Statistical Machine Learning

Semester 2, 2017

Workshop #5: Neural Networks

Prepared by: Yasmeen George

THE UNIVERSITY OF MELBOURNE

Neural Network Architecture

Given n inputs and p hidden layers,

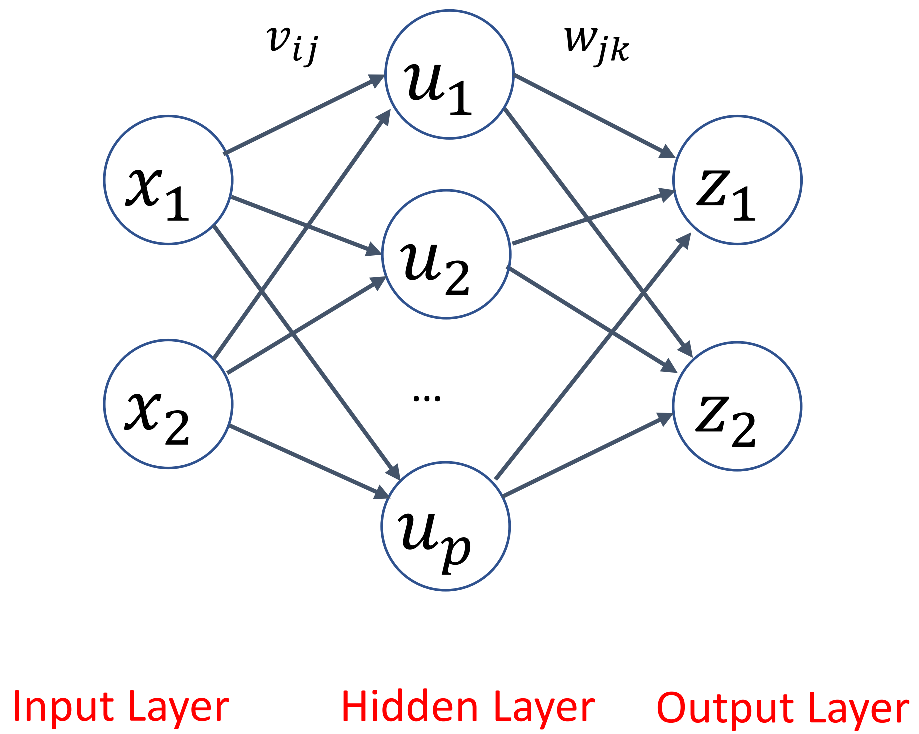
- How many weights are connected to each hidden neuron ? m+1

- How many weights should be trained for the whole hidden layer ? p*(m+1)

Given p hidden layers and k output neurons,

- How many weights are connected to each output neuron ? p+1

- How many weights should be trained for the whole output layer ? k*(p+1)

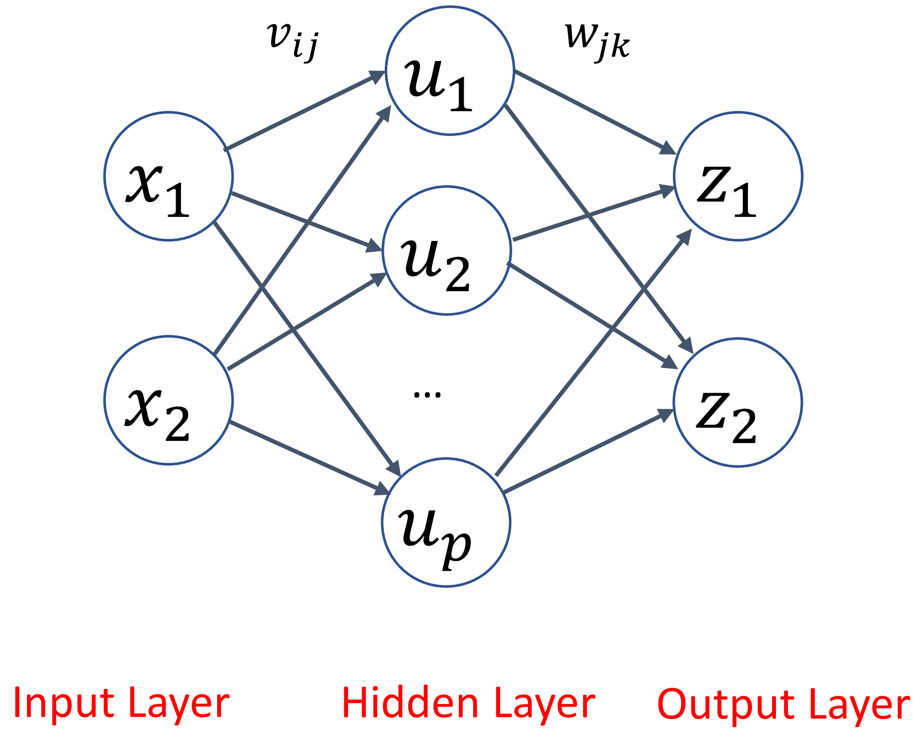


Hidden Layer forward pass calculations:

$$r_j = v_{0j} + \sum_{i=1}^{2} x_i v_{ij} = \sum_{i=0}^{2} x_i v_{ij}$$

$$u_j = \mathrm{g}(r_j)$$

$$\mathrm{g}(r) = \tanh(r) = \frac{e^r - e^{-r}}{e^r + e^{-r}}$$

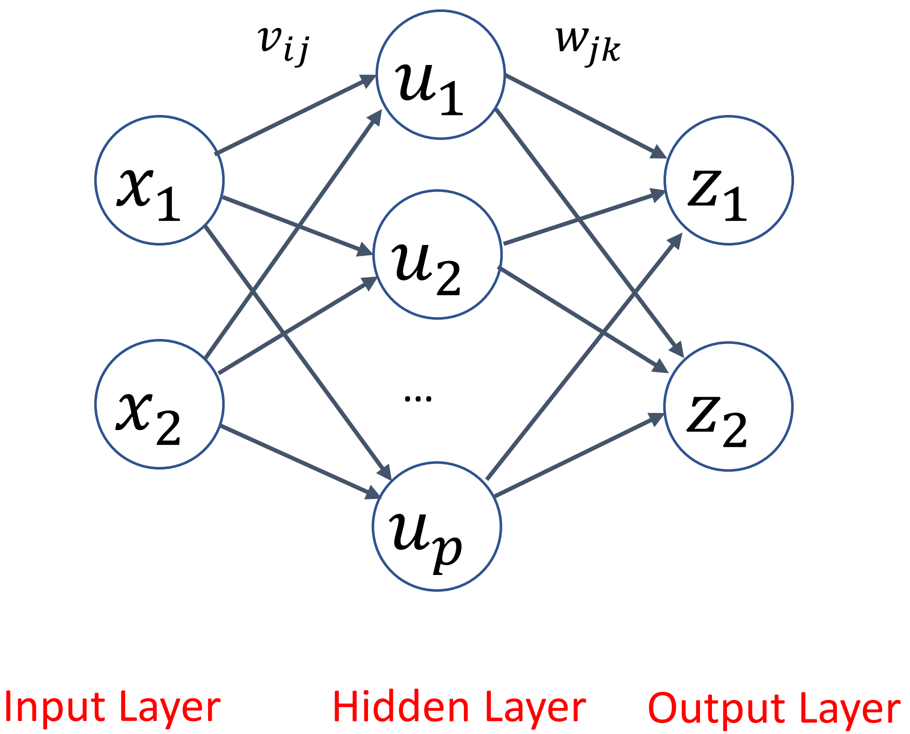


Output Layer forward pass calculations:

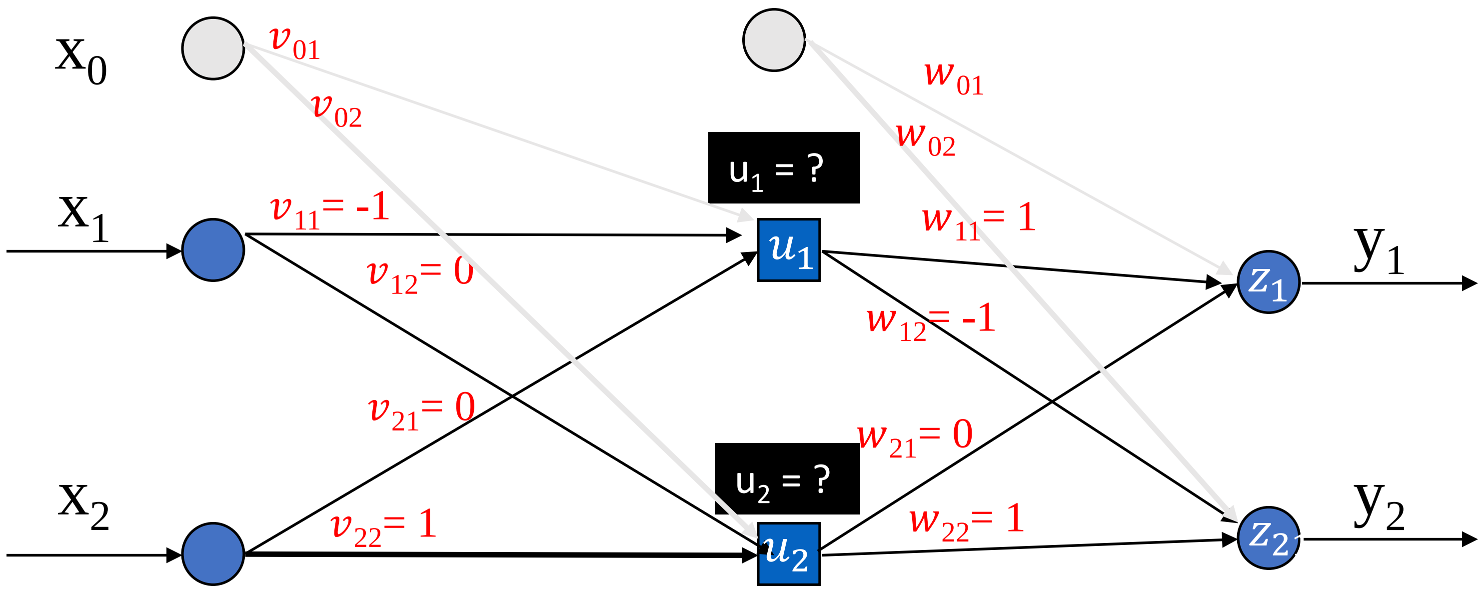
$$s_k = w_{0k} + \sum_{j=1}^{p} u_j w_{jk} = \sum_{j=0}^{p} u_j w_{jk}$$

$$z_k = f(s_k)$$

$$f(s) = \frac{1}{1 + e^{-s}}$$



An example: (Forward pass) – hidden calculations



Use “tanh” activation function (i.e. g(a) = tanh(a))

Have input [0 1] with target [1 0].

All biases set to 1

- $r_1 = 1 + -1 \times 0 + 0 \times 1 = 1 \rightarrow$
  $u_1 = \tanh(r_1) = \tanh(1) =$ **0.76**
- $r_2 = 1 + 0 \times 0 + 1 \times 1 = 2 \rightarrow$
  $u_2 = \tanh(r_2) = \tanh(2) =$ **0.97**

Weight Matrix V

[p x (m+1)]

v_{ij}	i = 0	i = 1	i =2
j=1	1	-1	0
j=2	1	0	1

Input vector x

[m+1 x 1]

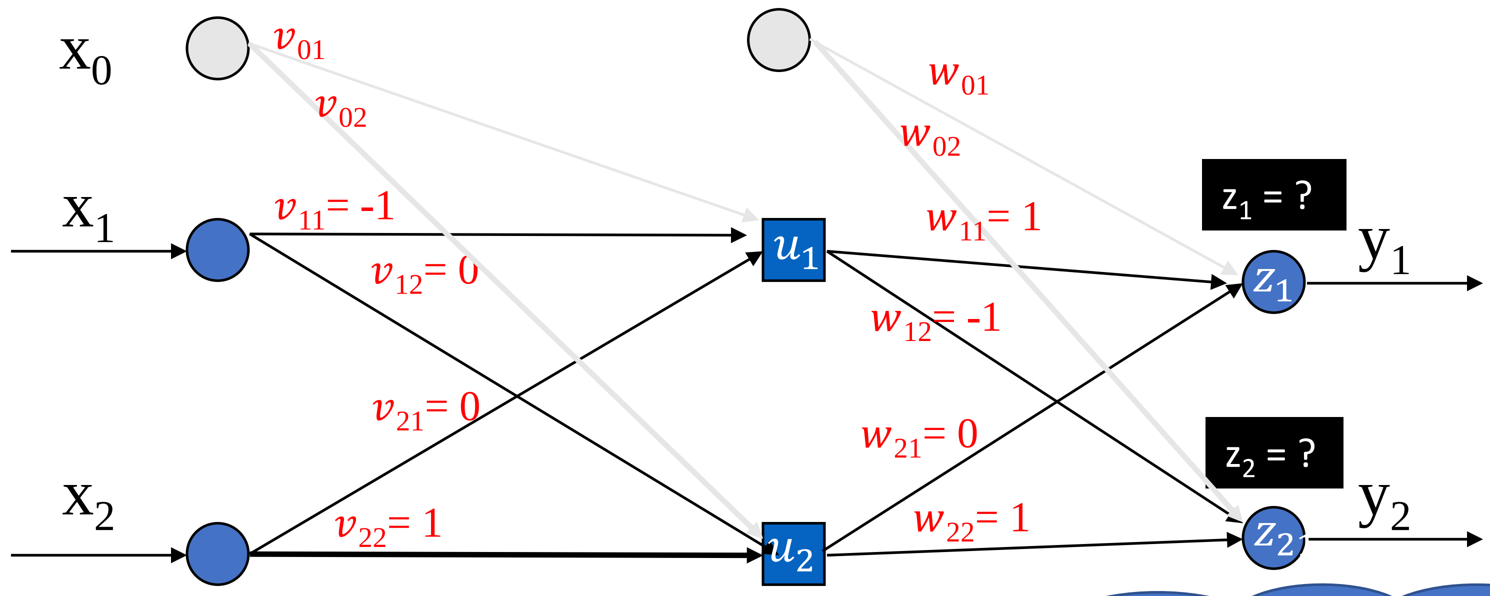
[1 0 1]′

Vector r

[p x 1]

[1 2]′

An example: (Forward pass) – output calculations



Back to tutorial to fill in compute_forward(x,V,W) & ann_predict(X,V,W) functions

Use identity activation function (i.e. g(a) = a)

Have input [0 1] with target [1 0].

All biases set to 1

- $s_1 = 1 + 1 \text{x} 0.76 + 0 \text{x} 0.97 = 1.76 \rightarrow$
  $z_1 = s_1 =$ **1.76**
- $s_2 = 1 + -1 \text{x} 0.76 + 1 \text{x} 0.97 = 1.21 \rightarrow$
  $z_2 = s_2 =$ **1.21**

Weight Matrix W
[k x (p+1)]

w_{jk}	j = 0	j = 1	j =2
k=1	1	1	0
k=2	1	-1	1

×

Input vector u
[p+1 x 1]
[1 0.76 0.97]′

=

Vector s
[k x 1]
[1.76 1.21]′

Backpropagation update rule : (1)

- Discrepancy $l = 0.5 \cdot \sum_{k=1}^{c} (y_k - z_k)^2$
- Partial derivatives $\frac{\partial l}{\partial w_{jk}} = \boxed{\frac{\partial l}{\partial s_k}} \frac{\partial s_k}{\partial w_{jk}}$ and $\frac{\partial l}{\partial v_{ij}} = \boxed{\frac{\partial l}{\partial s_k}} \frac{\partial s_k}{\partial v_{ij}}$

let's call δ_k

- $\delta_k = \frac{\partial l}{\partial s_k} = -(y_k - z_k) z_k (1 - z_k)$
- $\frac{\partial l}{\partial w_{jk}} = \delta_k u_j$
- $\frac{\partial l}{\partial v_{ij}} = g'(r_j) x_i \sum_{k=1}^{c} \delta_k w_{jk}$

Stochastic Gradient Descent update rule:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla l\left(\boldsymbol{\theta}^{(t)}\right)$$

$$w_{jk}{}^{(t+1)} = w_{jk}{}^{(t)} - \eta \frac{\partial l}{\partial w_{jk}}$$

$$v_{ij}{}^{(t+1)} = v_{ij}{}^{(t)} - \eta \frac{\partial l}{\partial v_{ij}}$$

Backpropagation update rule: (2)

- Discrepancy $l = -\sum_{k=1}^{c} y_k \log(z_k) - (1 - y_k)\log(1 - z_k)$
- Partial derivatives $\frac{\partial l}{\partial w_{jk}} = \boxed{\frac{\partial l}{\partial s_k}} \frac{\partial s_k}{\partial w_{jk}}$ and $\frac{\partial l}{\partial v_{ij}} = \boxed{\frac{\partial l}{\partial s_k}} \frac{\partial s_k}{\partial v_{ij}}$

let's call δ_k

- $\delta_k = \frac{\partial l}{\partial s_k} = (z_k - y_k)$
- $\frac{\partial l}{\partial w_{jk}} = \delta_k u_j$
- $\frac{\partial l}{\partial v_{ij}} = g'(r_j) x_i \sum_{k=1}^{c} \delta_k w_{jk}$
- $= (1 - u_j^2) x_i \sum_{k=1}^{c} \delta_k w_{jk}$
- $= (1 - u_j^2) x_i \delta_1 w_{j1} + (1 - u_j^2) x_i \delta_1 w_{j1}$

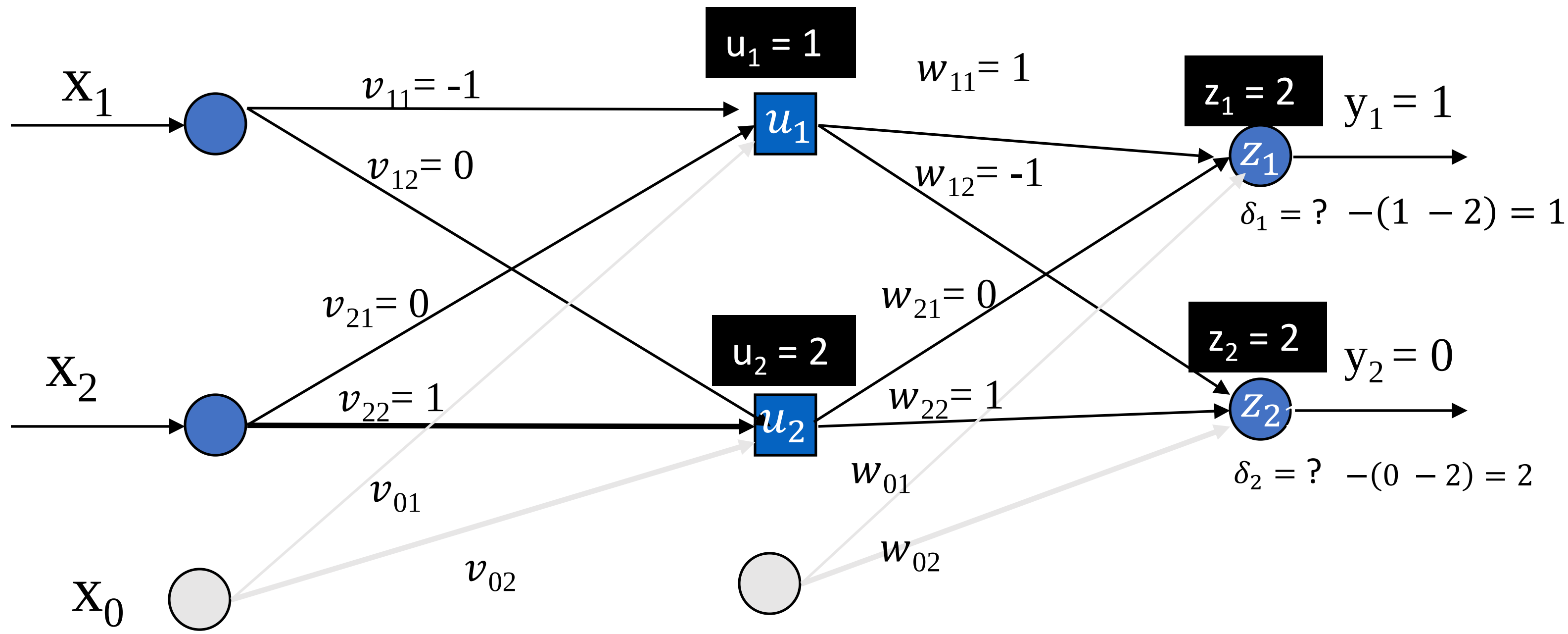
Stochastic Gradient Descent update rule:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla l\left(\boldsymbol{\theta}^{(t)}\right)$$

$$w_{jk}^{(t+1)} = w_{jk}^{(t)} - \eta \frac{\partial l}{\partial w_{jk}}$$

$$v_{ij}^{(t+1)} = v_{ij}^{(t)} - \eta \frac{\partial l}{\partial v_{ij}}$$

An example: (Backward pass) – output layer



Have input [0 1] with target [1 0]. Learning rate η = 0.1

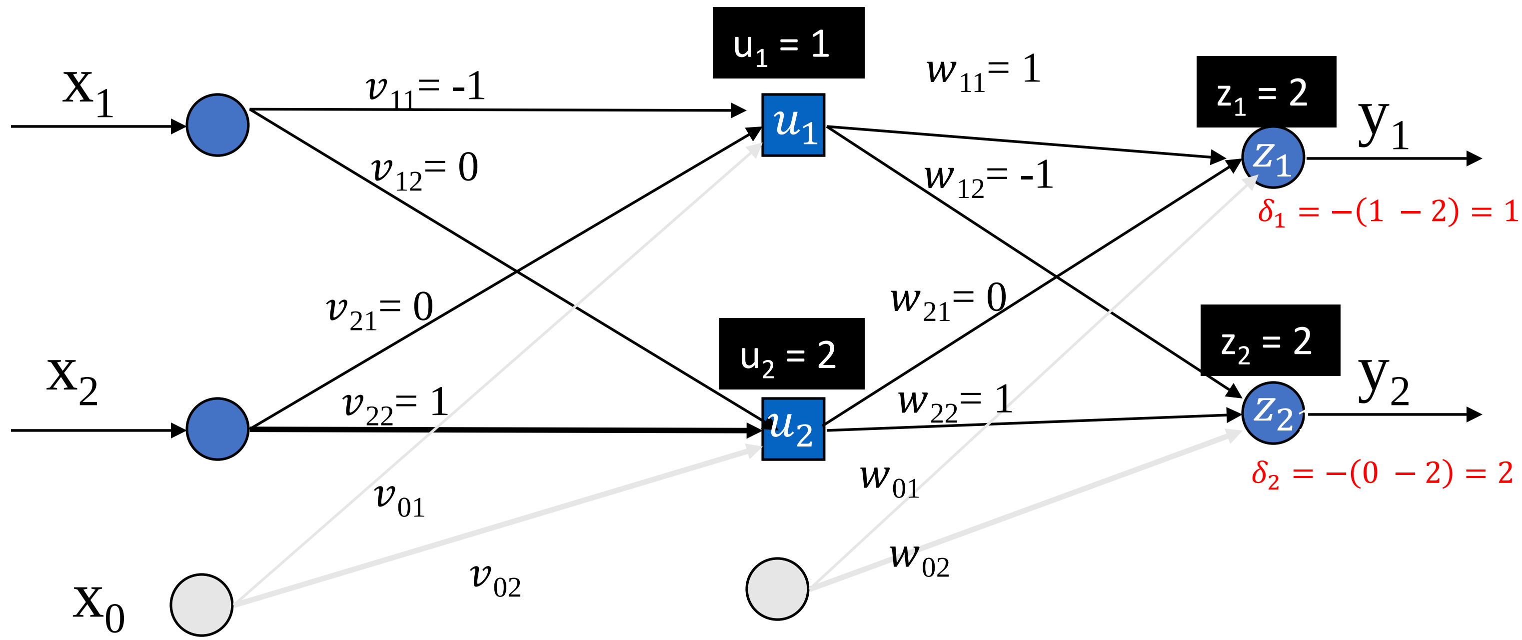
$$\delta_k = -(y_k - z_k)\ (\frac{\partial f_k}{\partial s_k} = 1\ )$$

$$w_{jk}{}^{(t+1)} = w_{jk}{}^{(t)} - \eta \frac{\partial l}{\partial w_{jk}}$$
$$= w_{jk}{}^{(t)} - \eta\, \delta_k u_j$$

k=1, j =1 → w_{11}= 1 - 0.1 * 1 * 1 = 0.9
k=1, j =2 → w_{21}= 0 - 0.1 * 1 * 2 = -0.2
k=1, j =0 → w_{01}= 1 - 0.1 * 1 * 1 = 0.9

k=2, j =1 → w_{12}= -1 - 0.1 * 2 * 1 = -1.2
k=2, j =2 → w_{22}= 1 - 0.1 * 2 * 2 = 0.6
k=2, j =0 → w_{02}= 1 - 0.1 * 2 * 1 = 0.8

An example: (Backward pass) – hidden layer



Have input [0 1] with target [1 0]. Learning rate η = 0.1

$$\delta_k = -(y_k - z_k) \; \left(\frac{\partial f_k}{\partial s_k} = 1\right)$$

$$v_{ij}^{(t+1)} = v_{ij}^{(t)} - \eta \frac{\partial l}{\partial v_{ij}}$$

$$= v_{ij}^{(t)} - \eta x_i \sum_{k=1}^{c} \delta_k \, w_{jk}$$

j=1, i =1 → v_{11}= -1 - 0.1 * -1 * 0 = -1
j=1, i =2 → v_{21}= 0 - 0.1 * -1 * 1 = 0.1
j=1, i =0 → v_{01}= 1 - 0.1 * -1 * 1 = 1

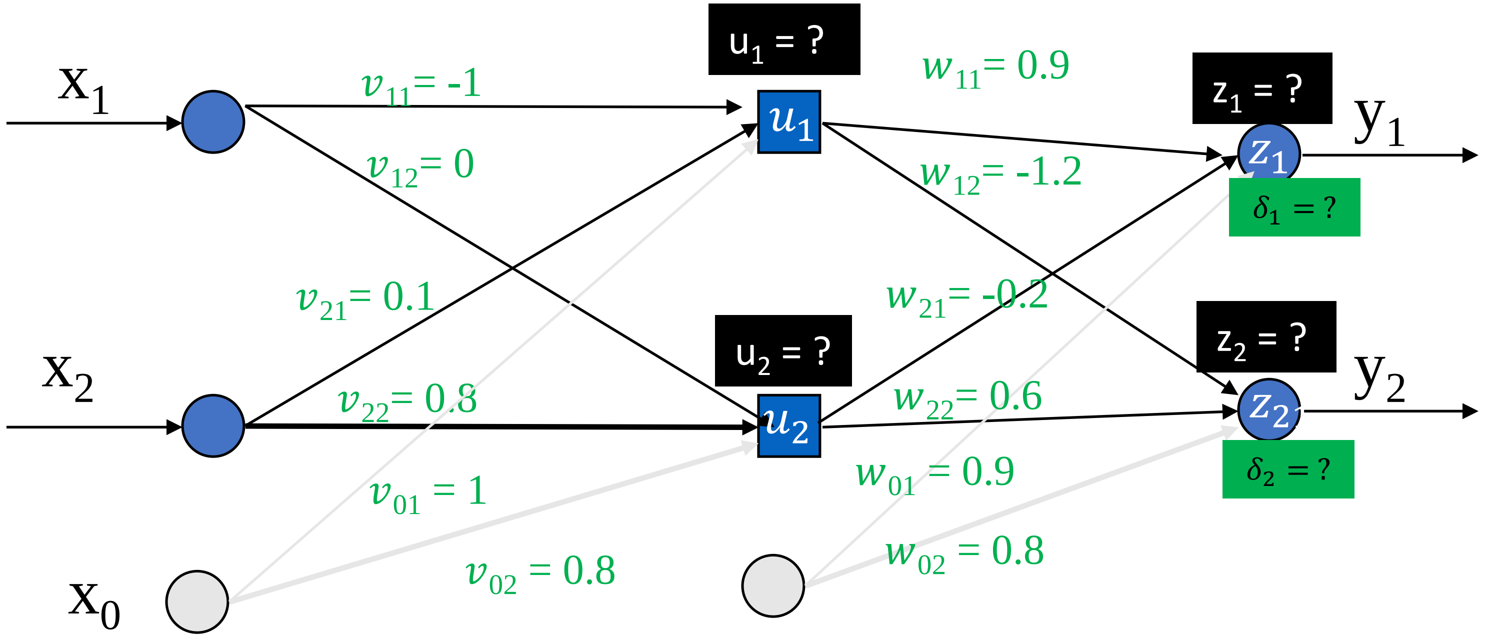
Note: use old weights w_{jk}

j=2, i =1 → v_{12}= 0 - 0.1 * 2 * 0 = 0
j=2, i =2 → v_{22}= 1 - 0.1 * 2 * 1 = 0.8
j=2, i =0 → v_{02}= 1 - 0.1 * 2 * 1 = 0.8

$\sum_{k=1}^{c} \delta_k \, w_{1k}$= 1 x 1 + -1 x 2 = -1

$\sum_{k=1}^{c} \delta_k \, w_{2k}$= 0 x 1 + 1 x 2 = 2

An example: updated weights after ONE iteration



Back to tutorial to fill in
update_params(x,y,V,W,eta) &
ann_train(X,y,V0,W0)functions