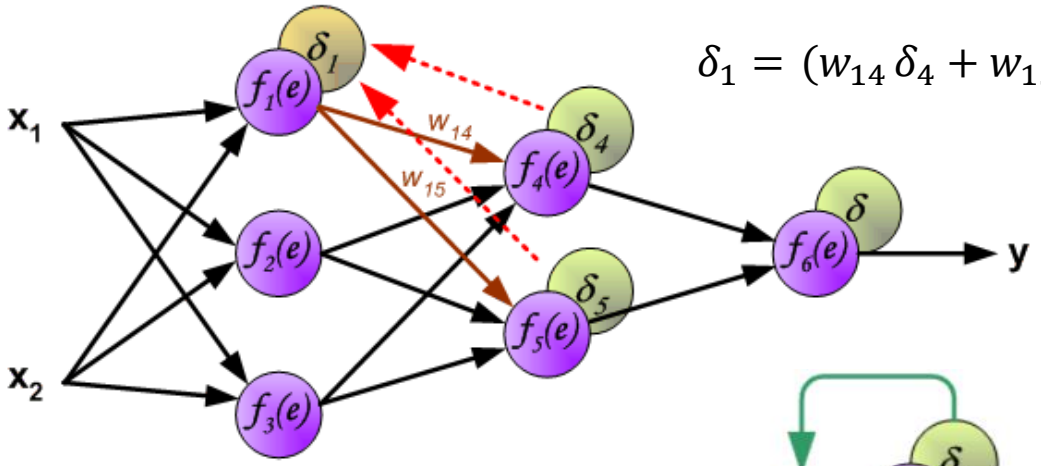
$$w'_{15} = w_{15} - \alpha\delta_5 y_1$$

$$w'_{25} = w_{25} - \alpha\delta_5 y_2$$

$$w'_{35} = w_{35} - \alpha\delta_5 y_3$$

MLP - Learning

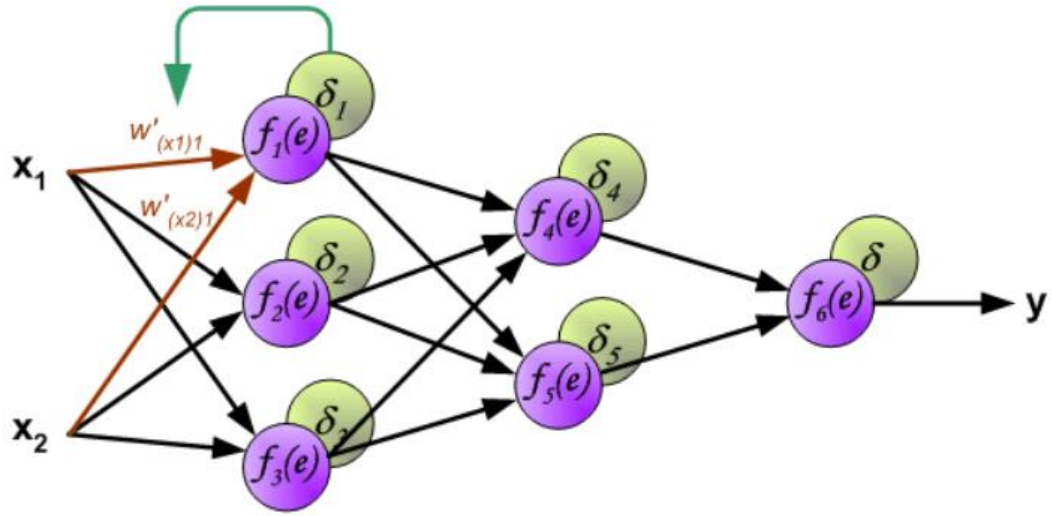
- **weight update with error back-propagation(4)**



$$\delta_1 = (w_{14}\,\delta_4 + w_{15}\delta_5)\frac{\partial f_1(e)}{\partial e}$$

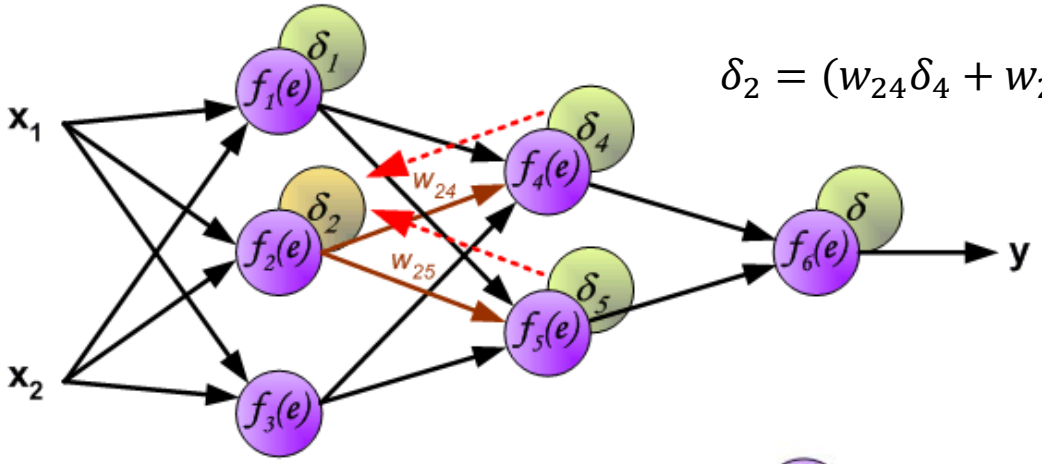
$$w'_{(x1)1} = w_{(x1)1} - \alpha\delta_1 x_1$$

$$w'_{(x2)1} = w_{(x2)1} - \alpha\delta_1 x_2$$



MLP - Learning

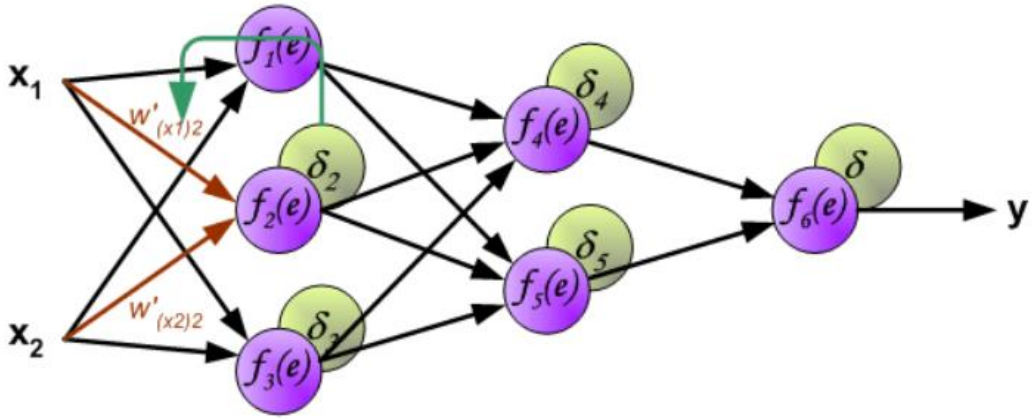
- **weight update with error back-propagation(5)**



$$\delta_2 = (w_{24}\delta_4 + w_{25}\delta_5)\frac{\partial f_2(e)}{\partial e}$$

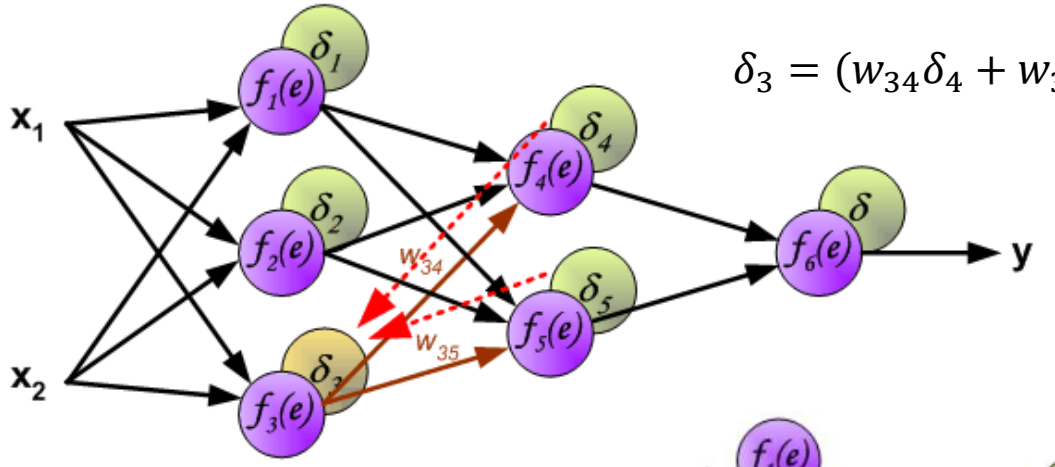
$$w'_{(x1)2} = w_{(x1)2} - \alpha\delta_2 x_1$$

$$w'_{(x2)2} = w_{(x2)2} - \alpha\delta_2 x_2$$



MLP - Learning

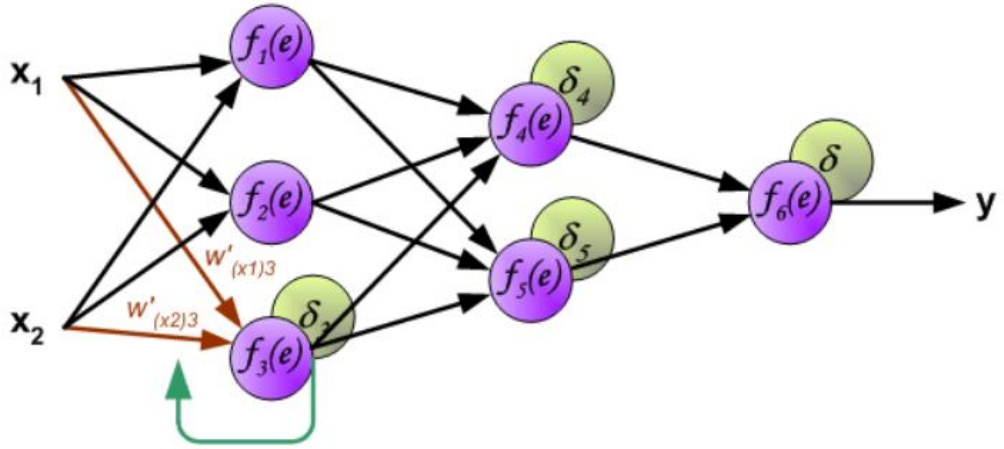
- **weight update with error back-propagation(6)**



$$\delta_3 = (w_{34}\delta_4 + w_{35}\delta_5)\frac{\partial f_3(e)}{\partial e}$$

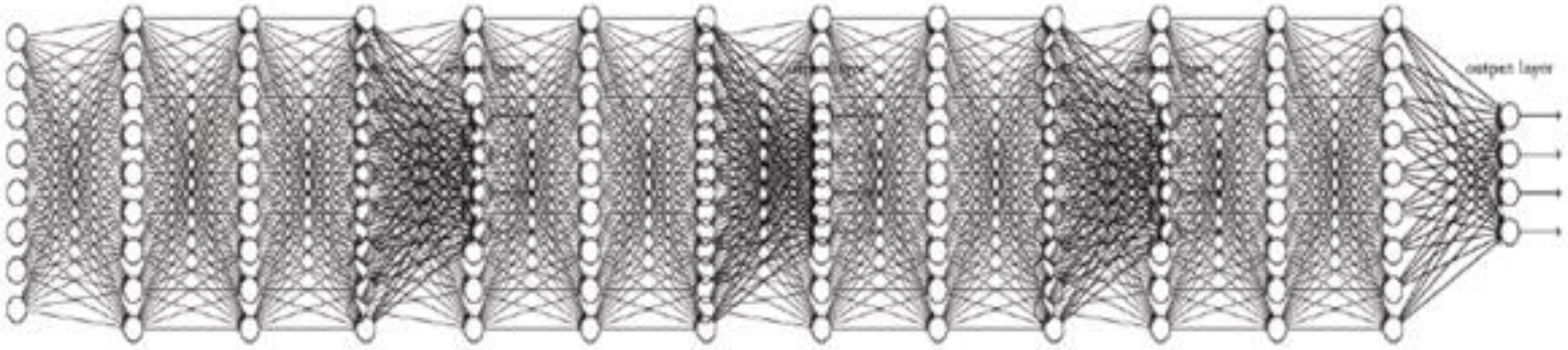
$$w'_{(x1)3} = w_{(x1)3} - \alpha\delta_3 x_1$$

$$w'_{(x2)3} = w_{(x2)3} - \alpha\delta_3 x_2$$



MLP – Breakthrough

- **Difficulties in Deep MLP (Geoffrey Hinton)**



- **We used the wrong type of nonlinearities (sigmoid → ReLU)**
- **We initialized the weights in a stupid way (random → pre-train)**
- **Our labeled datasets were thousands of times too small (BigData)**
- **Our computers were millions of times too slow (GPU)**

References

- Deep Learning, Y. LeCun, Y. Bengio, G. Hinton, Nature 14539, May 2015
- Introduction to Pattern Recognition: A Matlab approach, S. Theodoridis et al., Academic Press, 2010
- Pattern Recognition, Oilseok, Kyobo Bookstore, 2008
- Neural Networks and Deep Learning, M. Nielsen, 2016, http://neuralnetworksanddeeplearning.com/
- From Machine Learning to Deep Learning, Dongmin Kwak et al., DeepCumen, 2015, http://deepcumen.com/
- HUB-AI Community, 2014, http://hub-ai.com/annhmmcrt/17009
- Principles of Backpropagation, M. Bernacki, et al., 2004, http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

THANK YOU!