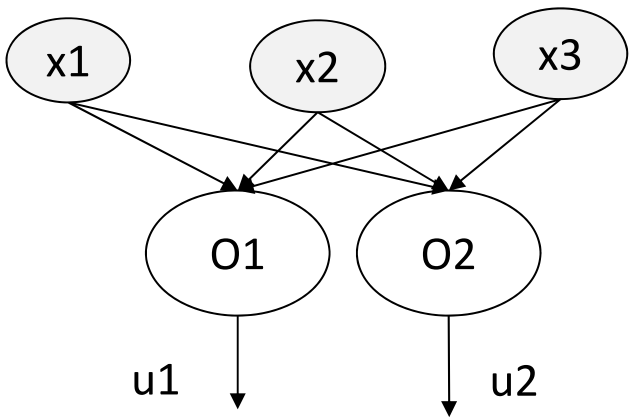
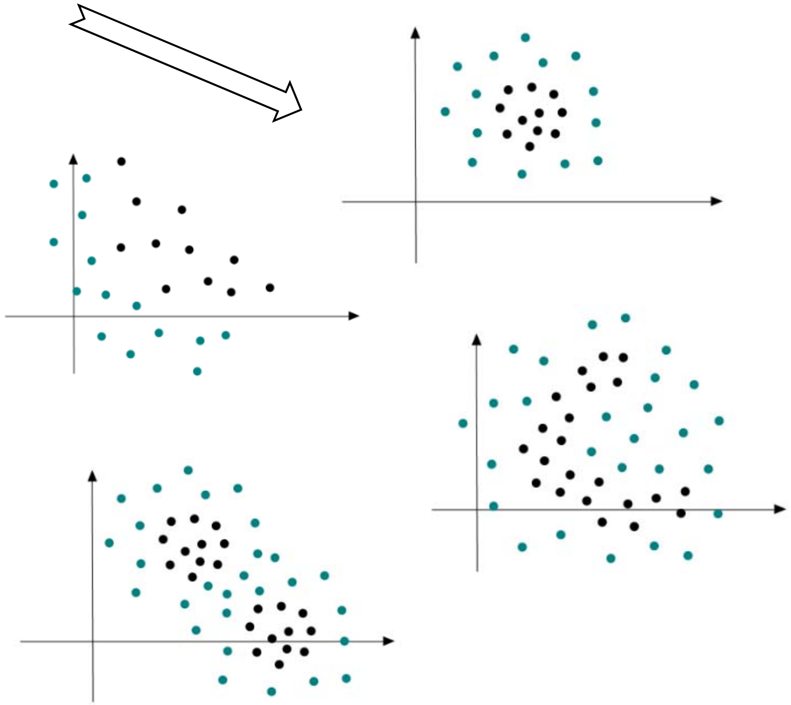
Neuroengineering (I)
3. Multi-layer feed-forward neural networks

- **Scuola di Ingegneria Industriale e dell'Informazione** – Politecnico di Milano
- Prof. Pietro Cerveri

Issues for single layer networks

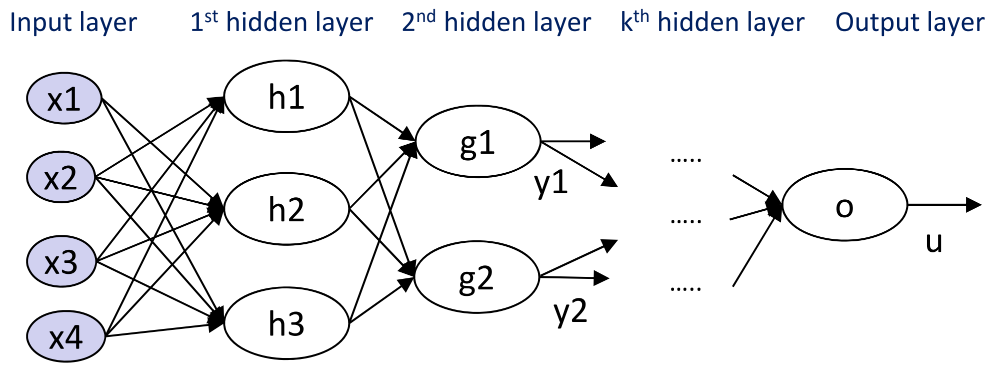
- <u>Only linearly separable problems</u>
  - Delta rule does not converge
  - Delta rule does not minimize the number of mistakes (error in reproducing the output)
- <u>Logsig/Tanh activation</u>
  - Only one predefined non-linearity (no combination)
  - For small P the transform is almost linear
- <u>Adding more output neurons does not help…</u>





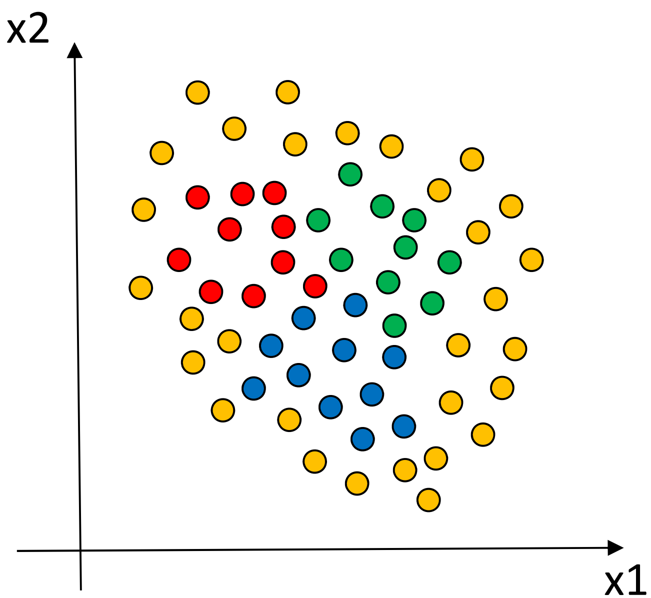
Multi-layer network

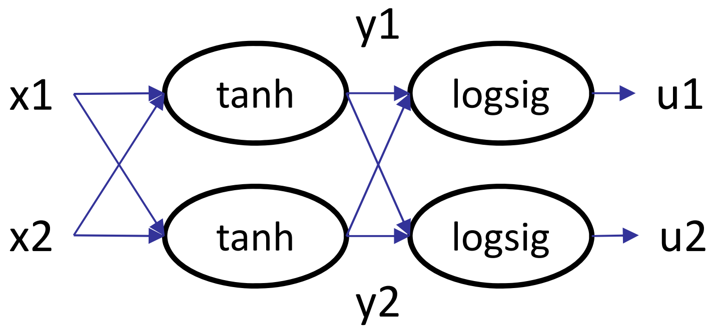
- Adding additional layers between the input and the output



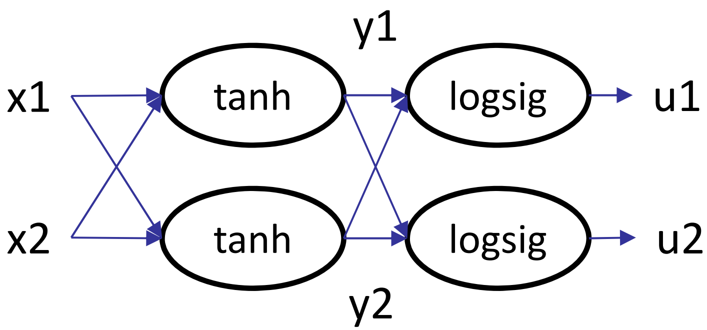
- Sequential composition of non-linearity to allow complex input/output transformation

Multi-class pattern classification



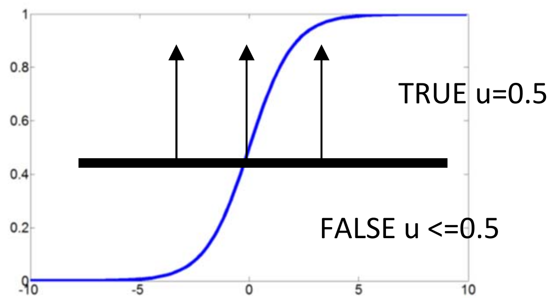


Multi-class pattern classification



y1 = tanh(w11*x1+w12*x2+w13*(-1))
y2 = tanh(w21*x1+w22*x2+w23*(-1))

u1 = logsig(c11*y1+c12*y2+c13*(-1))
u2 = logsig(c21*y1+c22*y2+c23*(-1))

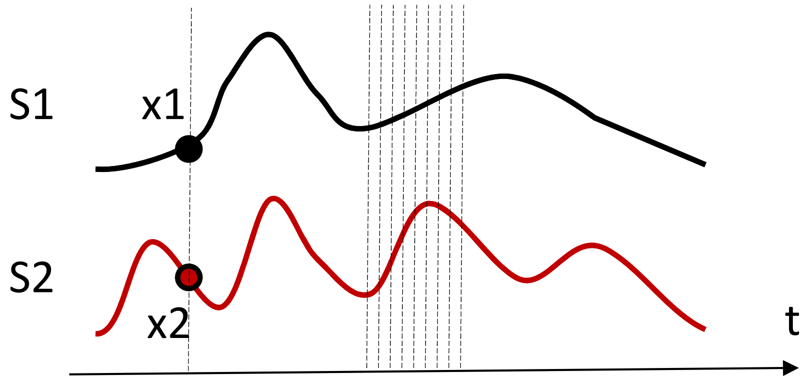


u1=0, u2=0 -> class 1
u1=0, u2=1 -> class 2
u1=1, u2=0 -> class 2
u1=1, u2=1 -> class 4

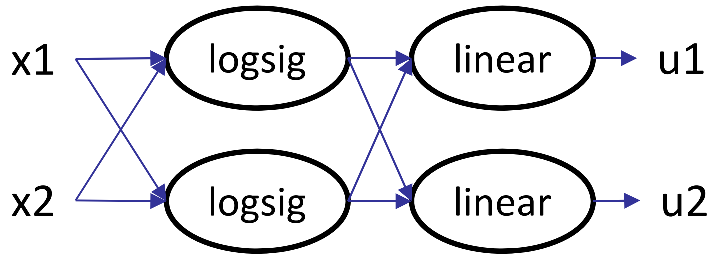
but not only classification .....

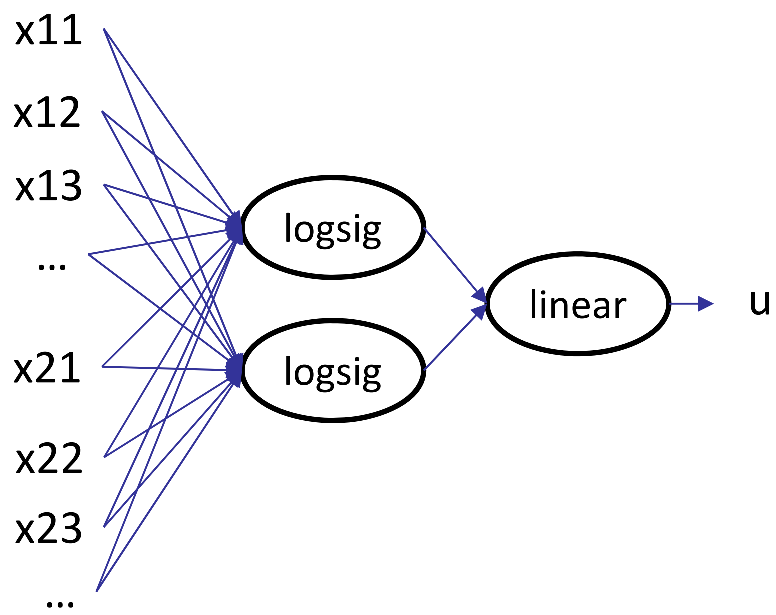
Signal processing

compute sample by sample the
product and sum of the two signals



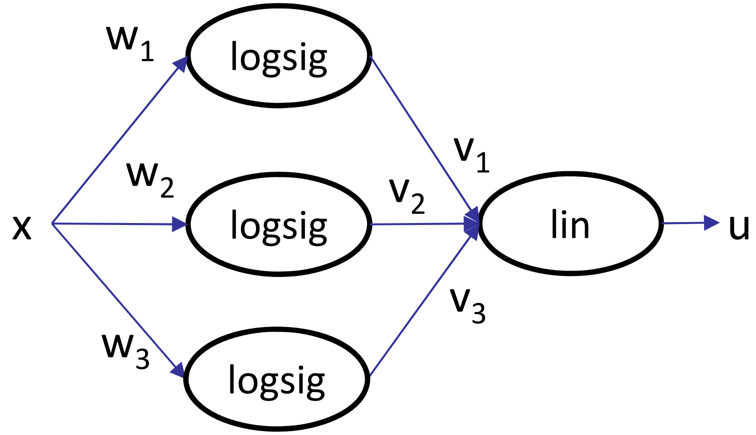
compute period by period the norm of
the difference between the two signals



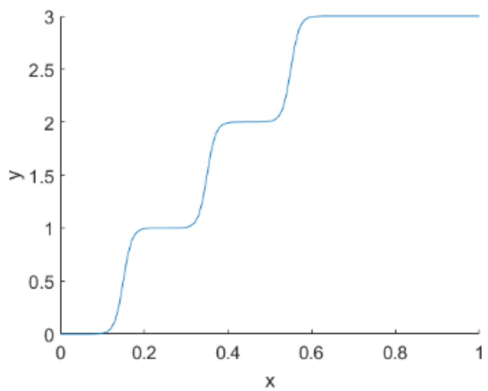


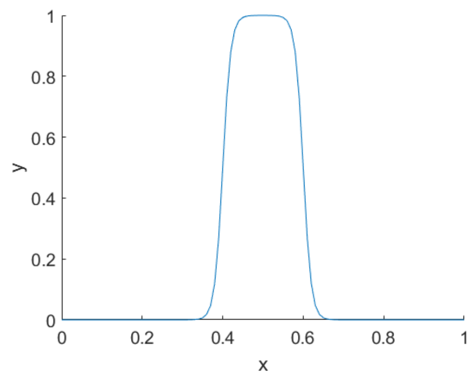
Function modeling

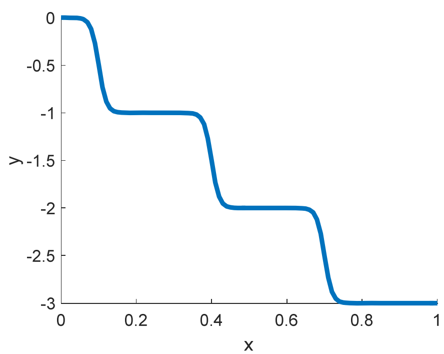
Given any x in the function domain, you get a value in y≡u function co-domain



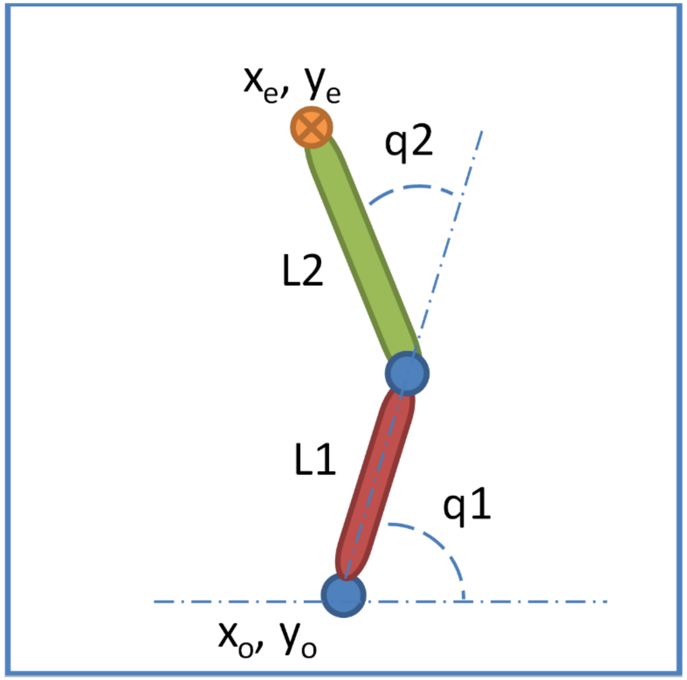
u = v1*logsig(w1*x1+T1*(-1)) +
v2*logsig(w2*x1+T2*(-1)) +
v3*logsig(w3*x1+T3*(-1)) +
T4(*-1)

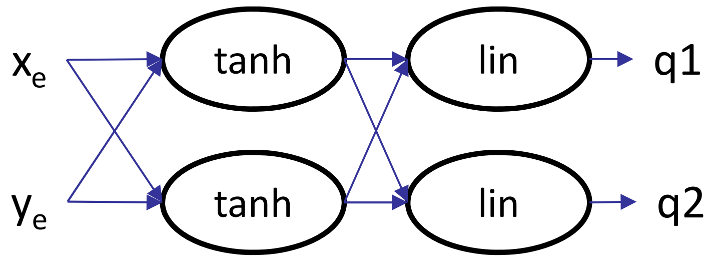






Model parameter mapping





Modeling inverse kinematics

Direct kinematics of the 2-DOF arm (angles -> end-effector)

xe = xo+L1*cos(q1) + L2 * cos(q1+q2)

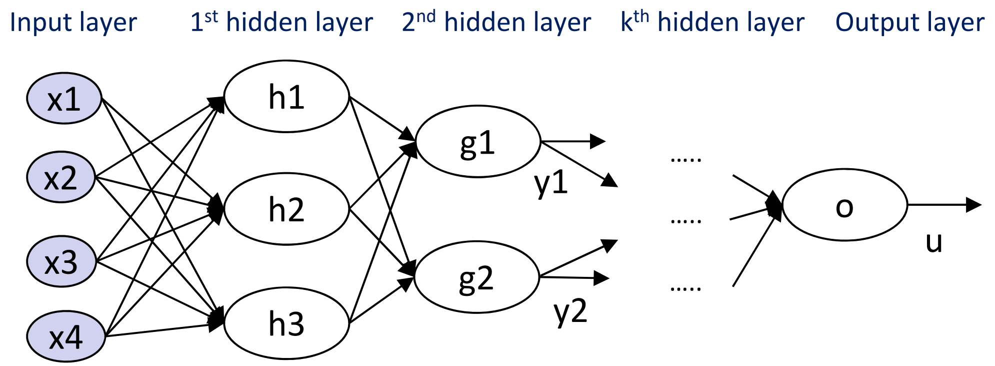
ye = yo+L1*sin(q1) + L2 * sin(q1+q2)

Cost/loss functions tailored to the application

- Cost function used to minimize to prediction error of the network
- Error functions (usually for regression problems)
  - MAE = $\sum(1/n * |t_i - u_i|)$
  - RMSE = $\sqrt{(1/n * \sum (t_i - u_i)^2)}$
  - RMSEL = $\sqrt{(1/n * \sum \log(t_i+1) - \log(u_i+1))^2)}$
- Binary Cross-Entropy (usually for binary classification problems)
  - bCE = $-(t*\log(u) + (1-t)*\log(1-u))$
- Categorical Cross-Entropy (Usually for multi-class classification problems)
  - cCE = $-\Sigma (t_i*\log(u_i))$

Learning in multi-layer network

- Adding additional layers between the input and the output?



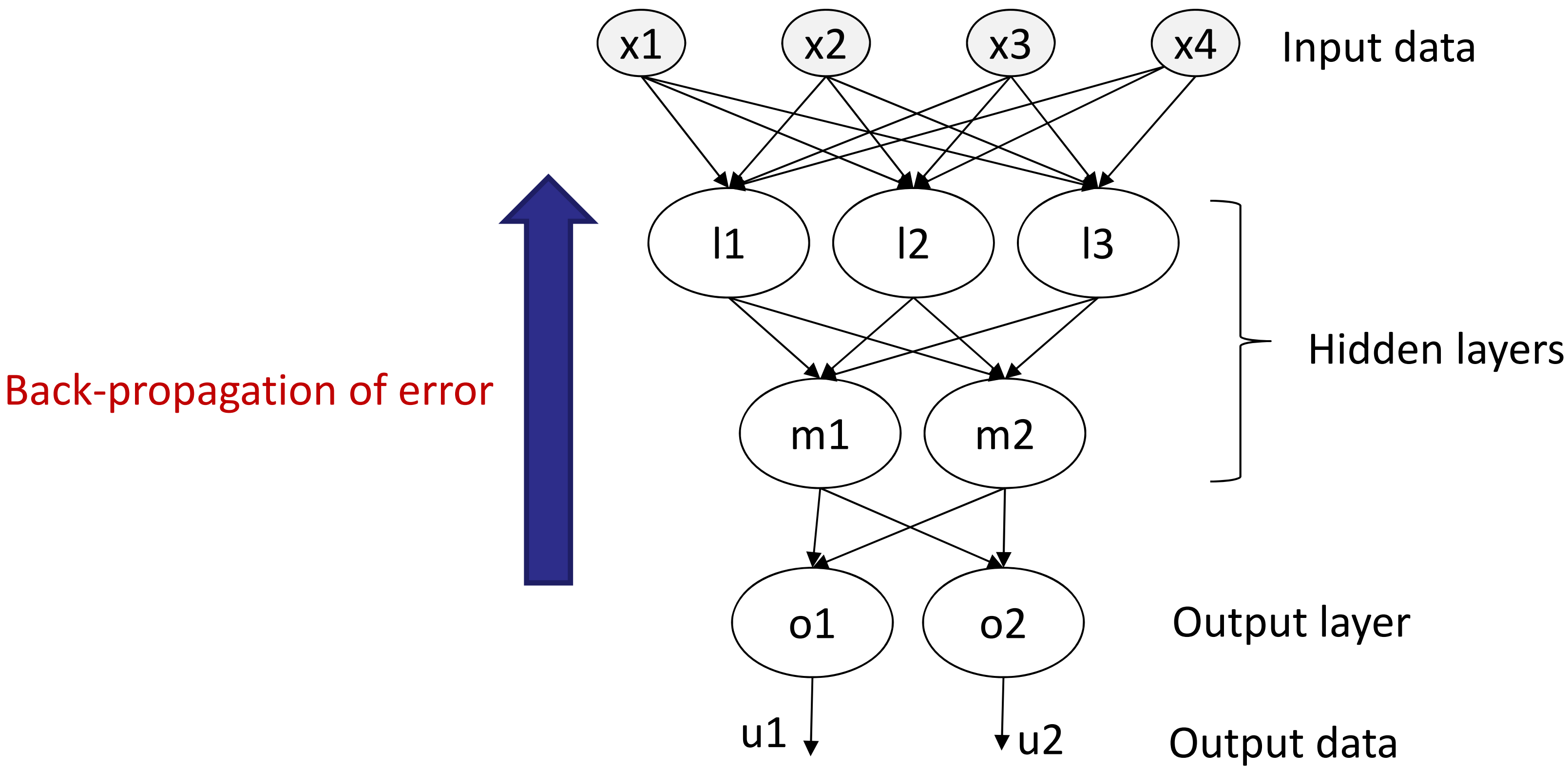
- Is Delta rule still applicable?

- No explicit supervision for the outputs of hidden layer neurons (y_i)

- Solution: Rumelhart, Hinton, Williams 1986 (Nature) Supervised training technique based on the back-propagation of the error which evolves the Delta rule

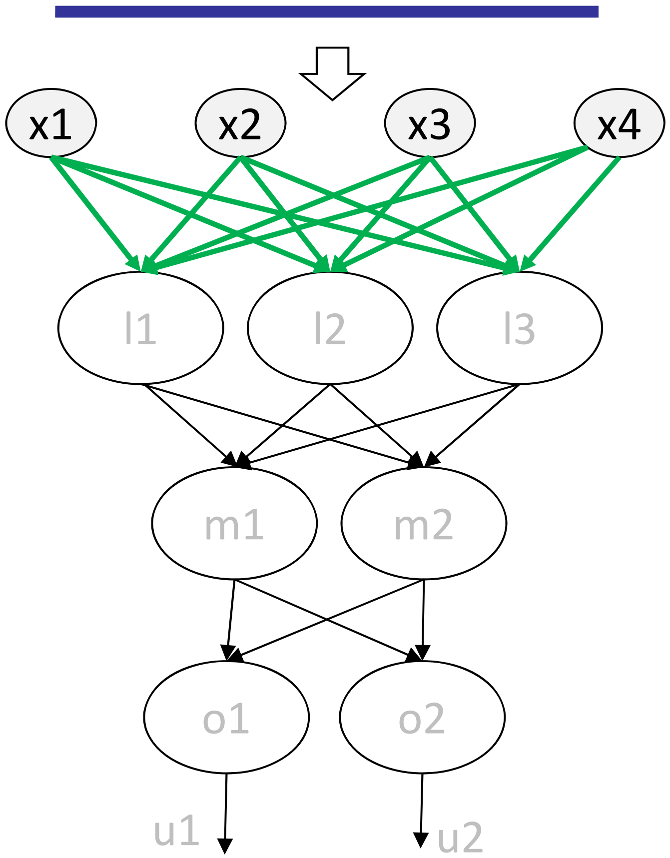
Back-propagation

- Signal error is propagated from output layers back to hidden layers so that all the synaptic weights can be updated



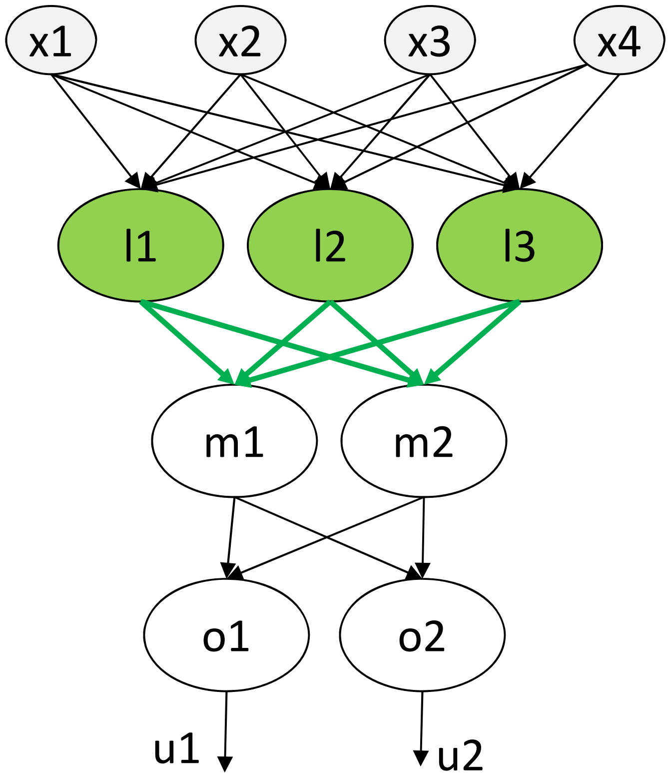
Learning in multi-layer FFNN

1. The training dataset is available (supervised approach). One input pattern enters the network via the input layer



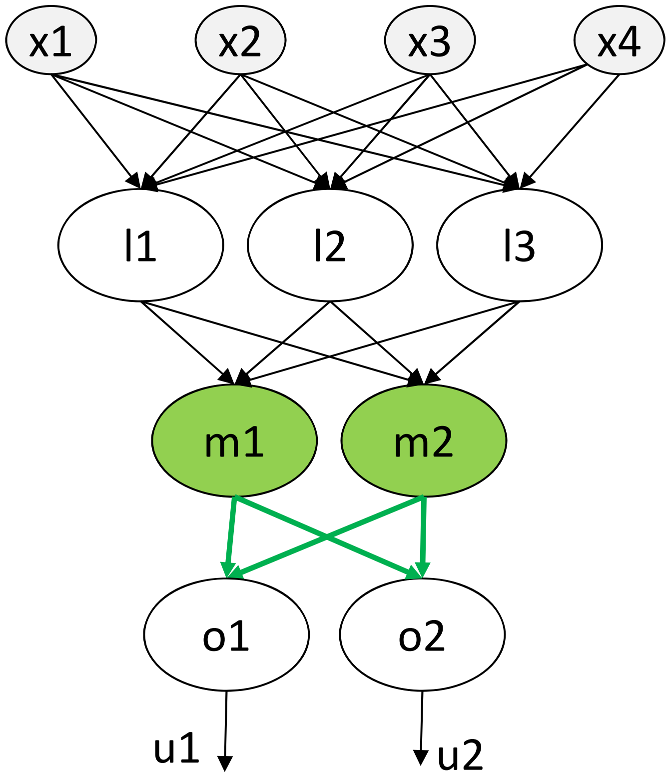
Learning in multi-layer FFNN

1. The training dataset is available (supervised approach). One input pattern enters the network via the input layer
2. Each neuron in the network processes the input pattern with the resulting values steadily “running" through the network, layer by layer, until a result is generated by the output layer



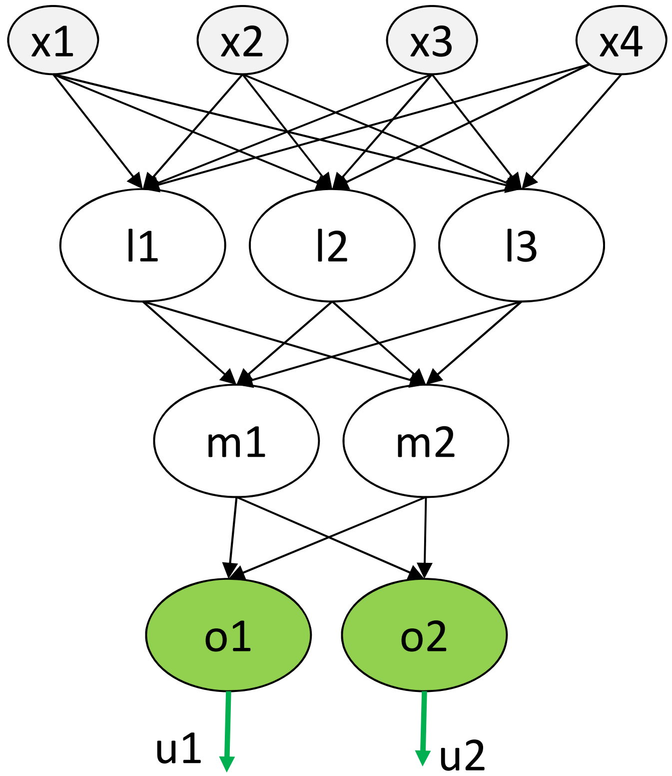
Learning in multi-layer FFNN

1. The training dataset is available (supervised approach). One input pattern enters the network via the input layer
2. Each neuron in the network processes the input pattern with the resulting values steadily “running" through the network, layer by layer, until a result is generated by the output layer



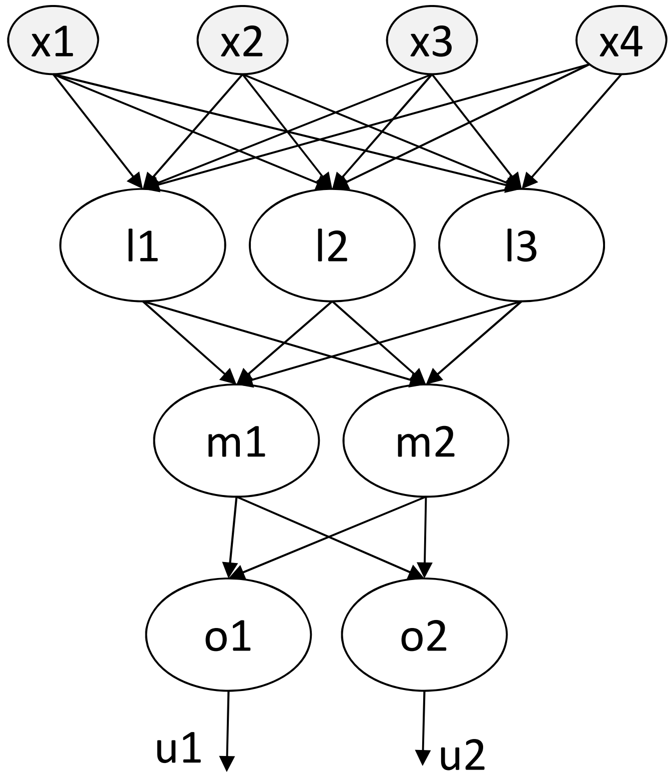
Learning in multi-layer FFNN

1. The training dataset is available (supervised approach). One input pattern enters the network via the input layer
2. Each neuron in the network processes the input pattern with the resulting values steadily “running" through the network, layer by layer, until a result is generated by the output layer



Learning in multi-layer FFNN

1. The training dataset is available (supervised approach). One input pattern enters the network via the input layer
2. Each neuron in the network processes the input pattern with the resulting values steadily “running" through the network, layer by layer, until a result is generated by the output layer
3. The actual output of the network is compared to expected output for that particular input. This results in the *error value (Forward step).*

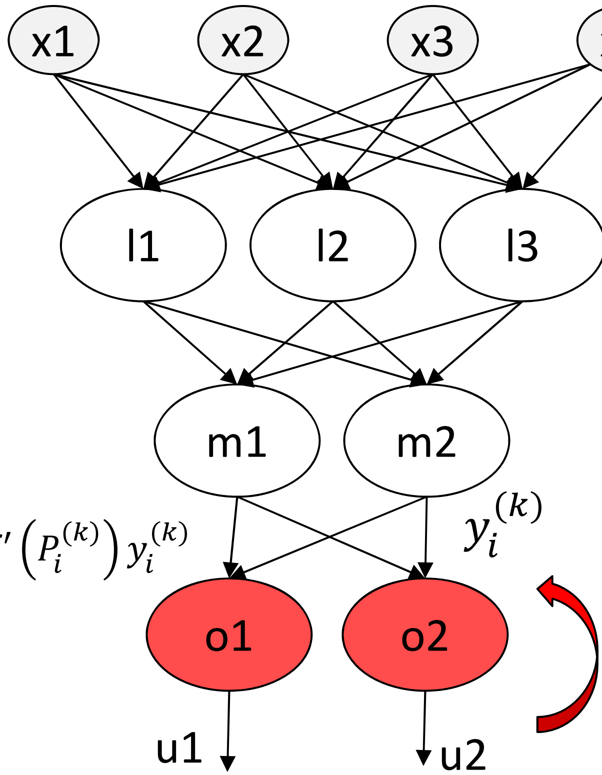


$$\boxed{E^{(k)} = \sum_{i=1}^{N} \frac{1}{2}\left(t_i^{(k)} - u_i^{(k)}\right)^2}$$

Learning in multi-layer FFNN

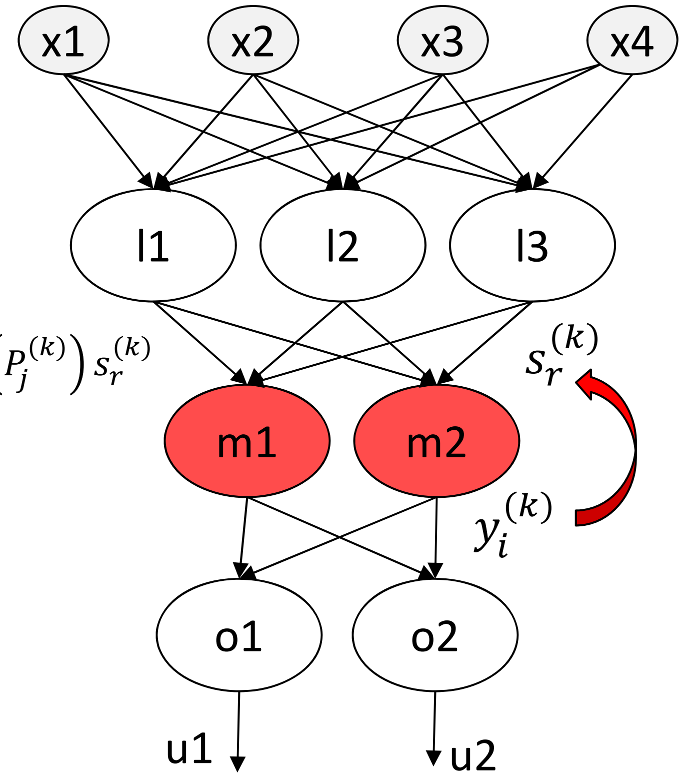
1. The training dataset is available (supervised approach). One input pattern enters the network via the input layer
2. Each neuron in the network processes the input pattern with the resulting values steadily "running" through the network, layer by layer, until a result is generated by the output layer
3. The actual output of the network is compared to expected output for that particular input. This results in the *error value (Forward step).*
4. Apply Delta rule to output layer

$$\Delta w_{ij} = \eta \left(t_i^{(k)} - u_i^{(k)} \right) f'\left(P_i^{(k)} \right) y_i^{(k)}$$



Learning in multi-layer FFNN

$$\Delta c_{jr} = \eta \left(\bar{y}_j^{(k)} - y_j^{(k)} \right) f' \left(P_j^{(k)} \right) s_r^{(k)}$$



5. Apply Delta rule to last hidden layer (m)

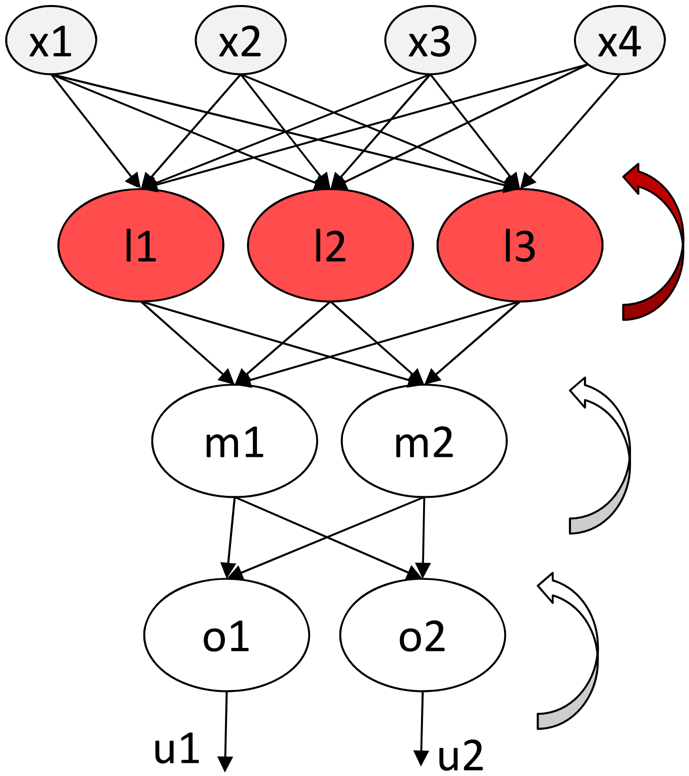
...but $\bar{y}_j^{(k)}$????

We only know the reference for the output layer

Backpropagation algorithm

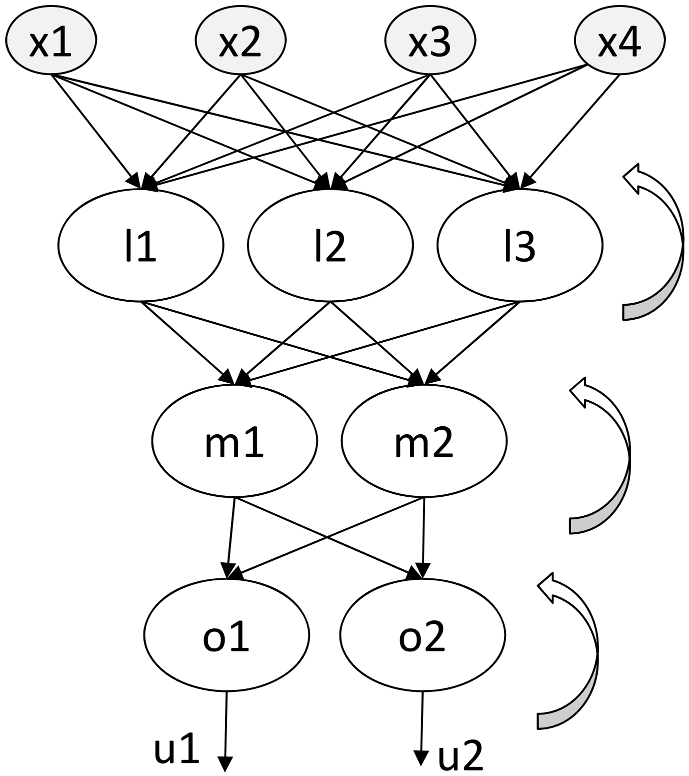
Learning in multi-layer FFNN

1. The training dataset is available (supervised approach). One input pattern enters the network via the input layer
2. Each neuron in the network processes the input pattern with the resulting values steadily "running" through the network, layer by layer, until a result is generated by the output layer
3. The actual output of the network is compared to expected output for that particular input. This results in the *error value* (*Forward step*).
4. Apply Delta rule to output layer
5. Apply **Backpropagation** to last hidden layer and continue up the first hidden layer *(Backward step)*



Learning in multi-layer FFNN

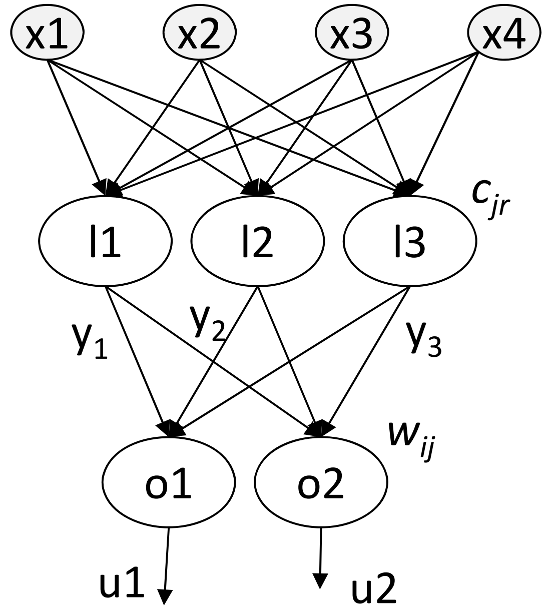
1. The training dataset is available (supervised approach). One input pattern enters the network via the input layer
2. Each neuron in the network processes the input pattern with the resulting values steadily “running" through the network, layer by layer, until a result is generated by the output layer
3. The actual output of the network is compared to expected output for that particular input. This results in the *error value (Forward step).*
4. Apply Delta rule to output layer
5. Apply **Backpropagation** to last hidden layer and continue up the first hidden layer *(Backward step)*
6. Step for all the patterns in the training set (one single iteration)
7. Iterate …….
8. The connection weights in the network are gradually adjusted, working backwards from the output layer, through the hidden layer, and to the input layer, until the correct output is produced.



Coming up with backpropagation rule

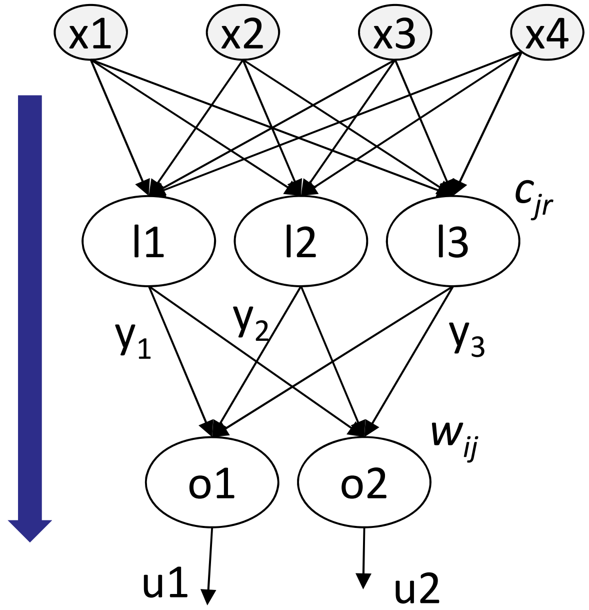
- Assuming a network with a single hidden layer

- R: number of patterns
- x_r: input signals (1:M = 4 + 1, virtual input x5=-1 for thresholds)
- l_j: neurons of the hidden layer with output y_j (1:L = 3 + 1 virtual input y4=-1 for thresholds)
- c_{jr}: weights of the hidden layer
- o_i: neurons of the output layer with output u_i (1:N = 2)
- w_{ij}: weights of the output layer

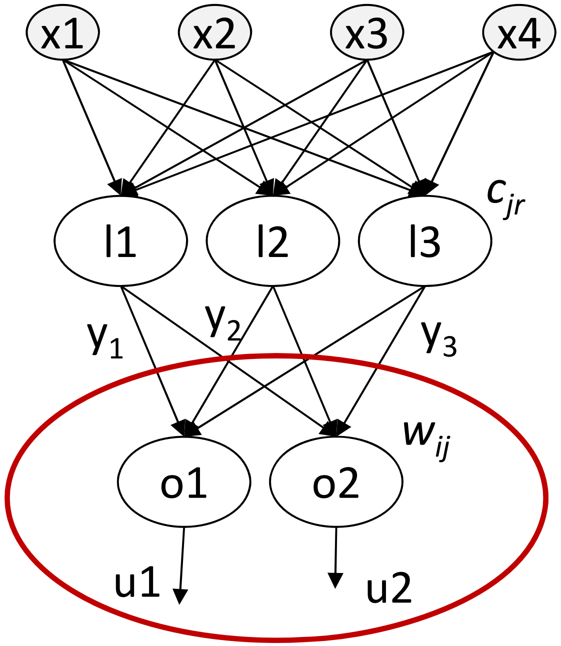


Back-propagation

- Each iteration involves two steps:
  1. Forward step
     - Presentation to the input of the k-th pattern
     - Compute the output signals for output and hidden units (weights are fixed to the initial value)



Back-propagation



- Each iteration involves two steps:
  1. Forward step
     - Presentation to the input of the *k-th* pattern
     - Compute the output signals for output and hidden units (weights are fixed to the initial value)
  2. Backward step
     - Update the weights w_{ij} using delta rule

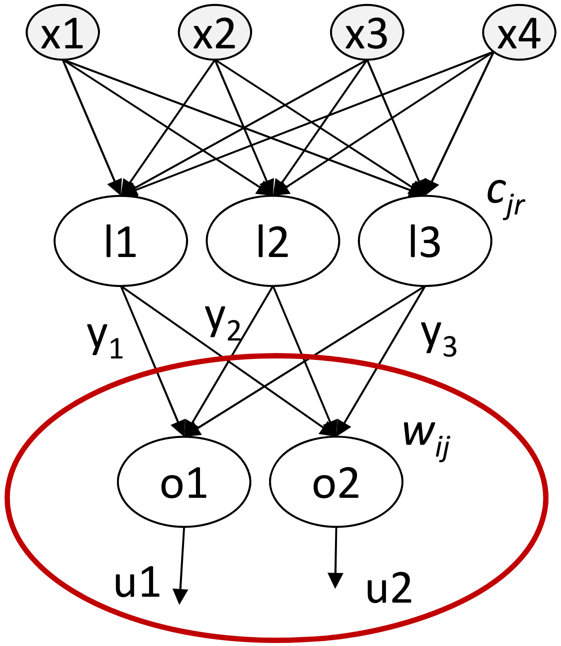
2 neurons u_i with 3+1 inputs y_j

$$\Delta w_{ij} = \eta \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) y_i^{(k)}$$

y=[y_1, y_2, y_3, -1]

$$P_i^{(k)} = \sum w_{ij} y_j^{(k)}$$

Back-propagation



- Each iteration involves two steps:
  1. Forward step
     - Presentation to the input of the k-th pattern
     - Compute the output signals for output and hidden units (weights are fixed to the initial value)
  2. Backward step
     - Update the weights w_{ij} using delta rule

2 neurons u_i with 3+1 inputs y_j

$$\Delta w_{ij} = \eta \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) y_i^{(k)}$$

$\mathbf{y}=[y_1, y_2, y_3, -1]$

$$P_1^{(k)} = \sum w_{1j} y_j^{(k)} \qquad P_2^{(k)} = \sum w_{2j} y_j^{(k)}$$

$$\Delta w_{11} = \eta \left(t_1^{(k)} - u_1^{(k)}\right) f'\left(P_1^{(k)}\right) y_1^{(k)} \qquad \Delta w_{21} = \eta \left(t_2^{(k)} - u_2^{(k)}\right) f'\left(P_2^{(k)}\right) y_1^{(k)}$$

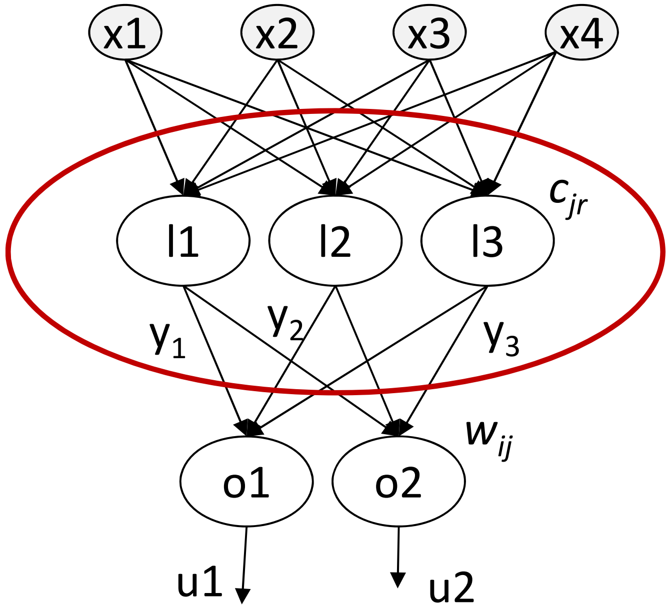
$$\Delta w_{12} = \eta \left(t_1^{(k)} - u_1^{(k)}\right) f'\left(P_1^{(k)}\right) y_2^{(k)} \qquad \Delta w_{22} = \eta \left(t_2^{(k)} - u_2^{(k)}\right) f'\left(P_2^{(k)}\right) y_2^{(k)}$$

$$\Delta w_{13} = \eta \left(t_1^{(k)} - u_1^{(k)}\right) f'\left(P_1^{(k)}\right) y_3^{(k)} \qquad \Delta w_{23} = \eta \left(t_2^{(k)} - u_2^{(k)}\right) f'\left(P_2^{(k)}\right) y_3^{(k)}$$

$$\Delta w_{14} = \eta \left(t_1^{(k)} - u_1^{(k)}\right) f'\left(P_1^{(k)}\right) (-1) \qquad \Delta w_{24} = \eta \left(t_2^{(k)} - u_2^{(k)}\right) f'\left(P_2^{(k)}\right) (-1)$$

thresholds

Back-propagation



- Each iteration involves two steps:
  1. Forward step
     - Presentation to the input of the *k*-th pattern
     - Compute the output signals for output and hidden units (weights are fixed to the initial value)
  2. Backward step
     - Update the weights w_{ij} using delta rule
     - Update the weights c_{jr} using delta rule

$$\Delta c_{jr} = \eta \left(\bar{y}_j^{(k)} - y_j^{(k)}\right) f'\left(P_j^{(k)}\right) x_r^{(k)}$$

desired output of the *j*-th hidden neuron

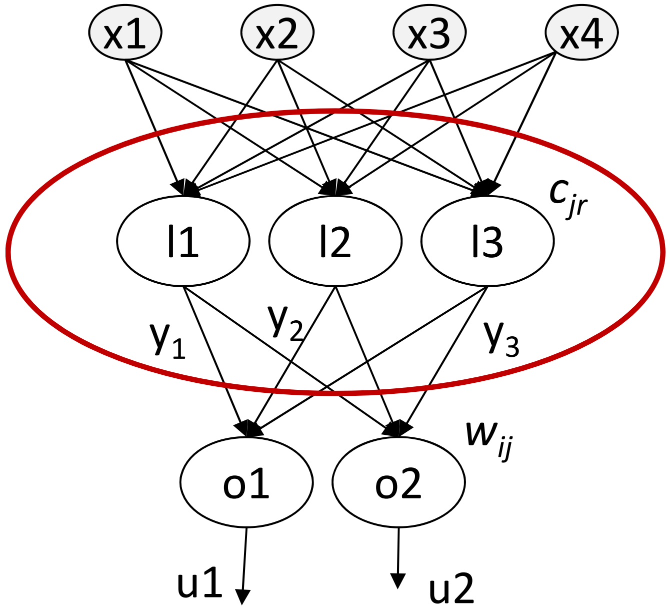
$$P_j^{(k)} = \sum c_{jr} x_r^{(k)}$$

3 neurons y_i with 4 + 1 inputs x_j

$\mathbf{x}=[x_1, x_2, x_3, x_4, -1]$

Back-propagation

- Each iteration involves two steps:
  1. Forward step
     - Presentation to the input of the k-th pattern
     - Compute the output signals for output and hidden units (weights are fixed to the initial value)
  2. Backward step
     - Update the weights w_{ij} using delta rule
     - Update the weights c_{jr} using delta rule



3 neurons y_i with 4 + 1 inputs x_j

$$\Delta c_{jr} = \eta \left(\bar{y}_j^{(k)} - y_j^{(k)}\right) f'\left(P_j^{(k)}\right) x_r^{(k)}$$

$\mathbf{x}=[x_1, x_2, x_3, x_4, -1]$

$$P_1^{(k)} = \sum c_{1r} x_r^{(k)} \qquad P_2^{(k)} = \sum c_{2r} x_r^{(k)} \qquad P_3^{(k)} = \sum c_{3r} x_r^{(k)}$$

$$\Delta c_{11} = \eta \left(\bar{y}_1^{(k)} - y_1^{(k)}\right) f'\left(P_1^{(k)}\right) x_1^{(k)} \qquad \Delta c_{21} = \eta \left(\bar{y}_2^{(k)} - y_2^{(k)}\right) f'\left(P_2^{(k)}\right) x_1^{(k)} \qquad \Delta c_{31} = \eta \left(\bar{y}_3^{(k)} - y_3^{(k)}\right) f'\left(P_3^{(k)}\right) x_1^{(k)}$$

$$\Delta c_{12} = \eta \left(\bar{y}_1^{(k)} - y_1^{(k)}\right) f'\left(P_1^{(k)}\right) x_2^{(k)} \qquad \Delta c_{22} = \eta \left(\bar{y}_2^{(k)} - y_2^{(k)}\right) f'\left(P_2^{(k)}\right) x_2^{(k)} \qquad \Delta c_{32} = \eta \left(\bar{y}_3^{(k)} - y_3^{(k)}\right) f'\left(P_3^{(k)}\right) x_2^{(k)}$$

$$\Delta c_{13} = \eta \left(\bar{y}_1^{(k)} - y_1^{(k)}\right) f'\left(P_1^{(k)}\right) x_3^{(k)} \qquad \Delta c_{23} = \eta \left(\bar{y}_2^{(k)} - y_2^{(k)}\right) f'\left(P_2^{(k)}\right) x_3^{(k)} \qquad \Delta c_{33} = \eta \left(\bar{y}_3^{(k)} - y_3^{(k)}\right) f'\left(P_3^{(k)}\right) x_3^{(k)}$$

$$\Delta c_{14} = \eta \left(\bar{y}_1^{(k)} - y_1^{(k)}\right) f'\left(P_1^{(k)}\right) x_4^{(k)} \qquad \Delta c_{24} = \eta \left(\bar{y}_2^{(k)} - y_2^{(k)}\right) f'\left(P_2^{(k)}\right) x_4^{(k)} \qquad \Delta c_{34} = \eta \left(\bar{y}_3^{(k)} - y_3^{(k)}\right) f'\left(P_3^{(k)}\right) x_4^{(k)}$$

$$\Delta c_{15} = \eta \left(\bar{y}_1^{(k)} - y_1^{(k)}\right) f'\left(P_1^{(k)}\right) (-1) \qquad \Delta c_{25} = \eta \left(\bar{y}_2^{(k)} - y_2^{(k)}\right) f'\left(P_2^{(k)}\right) (-1) \qquad \Delta c_{35} = \eta \left(\bar{y}_3^{(k)} - y_3^{(k)}\right) f'\left(P_3^{(k)}\right) (-1)$$

Single pattern

Action potential of the hidden neurons

Action potential of the output neurons

$$\Delta c_{jr} = f\left(\frac{\partial E}{\partial c_{jr}}\right) =$$

$$\frac{\partial E}{\partial c_{jr}} = \frac{\partial E}{\partial P_j^H}\frac{\partial P_j^H}{\partial c_{jr}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial P_j^H}\frac{\partial P_j^H}{\partial c_{jr}} = \sum_i^N \left(\frac{\partial E}{\partial P_i^O}\frac{\partial P_i^O}{\partial y_j}\right)\frac{\partial y_j}{\partial P_j^H}\frac{\partial P_j^H}{\partial c_{jr}} =$$

$$\sum_{i=1}^N \left(\frac{\partial E}{\partial u_i}\frac{\partial u_i}{\partial P_i^O}\frac{\partial P_i^O}{\partial y_j}\right)\frac{\partial y_j}{\partial P_j^H}\frac{\partial P_j^H}{\partial c_{jr}} = \sum_{i=1}^N \left((t_i - u_i) f'\left(P_i^O\right) w_{ij}\right)\frac{\partial y_j}{\partial P_j^H}\frac{\partial P_j^H}{\partial c_{jr}} =$$

$$\underline{\sum_{i=1}^N \left((t_i - u_i) f'\left(P_i^O\right) w_{ij}\right) f'\left(P_j^H\right) x_r}$$

$$\left(\bar{y}_j - y_j\right) = \sum_{i=1}^N \left((t_i - u_i) f'(P_i^O) w_{ij}\right)$$

For batch updating with R patterns

$$\Delta c_{jr} = \eta \sum_{k=1}^{R} \left(\overline{y}_j^{(k)} - y_j^{(k)} \right) f'\left(P_j^{H(k)}\right) x_r^{(k)} =$$

$$\eta \sum_{k=1}^{R} \left(\sum_{i=1}^{N} \left(\left(t_i^{(k)} - u_i^{(k)} \right) f'\left(P_i^{O(k)}\right) w_{ij} \right) f'\left(P_j^{H(k)}\right) x_r^{(k)} \right)$$

$$errO = \left(t_i - u_i\right) * f'\left(P_i^O\right)$$

$$errH_{ij} = f'\left(P_j^H\right) * w_{ij} * errO_i$$

$$P_i^{O(k)} = \sum_j w_{ij} y_j^{(k)} \qquad P_j^{H(k)} = \sum_r c_{jr} x_r^{(k)}$$

Generalized back-propagation

$w_{ij}^{(l)}$ with *l*:1,….*L* where *l* is the index of the layer

$$\Delta w_{ij}^{(l)} = \eta \sum_{k=1}^{R} \delta_i^{(l),(k)} y_j^{(l-1),(k)}$$

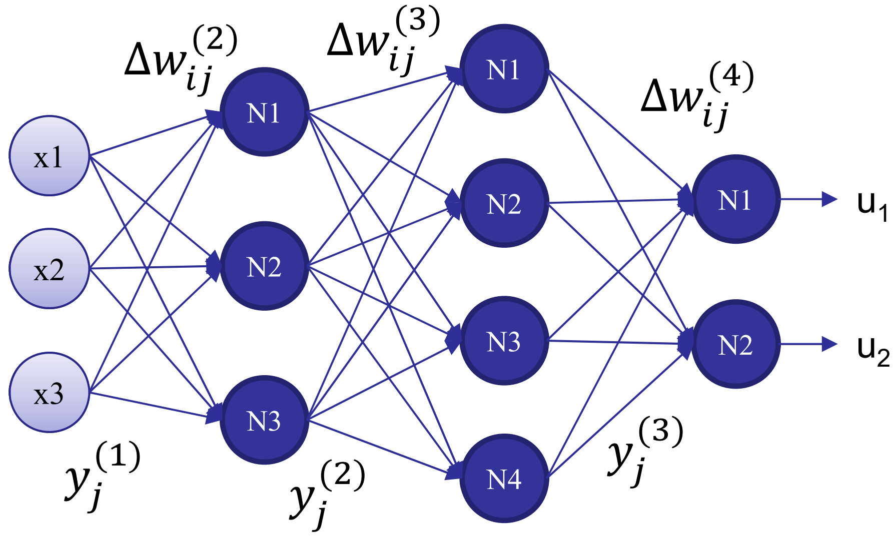
$y_j^{(l-1),(k)}$ is the output signal of the j-th neuron of the (l-1)-th layer in correspondence of the k-th pattern of the training set

$$\delta_i^{(l),(k)} = \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) \qquad \text{if} \quad l = L$$

$$\delta_i^{(l),(k)} = \sum_{r=1}^{M_{l+1}} \left(\delta_r^{(l+1),(k)} w_{ri}^{(l+1)}\right) f'\left(P_i^{(k)}\right) \quad \text{if} \quad l < L$$

M_{l+1} is the number of neurons in the (*l*+1)-th layer

L1 input	L2 1° hidden layer	L3 2° hidden layer	L4 output

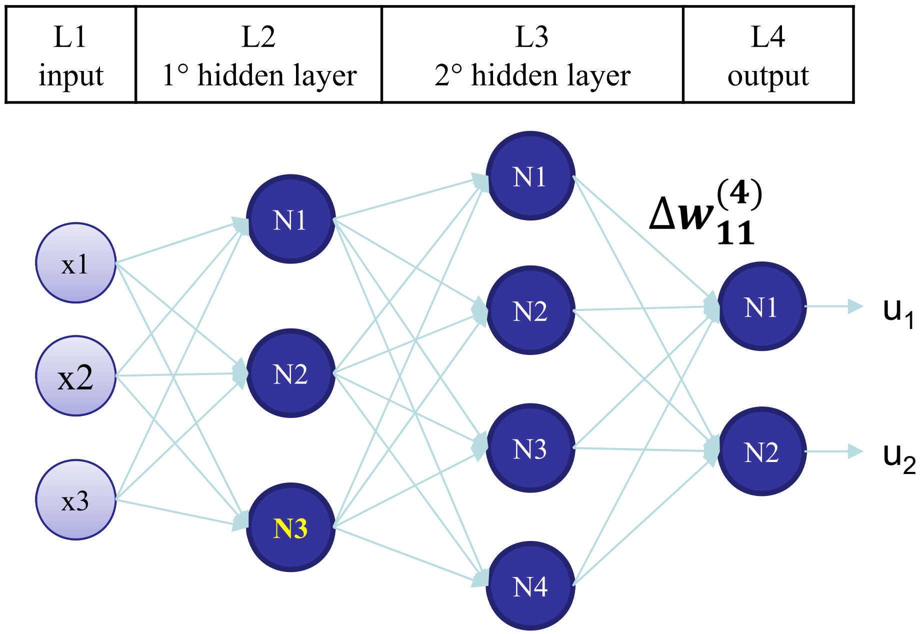


$$\Delta w_{ij}^{(l)} = \eta \sum_{k=1}^{R} \delta_i^{(l),(k)} y_j^{(l-1),(k)}$$

$y_j^{(l-1),(k)}$ is the output signal of the j-th neuron of the (l-1)-th layer in correspondence of the k-th pattern of the training set

$$\delta_i^{(l),(k)} = \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(l)(k)}\right) \qquad \text{if} \quad l = L$$

$$\delta_i^{(l),(k)} = \sum_{r=1}^{M_{l+1}} \left(\delta_r^{(l+1),(k)} w_{ri}^{(l+1)}\right) f'\left(P_i^{(l)(k)}\right) \qquad \text{if} \quad l < L$$



$$\Delta w_{ij}^{(l)} = \eta \sum_{k=1}^{R} \delta_i^{(l),(k)} y_j^{(l-1),(k)}$$

$y_j^{(l-1),(k)}$ is the output signal of the j-th neuron of the (l-1)-th layer in correspondence of the k-th pattern of the training set

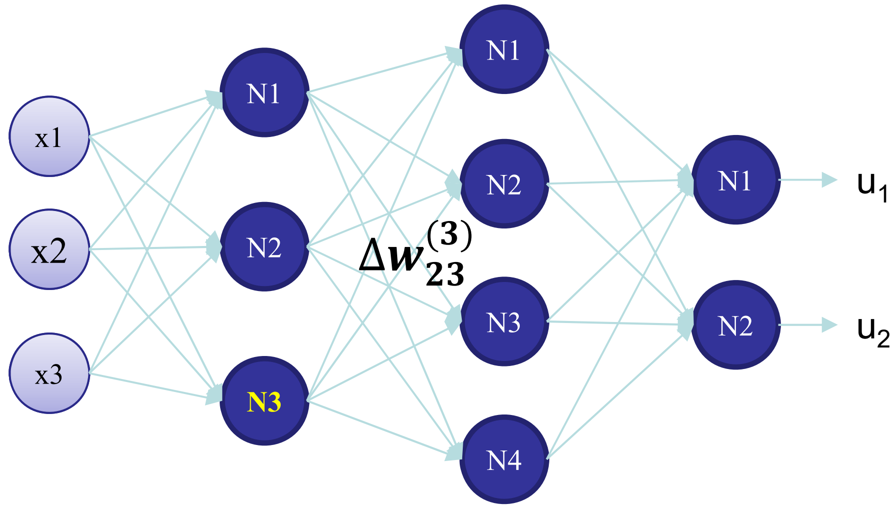
$$\delta_i^{(l),(k)} = \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(l)(k)}\right) \qquad \text{if} \quad l = L$$

$$\delta_i^{(l),(k)} = \sum_{r=1}^{M_{l+1}} \left(\delta_r^{(l+1),(k)} w_{ri}^{(l+1)}\right) f'\left(P_i^{(l)(k)}\right) \qquad \text{if} \quad l < L$$

$$R = 1$$

$$\Delta \boldsymbol{w}_{\mathbf{1,1}}^{(\mathbf{4})} = (t_1 - u_1) f'\left(P_1^{(4)}\right) y_1^{(3)}$$

L1 input	L2 1° hidden layer	L3 2° hidden layer	L4 output



$$\Delta w_{ij}^{(l)} = \eta \sum_{k=1}^{R} \delta_i^{(l),(k)} y_j^{(l-1),(k)}$$

$y_j^{(l-1),(k)}$ is the output signal of the j-th neuron of the (l-1)-th layer in correspondence of the k-th pattern of the training set

$$\delta_i^{(l),(k)} = \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(l)(k)}\right) \qquad \text{if} \quad l = L$$

$$R = 1$$

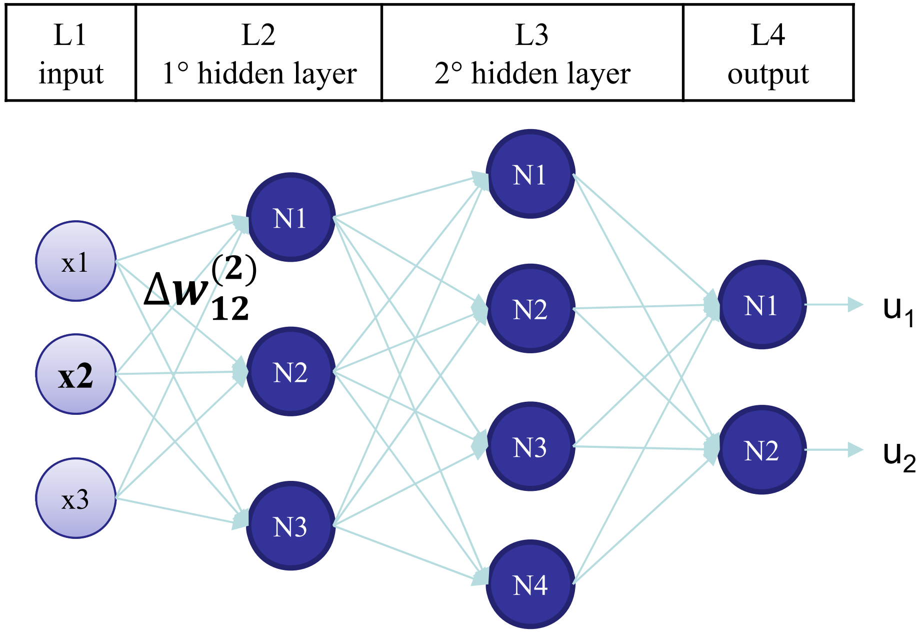
$$\delta_i^{(l),(k)} = \sum_{r=1}^{M_{l+1}} \left(\delta_r^{(l+1),(k)} w_{ri}^{(l+1)}\right) f'\left(P_i^{(l)(k)}\right) \qquad \text{if} \quad l < L$$

$$\delta_i^{(4)} = (t_i - u_i) f'\left(P_i^{(4)}\right)$$

$$\delta_i^{(3)} = \textstyle\sum_{r=1}^{2} \left(\delta_r^{(4)} w_{r,i}^{(4)}\right) f'\left(P_2^{(3)}\right)$$

$$\Delta \boldsymbol{w}_{2,3}^{(3)} = \delta_2^{(3)} y_3^{(2)} \qquad \Delta \boldsymbol{w}_{2,3}^{(3)} = \left(\sum_{r=1}^{2} \left(\delta_r^{(4)} w_{r,2}^{(4)}\right)\right) f'\left(P_2^{(3)}\right) y_3^{(2)}$$

$$\Delta \boldsymbol{w}_{2,3}^{(3)} = \left((t_1 - u_1) f'\left(P_1^{(4)}\right) w_{1,2}^{(4)} + (t_2 - u_2) f'\left(P_2^{(4)}\right) w_{2,2}^{(4)}\right) f'\left(P_2^{(3)}\right) y_3^{(2)}$$



$$\Delta w_{ij}^{(l)} = \eta \sum_{k=1}^{R} \delta_i^{(l),(k)} y_j^{(l-1),(k)}$$

$y_j^{(l-1),(k)}$ is the output signal of the j-th neuron of the (l-1)-th layer in correspondence of the k-th pattern of the training set

$$\delta_i^{(l),(k)} = \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(l)(k)}\right) \quad \text{if} \quad l = L$$

$$R = 1$$

$$\delta_i^{(2)} = \sum_{r=1}^{4} \left(\delta_r^{(3)} w_{r,i}^{(3)}\right) f'\left(P_i^{(2)}\right)$$

$$\delta_i^{(l),(k)} = \sum_{r=1}^{M_{l+1}} \left(\delta_r^{(l+1),(k)} w_{ri}^{(l+1)}\right) f'\left(P_i^{(l)(k)}\right) \quad \text{if} \quad l < L$$

$$\Delta \boldsymbol{w}_{\mathbf{1,2}}^{(\mathbf{2})} = \sum_{r=1}^{4} \left(\delta_r^{(3)} w_{r,1}^{(3)}\right) f'\left(P_1^{(2)}\right) y_2^{(1)} = \sum_{r=1}^{4} \left(\delta_r^{(3)} w_{r,1}^{(3)}\right) f'\left(P_1^{(2)}\right) x_2$$

Procedure for a batch update

Start from *l*=*L* (output layer) and compute $\delta_i^{(L),(k)}$

Move to *l*=*L*-1 and compute $\delta_i^{(L-1),(k)}$ as a function of $\delta_i^{(L),(k)}$

Continue until *l*=2 $\delta_i^{(l),(k)}$

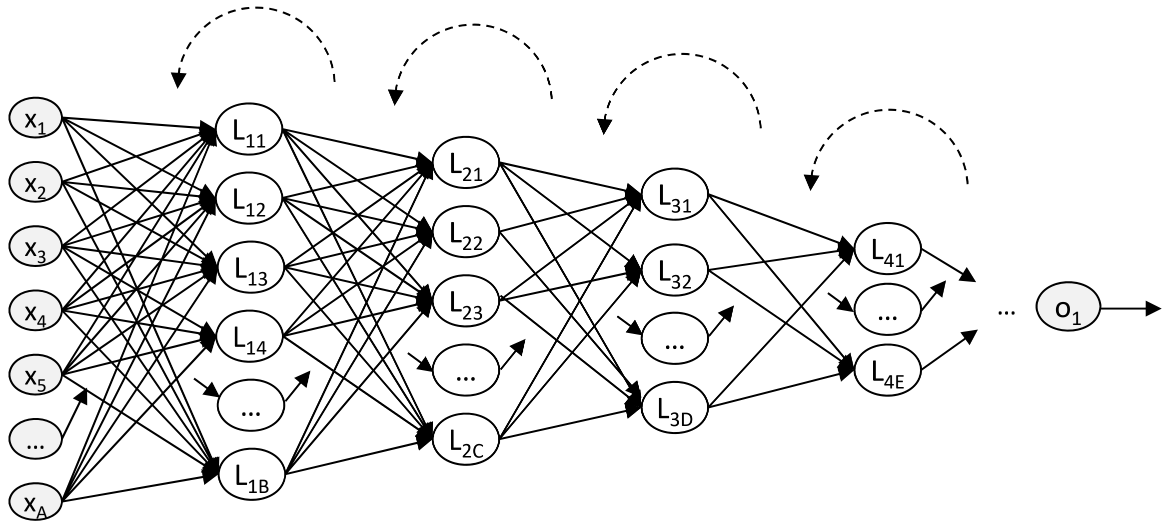
Compute layer-by-layer the weight updates

$$\Delta w_{ij}^{(l)} = \eta \sum_{k=1}^{R} \delta_i^{(l),(k)} y_j^{(l-1),(k)}$$

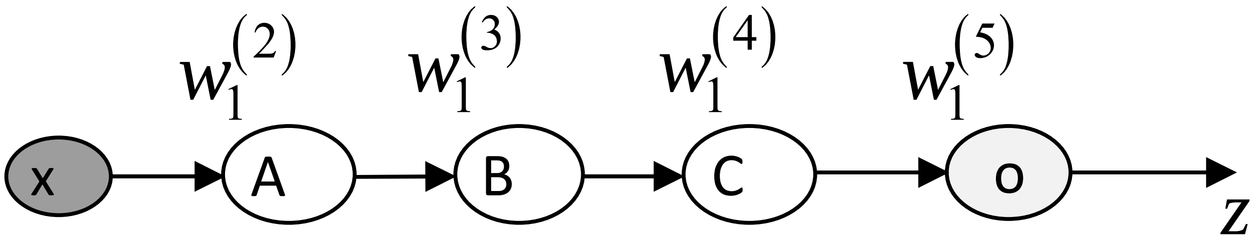
Troubles when training multi-layer fully connected NN

Vanishing gradient problem in the back-propagation

$$0 \leftarrow \frac{\partial E}{\partial w_{ij}^{(l-k)}} < \quad \frac{\partial E}{\partial w_{ij}^{(l-1)}} < \quad \frac{\partial E}{\partial w_{ij}^{(l)}} < \quad \frac{\partial E}{\partial w_{ij}^{(l+1)}} <$$



Troubles when training multi-layer fully connected NN



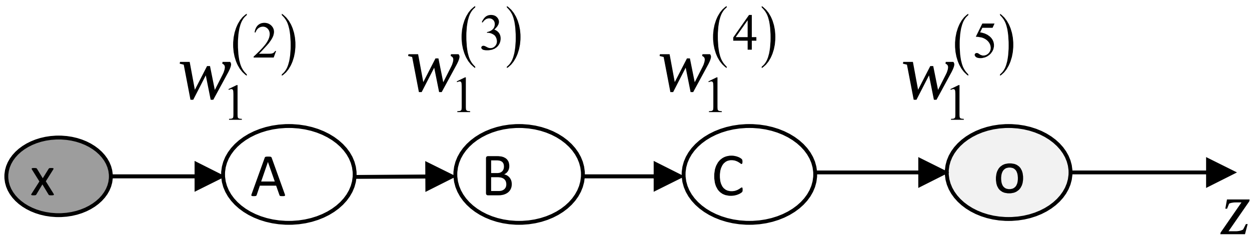
$$y^{(k)} = \varphi\left(w_1^{(k)} y^{(k-1)} - T^{(k)}\right)$$

$y^{(k)}$: neuron output at layer k

$T^{(k)}$ *:* neuron threshold

φ*:* sigmoidal activation function

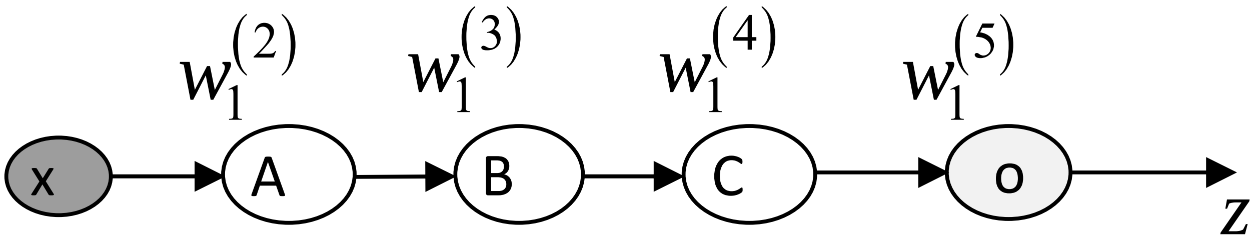
Troubles when training multi-layer fully connected NN



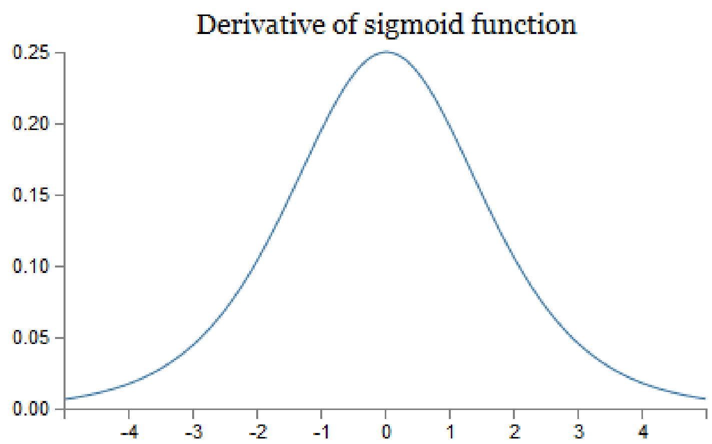
We analyze for instance the gradient $\frac{\partial E}{\partial T^{(2)}}$ associated to the first hidden neuron A

$$\frac{\partial E}{\partial T^{(2)}} = \varphi'\left(w_1^{(2)}x - T^{(2)}\right) \times w_1^{(3)}$$
$$\times \varphi'\left(w_1^{(3)}y^{(2)} - T^{(3)}\right) \times w_1^{(4)}$$
$$\times \varphi'\left(w_1^{(4)}y^{(3)} - T^{(4)}\right) \times w_1^{(5)}$$
$$\times \varphi'\left(w_1^{(5)}y^{(4)} - T^{(5)}\right) \times \frac{\partial E}{\partial z}$$

Troubles when training multi-layer fully connected NN

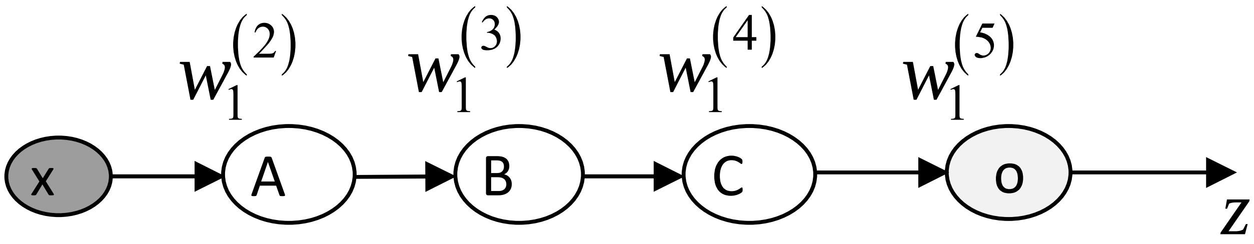


$$\frac{\partial E}{\partial T^{(2)}} = \varphi'\left(w_1^{(2)}x - T^{(2)}\right) \times w_1^{(3)} \times \varphi'\left(w_1^{(3)}y^{(2)} - T^{(3)}\right) \times w_1^{(4)} \times \varphi'\left(w_1^{(4)}y^{(3)} - T^{(4)}\right) \times w_1^{(5)} \times \varphi'\left(w_1^{(5)}y^{(4)} - T^{(5)}\right) \times \frac{\partial E}{\partial z}$$



Under the hypothesis that $|w_i| < 1$ $\quad \left|\varphi'\left(w_1^{(k)}y^{(k-1)} - T^{(k)}\right) \times w_1^{(k)}\right| < 1/4$

Troubles when training multi-layer fully connected NN



$$\frac{\partial E}{\partial T^{(2)}}$$

$$= \varphi'\left(w_1^{(2)}x - T^{(2)}\right) \times w_1^{(3)} \qquad < 1/4$$

$$\times \varphi'\left(w_1^{(3)}y^{(2)} - T^{(3)}\right) \times w_1^{(4)} \qquad < 1/4$$

$$\times \varphi'\left(w_1^{(4)}y^{(3)} - T^{(4)}\right) \times w_1^{(5)} \qquad < 1/4$$

$$\times \varphi'\left(w_1^{(5)}y^{(4)} - T^{(5)}\right) \times \frac{\partial E}{\partial z} \qquad < 1/4$$

Under the hypothesis that $\left|w_i\right| < 1$

the 2nd layer will get a weight update smaller than 1/64

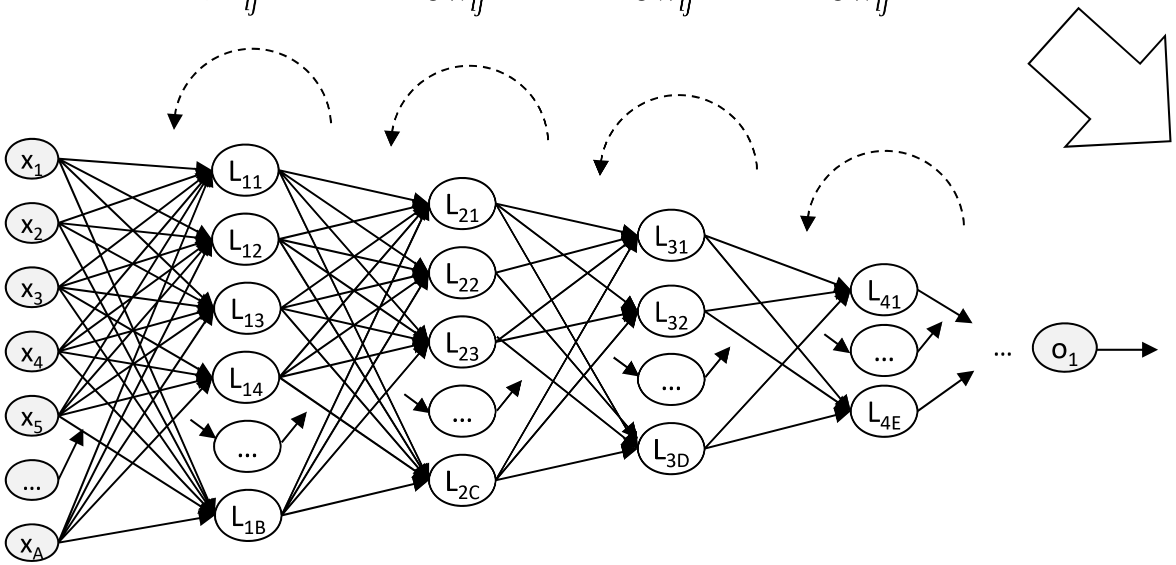
the 3rd layer will get a weight update smaller than 1/32

the 4th layer will get a weight update smaller than 1/16

Troubles when training multi-layer fully connected NN

Vanishing gradient problem in the back-propagation

$$0 \leftarrow \frac{\partial E}{\partial w_{ij}^{(l-k)}} < \quad \frac{\partial E}{\partial w_{ij}^{(l-1)}} < \quad \frac{\partial E}{\partial w_{ij}^{(l)}} < \quad \frac{\partial E}{\partial w_{ij}^{(l+1)}} <$$



1. Early hidden layers learn much more slowly than later hidden layers
2. Weights in the early hidden layers may undergo erratic updating

Possible solutions:

1. removing fully connection exploiting local receptive field and convolutional neural processing
2. uses autoencoder units;
3. automatic weight control of neurons that are saturating during training by means of dropout.

Remarks and issues on Multi-layer ANN (1)

- Structural issue
  - In theory hidden layers allow solving any classification task but…
    - Linear activation prevents general classification properties
      - The combination of hyper-planes generated by the internal neurons divides the input space in closed and partially-closed regions characterized by irregular linear bounds
    - Non-Linear activation functions to model non-linear decision boundaries
    - Feedforward NN with a single hidden layer of sigmoidal units are capable of approximating uniformly any continuous multivariate function, to any degree of accuracy (Hornik et al., 1989, " Multilayer feedforward networks are universal approximators" Neural Networks 2(5), 359-366).

Remarks and issues on Multi-layer ANN (2)

- Training
  - The shape of E for multi-layer networks is complex as it is determined by the sequence of non-linear operations during the computation of the activation functions
    - The surface error is not smooth as interlayer weights are multiplied one with each other (Local minima)
  - As for single layer network, It is not easy to establish the optimal value of the learning rate for multi-layer networks (it is dependent on the error function topology)
    - Small values of η imply good approximation but slow convergence
    - Large values of η imply fast convergence but possible either local minima or oscillations (see Perceptron lecture)

Questions to address

Training

1. How training should be efficiently carried out?
2. How many pattern in the training set?
3. How to select initial weights?
4. How to select the value of the learning rate?
5. The update of the weights must be performed after each training (on-line mode) or after the presentation of the whole training set (batch mode)?
6. When we should stop the training process?
7. How to avoid local minima and smooth zones of the error function?

Structure

1. How many hidden layers? Issue of the vanishing gradient
2. How many neurons for each layer?
3. . Which are the best activation functions?

Heuristic rules .. final values are determinate by a trial and error approach.