

Neural networks

Nando de Freitas



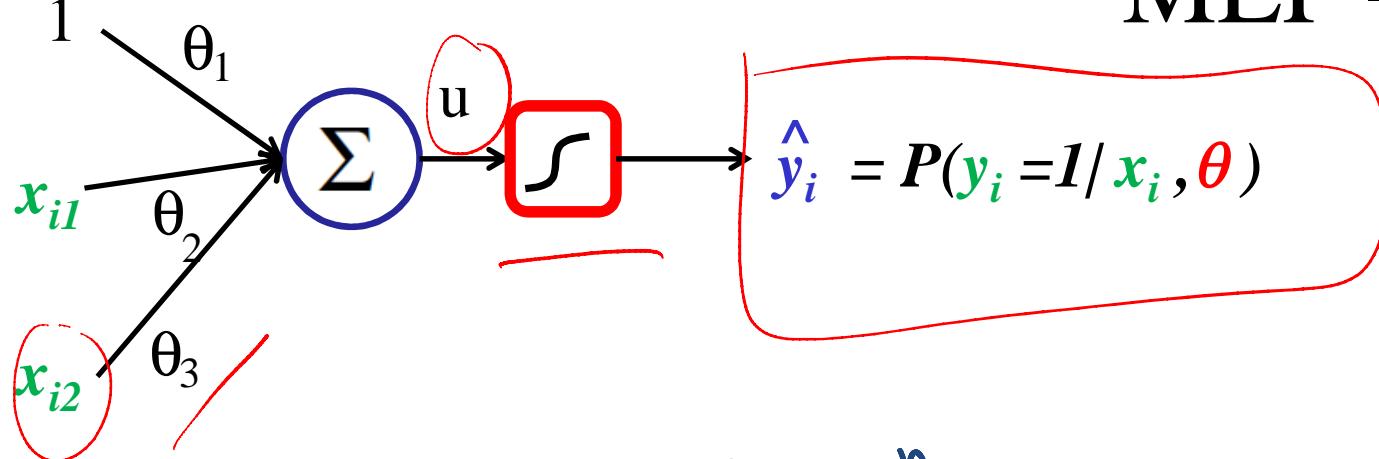
UNIVERSITY OF
OXFORD

Outline of the lecture

This lecture introduces you to the fascinating subject of classification and regression with artificial neural networks. In particular, it

- Introduces multi-layer perceptrons (MLPs)
- Teaches you how to combine probability with neural networks so that the nets can be applied to regression, binary classification and multivariate classification.
- Discusses the modular approach to backpropagation and neural network construction in Torch, which was introduced in the previous lecture.

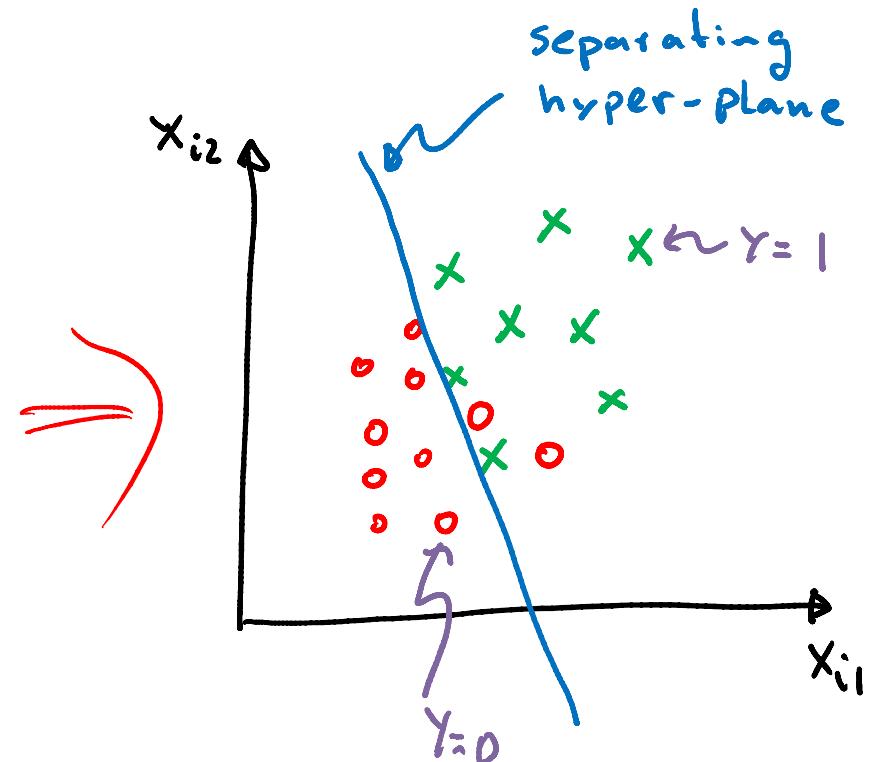
MLP – 1 neuron



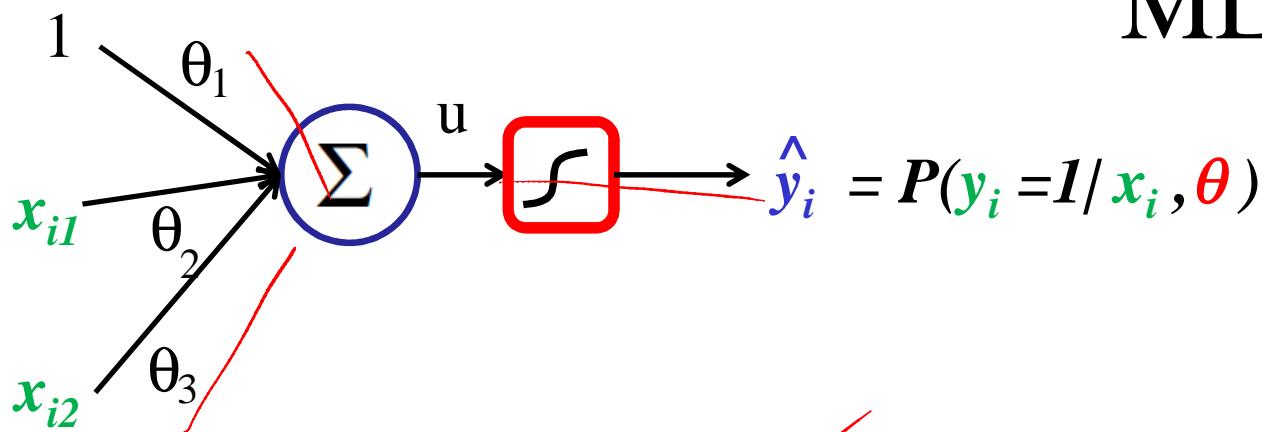
We are given the data $\{x_i, y_i\}_{i=1}^n$

e.g.

	x_{i1}	x_{i2}	y_i
$i=1$	0,2	6	0
$i=2$	0,3	22	-1
$i=3$	0,6	-0,6	-1
$i=4$	-0,4	58	0
:			



MLP – 1 neuron



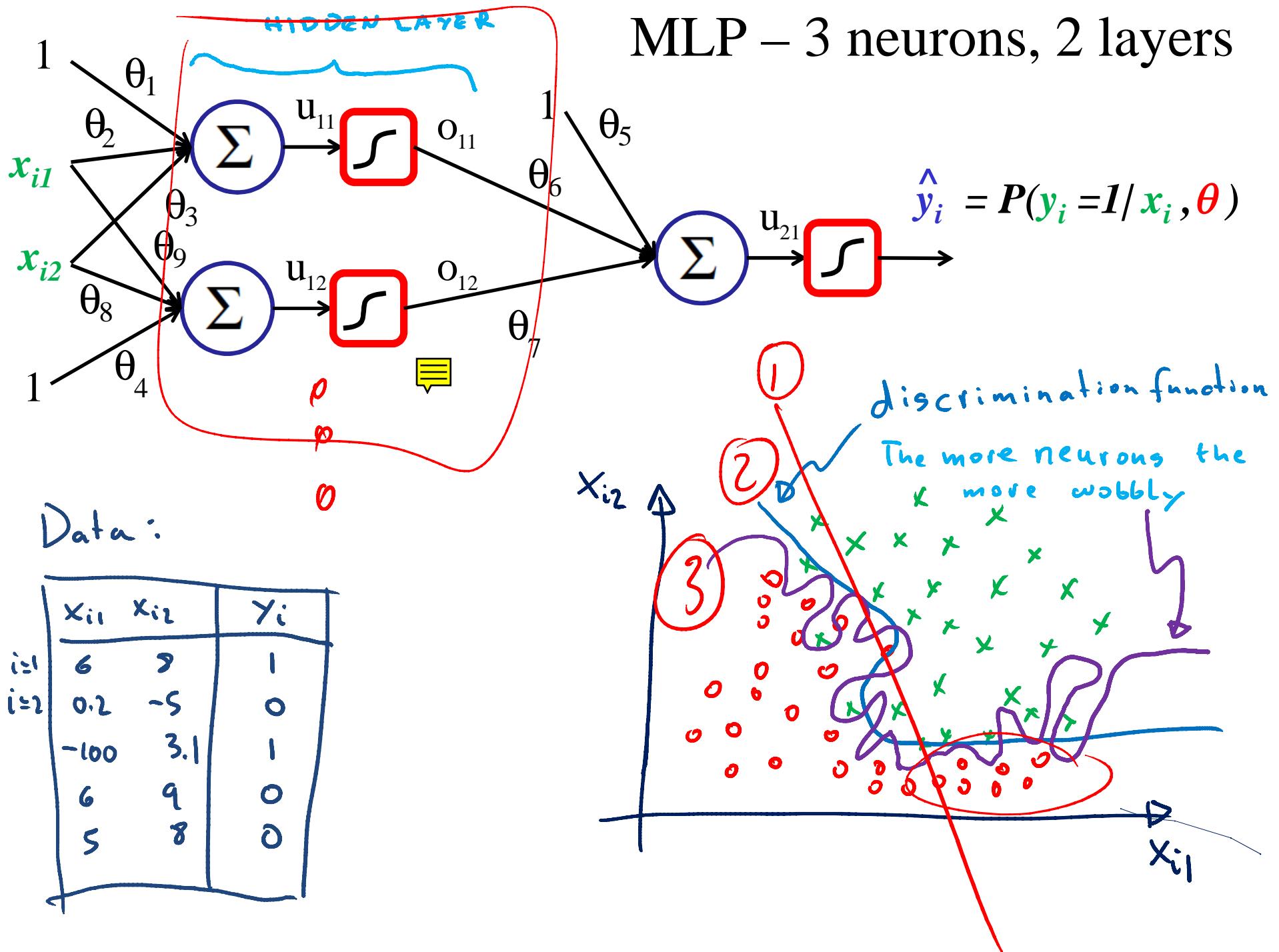
$$u = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2}$$

$$\hat{y}_i = \frac{1}{1+e^{-u}} = \frac{1}{1+e^{-\theta_0 - \theta_1 x_{i1} - \theta_2 x_{i2}}} = P(y_i = 1 | x_i, \theta)$$

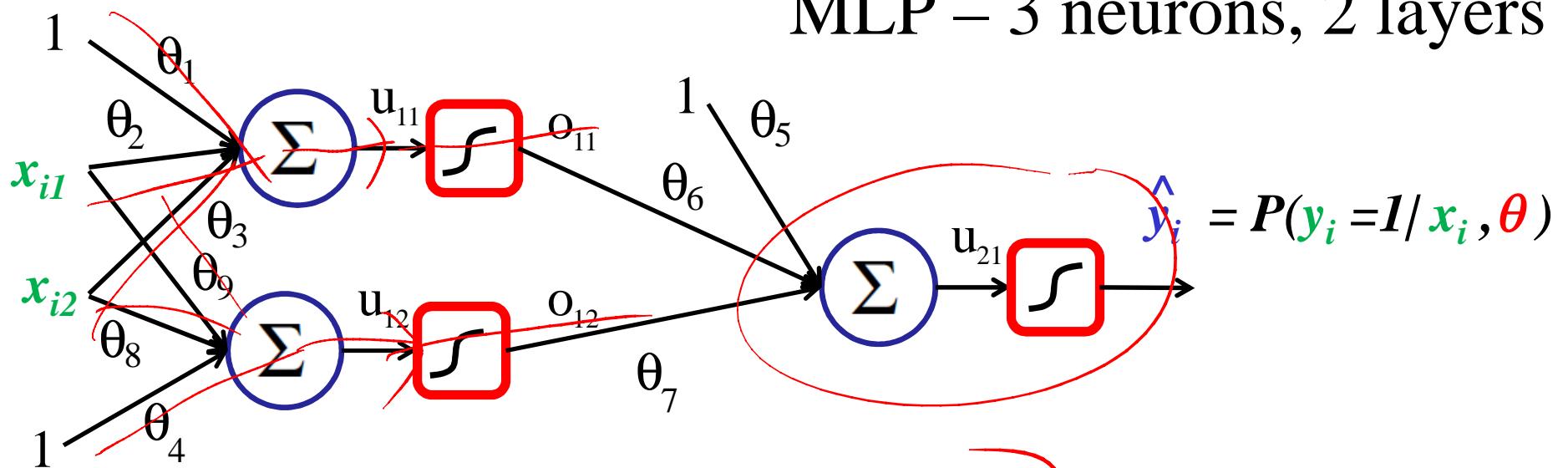
$$P(y_i = 1 | x_i, \theta) = \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1-y_i} = \begin{cases} \hat{y}_i & \text{When } y_i = 1 \\ 1 - \hat{y}_i & \text{Otherwise} \end{cases}$$

For n independent observations (Bernoulli)

$$\underline{P(Y | X, \theta)} = \prod_{i=1}^n P(y_i | x_i, \theta)$$



MLP – 3 neurons, 2 layers



$$u_{11} = \theta_1 + \theta_2 x_{i1} + \theta_3 x_{i2}$$

$$u_{12} = \theta_4 + \theta_5 x_{i1} + \theta_6 x_{i2}$$

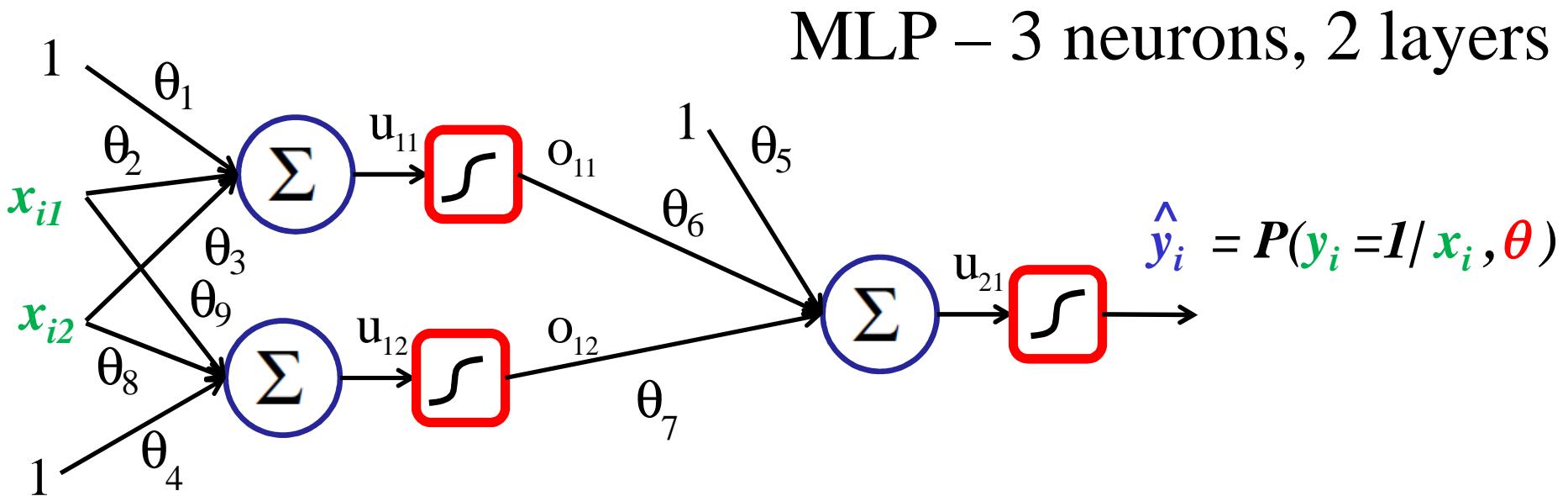
$$o_{11} = \frac{1}{1+e^{-u_{11}}}$$

$$o_{12} = \frac{1}{1+e^{-u_{12}}}$$

$$\hat{y}_i = \frac{1}{1+e^{-u_{21}}}$$

$$u_{21} = \theta_5 + \theta_6 o_{11} + \theta_7 o_{12}$$

$$P(y_i | \mathbf{x}_i, \theta) = \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1-y_i}$$



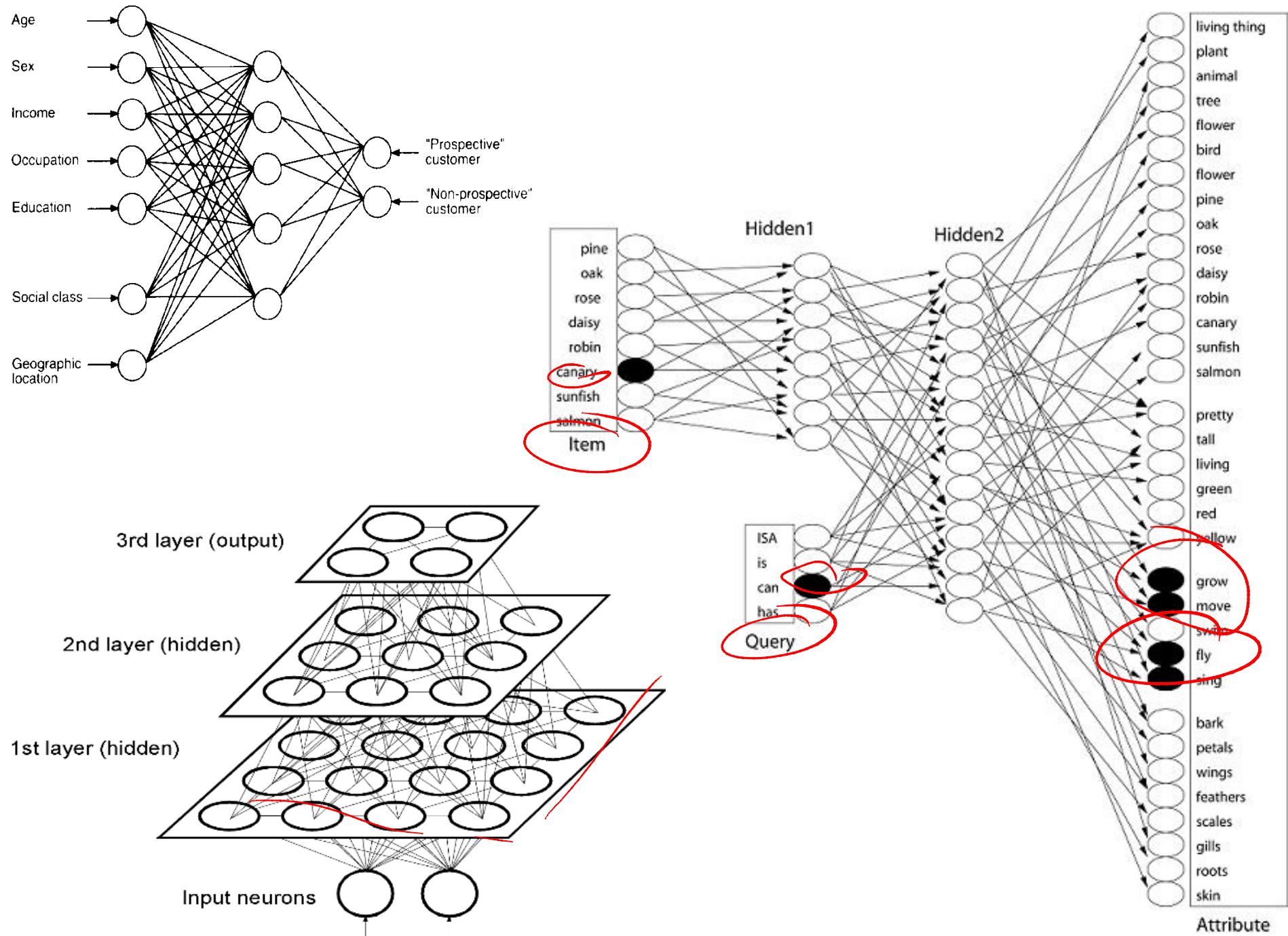
For n independent observations.

$$P(Y|X, \Theta) = \prod_{i=1}^n \hat{y}_i^{y_i} (1-\hat{y}_i)^{1-y_i} = \prod_{i=1}^n P(Y_i|X_i, \Theta)$$

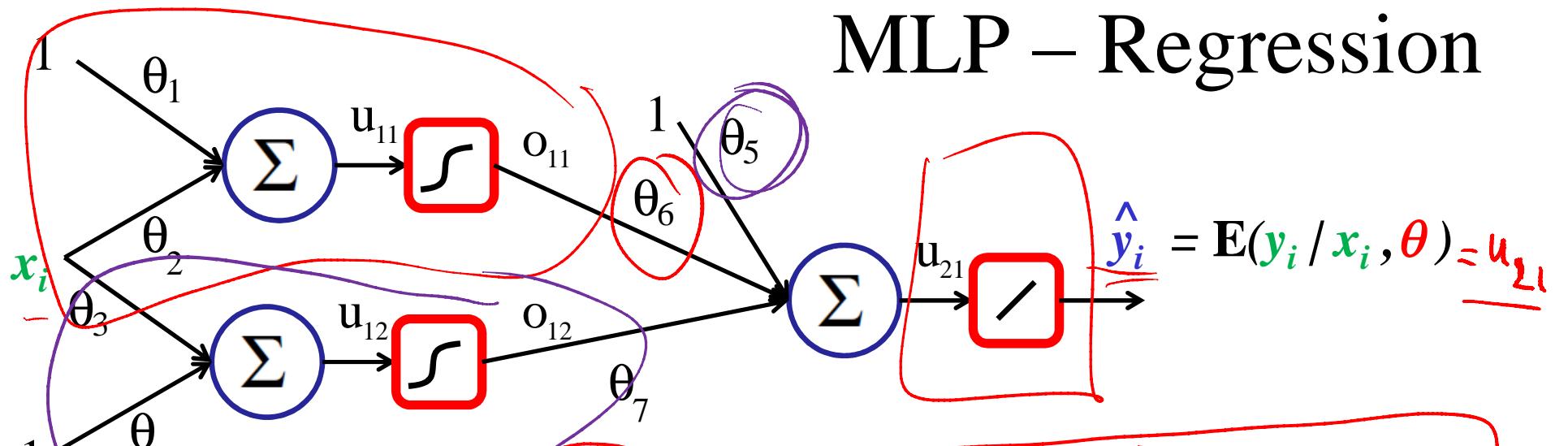
Cost:

$$C(\Theta) = -\log P(Y|X, \Theta) = -\sum_{i=1}^n y_i \log \hat{y}_i + (1-y_i) \log (1-\hat{y}_i)$$

i.e. minimize the cross-entropy error.

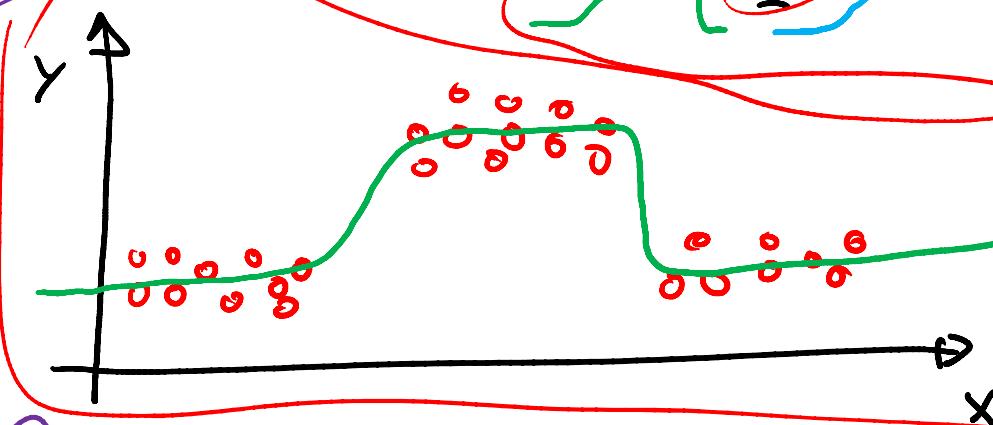


MLP – Regression



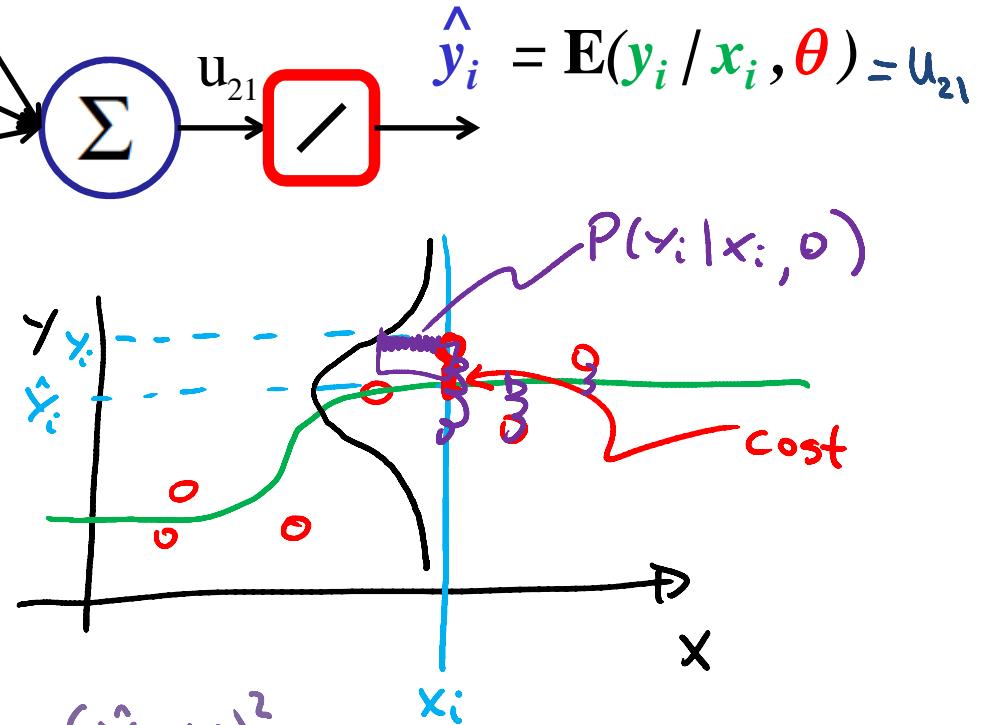
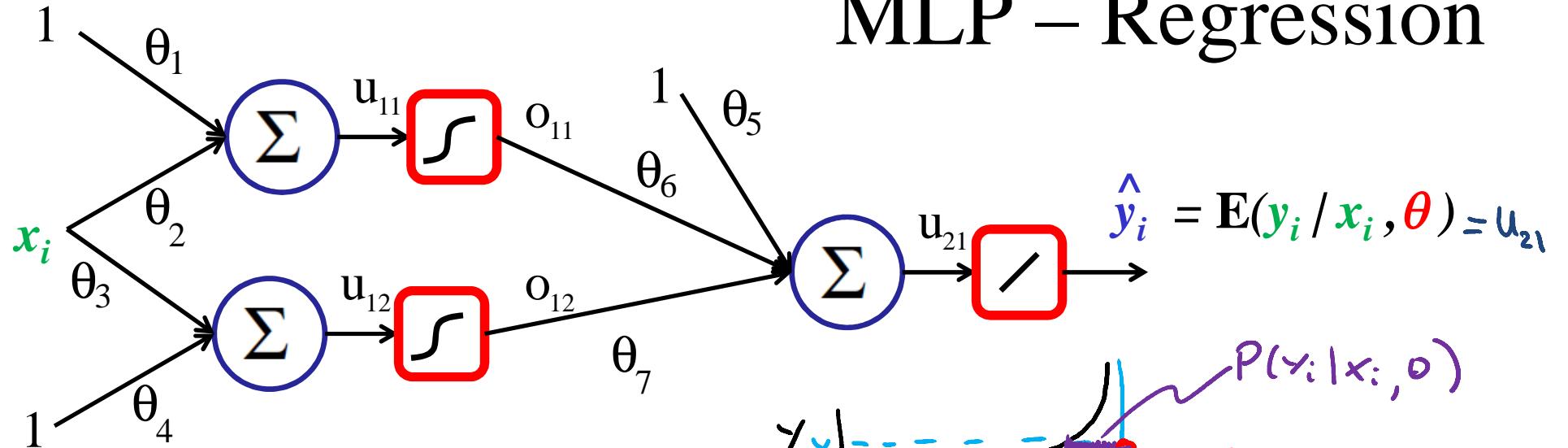
Data:

x_i	y_i
0.2	0.6
0.9	0.4
-0.6	5
-0.3	6.2



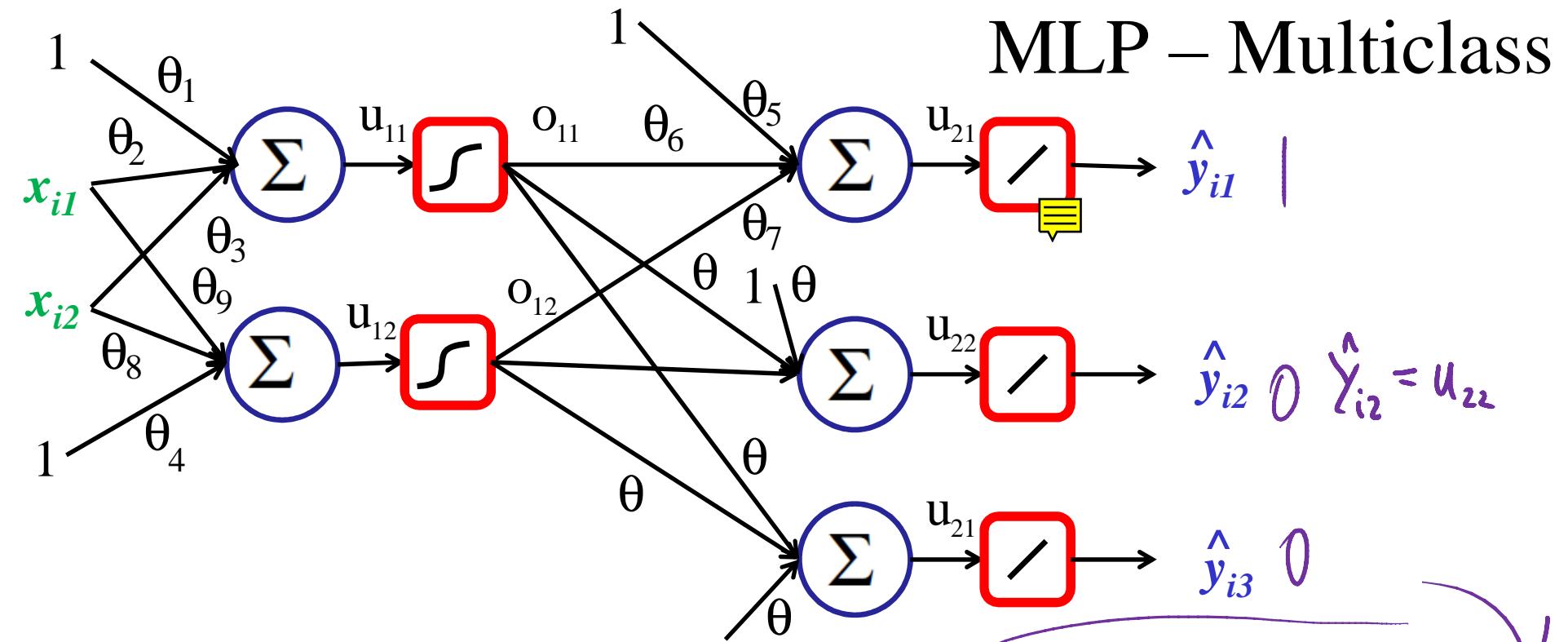
$$\hat{y}_i = \Theta_5 + \frac{\Theta_6}{1 + e^{-\Theta_1 - \Theta_2 x_i}} + \frac{\Theta_7}{1 + e^{-\Theta_4 - \Theta_3 x_i}}$$

MLP – Regression



$$P(y_i | x_i, \theta) = (2\pi\sigma^2)^{-n/2} e^{-\frac{1}{2\sigma^2} (\hat{y}_i - y_i)^2}$$

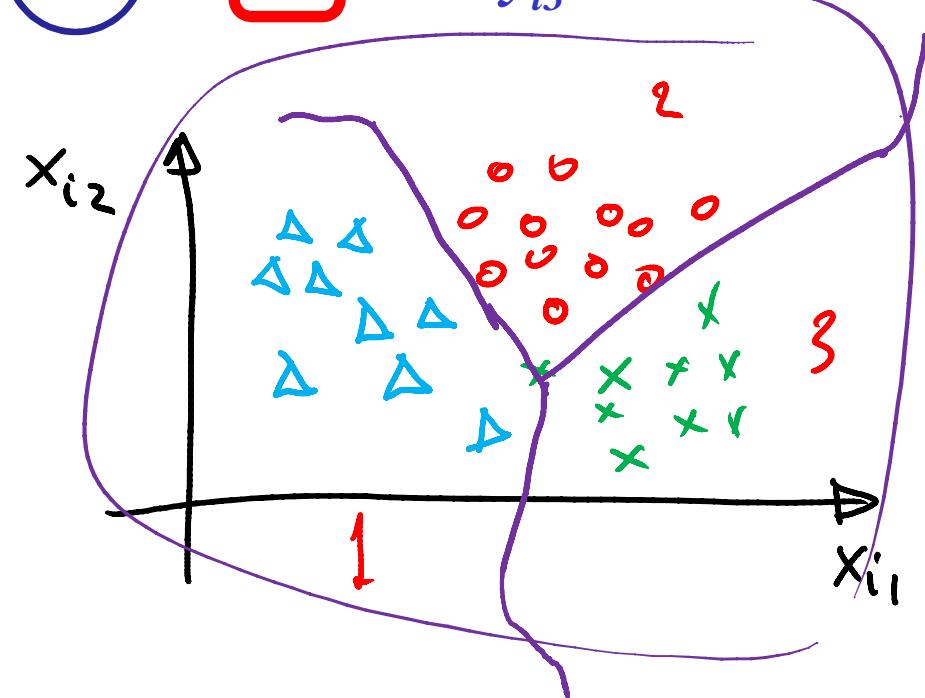
$$C(\theta) = \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad \Leftrightarrow \quad \hat{y}_i = f(\theta, x_i)$$

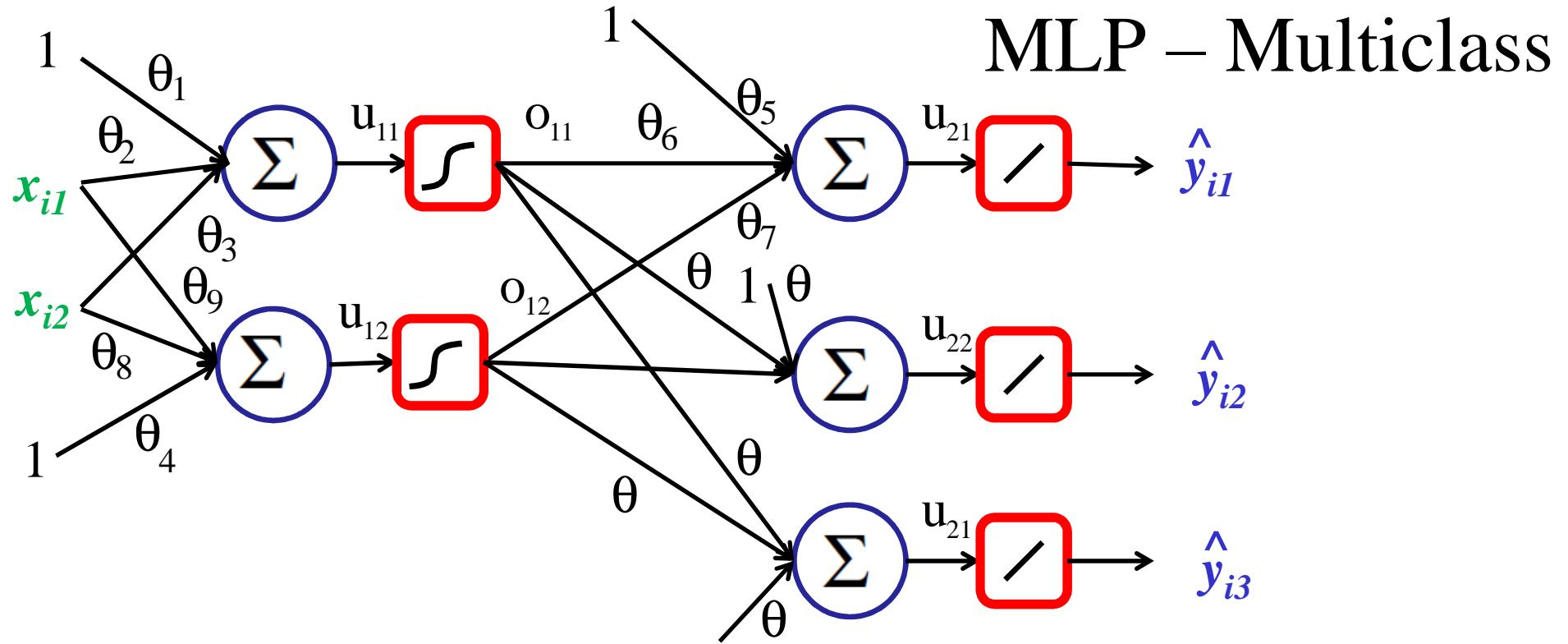


Data :

x_{i1}	x_{i2}	y_{i1}	y_{i2}	y_{i3}
0.2	0.3	0	1	0
-5	-6	1	0	0
-20	4	1	0	0
42	6.8	0	0	1

█ Class 2
△ Class 1
x Class 3





To get a probabilistic model, define: SOFTMAX

$$P(\underline{y_i = (0 \ 1 \ 0)} | x_i, \theta) = P(\underline{y_i = 2} | x_i, \theta) =$$

$$\frac{e^{\hat{y}_2}}{e^{\hat{y}_1} + e^{\hat{y}_2} + e^{\hat{y}_3}}$$

$$\mathbb{I}_2(y_i) = \begin{cases} 1 & y_i = 2 \\ 0 & \text{o.w.} \end{cases}$$

MLP – Multiclass

Then,

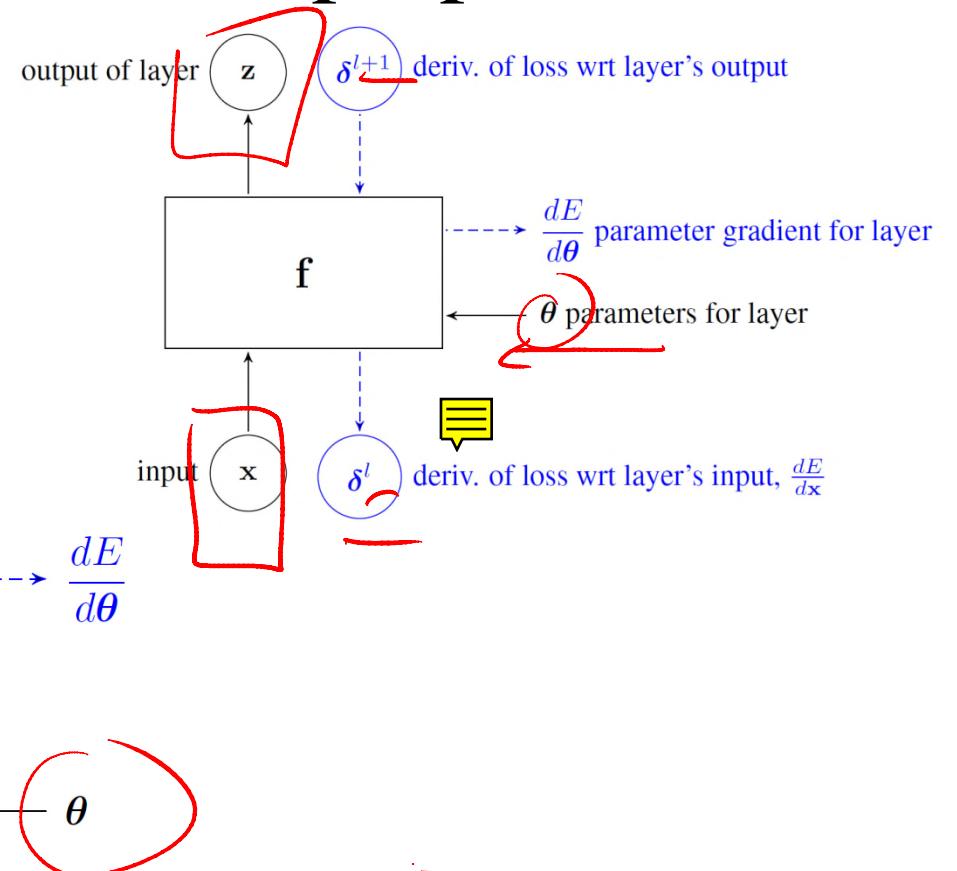
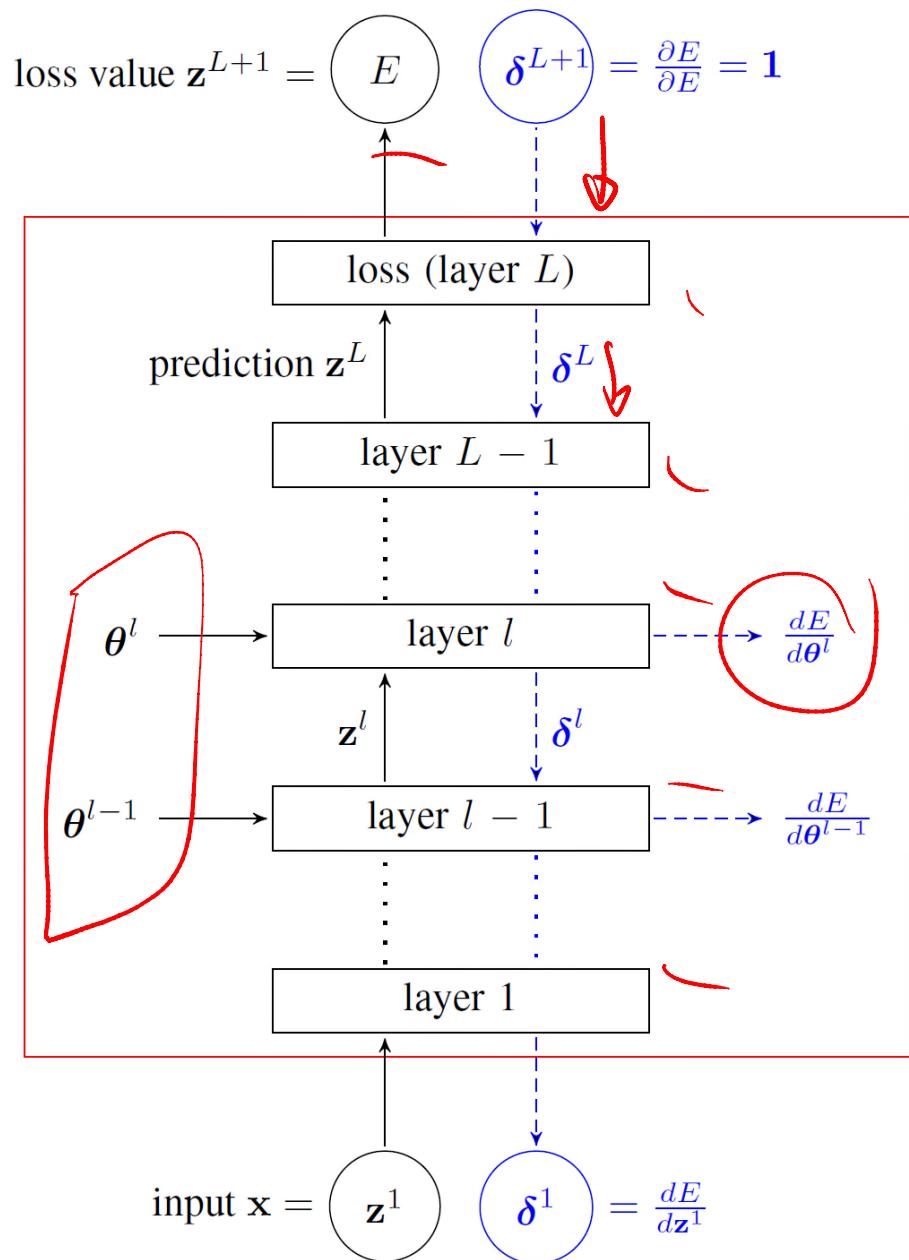
$$P(y_i | x_i, \theta) = \left[\frac{e^{\hat{y}_{i1}}}{e^{\hat{y}_{i1}} + e^{\hat{y}_{i2}} + e^{\hat{y}_{i3}}} \right] \mathbb{I}_1(y_i) + \left[\frac{e^{\hat{y}_{i2}}}{e^{\hat{y}_{i1}} + e^{\hat{y}_{i2}} + e^{\hat{y}_{i3}}} \right] \mathbb{I}_2(y_i) + \left[\frac{e^{\hat{y}_{i3}}}{e^{\hat{y}_{i1}} + e^{\hat{y}_{i2}} + e^{\hat{y}_{i3}}} \right] \mathbb{I}_3(y_i)$$

$$= \begin{cases} e^{\hat{y}_{i1}} / \text{sum} & y_i = 1 \\ e^{\hat{y}_{i2}} / \text{sum} & y_i = 2 \\ e^{\hat{y}_{i3}} / \text{sum} & y_i = 3 \end{cases}$$

Cost:

$$C(\theta) = -\log P(y_i | x_i, \theta) = -\sum_{i=1}^n \sum_{j=1}^3 \mathbb{I}_j(y_i) \log \frac{e^{\hat{y}_{ij}}}{\text{sum}}$$

Deep learning & backprop



```

1 | model = nn.Sequential()
2 | model.add( nn.Linear(2,3) )
3 | model.add( nn.LogSoftMax() )

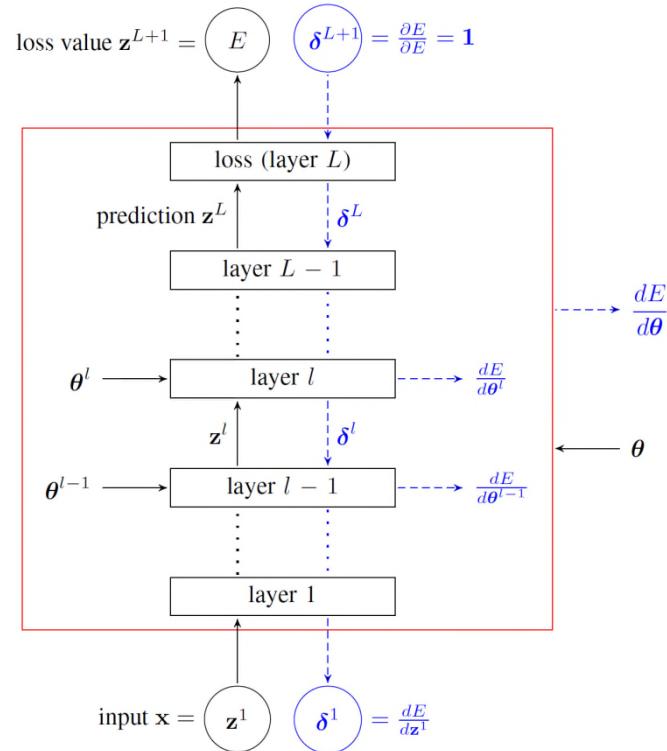
```

```

1 | -- params/gradients
2 | x, dl_dx = model.getParameters()

```

Deep learning & backprop



$\mathbf{z}^{l+1} = \mathbf{f}^l(\mathbf{z}^l; \boldsymbol{\theta}^l)$ forward [matrix]

$\delta^l := \frac{\partial E}{\partial \mathbf{z}^l} = \frac{\partial E}{\partial \mathbf{z}^{l+1}} \frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l} = \delta^{l+1} \frac{\partial \mathbf{f}^l(\mathbf{z}^l; \boldsymbol{\theta}^l)}{\partial \mathbf{z}^l}$ Jacobian [matrix]

$\delta_i^l = \sum_j \frac{\partial E}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_i^l} = \sum_j \delta_j^{l+1} \frac{\partial f_j^l(\mathbf{z}^l; \boldsymbol{\theta}^l)}{\partial z_i^l}$

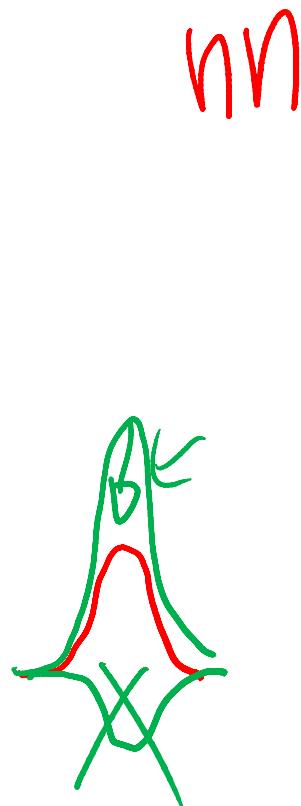
$\frac{\partial E}{\partial \boldsymbol{\theta}^l} = \frac{\partial E}{\partial \mathbf{z}^{l+1}} \frac{\partial \mathbf{z}^{l+1}}{\partial \boldsymbol{\theta}^l} = \delta^{l+1} \frac{\partial \mathbf{f}^l(\mathbf{z}^l; \boldsymbol{\theta}^l)}{\partial \boldsymbol{\theta}^l}$

$\frac{\partial E}{\partial \theta_i^l} = \sum_j \frac{\partial E}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial \theta_i^l} = \sum_j \delta_j^{l+1} \frac{\partial f_j^l(\mathbf{z}^l; \boldsymbol{\theta}^l)}{\partial \theta_i^l}$ [matrix]

Reverse auto-diff: why it works

$$\frac{\partial f^L(\mathbf{f}^{L-1}(\dots \mathbf{f}^3(\mathbf{f}^1(\mathbf{x}))))}{\partial \mathbf{x}} = \frac{\cancel{\partial f^L} \cdot \cancel{\partial \mathbf{f}^{L-1}} \cdots \cancel{\partial \mathbf{f}^2} \cdot \cancel{\partial \mathbf{f}^1}}{\cancel{\partial \mathbf{f}^{L-1}} \cdot \cancel{\partial \mathbf{f}^{L-2}} \cdots \cancel{\partial \mathbf{f}^1} \cdot \cancel{\partial \mathbf{x}}}$$
$$\frac{\partial f^L(\mathbf{f}^{L-1}(\dots \mathbf{f}^2(\mathbf{f}^1(\mathbf{x}))))}{\partial \mathbf{x}} = \left(\left(\left(\left(\frac{\partial f^L}{\partial \mathbf{f}^{L-1}} \cdot \frac{\partial \mathbf{f}^{L-1}}{\partial \mathbf{f}^{L-2}} \right) \cdots \left(\frac{\partial \mathbf{f}^2}{\partial \mathbf{f}^1} \right) \cdot \frac{\partial \mathbf{f}^1}{\partial \mathbf{x}} \right) \right) \right)$$

Deep learning: linear layer

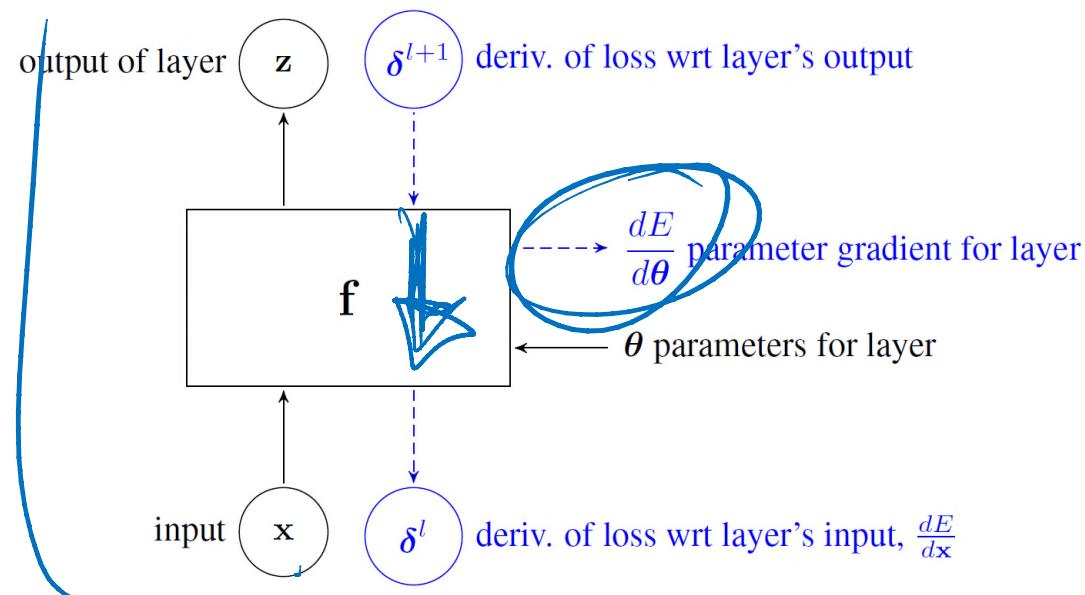


nn

$$z_j = f_j(\mathbf{x}; \theta_j) = \sum_i x_i \theta_{ij}$$

$$\delta_i^l = \sum_j \delta_{j+1}^{l+1} \frac{\partial f_j(\mathbf{x}; \theta_j)}{\partial x_i} = \sum_j \delta_{j+1}^{l+1} \theta_{ij}$$

$$\frac{\partial E}{\partial \theta_{ij}} = \sum_j \delta_{j+1}^{l+1} \frac{\partial f_j(\mathbf{x}; \theta_j)}{\partial \theta_{ij}} = \delta_{j+1}^{l+1} x_i$$

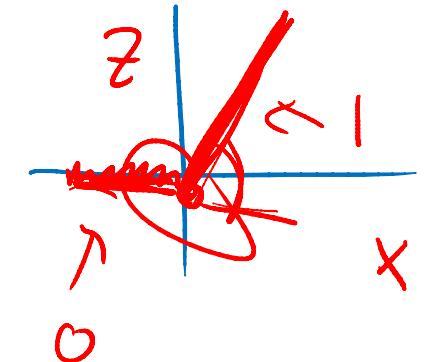


Deep learning: ReLU layer

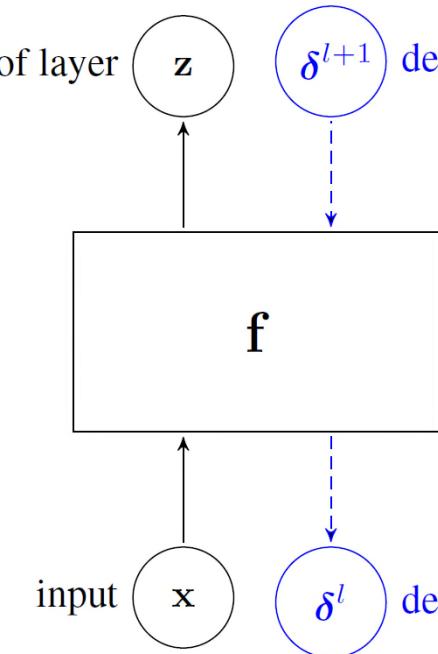
Fukushima

$$z_j = f_j(x_j) = \max(0, x_j)$$

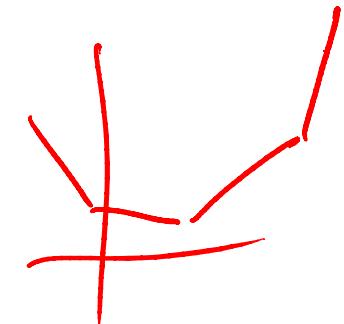
$$\delta_i^l = \sum_j \delta_{j+1}^{l+1} \frac{\partial f_j(x_j)}{\partial x_i} = \delta_{i+1}^{l+1} \mathbb{I}_{[x_i > 0]}$$



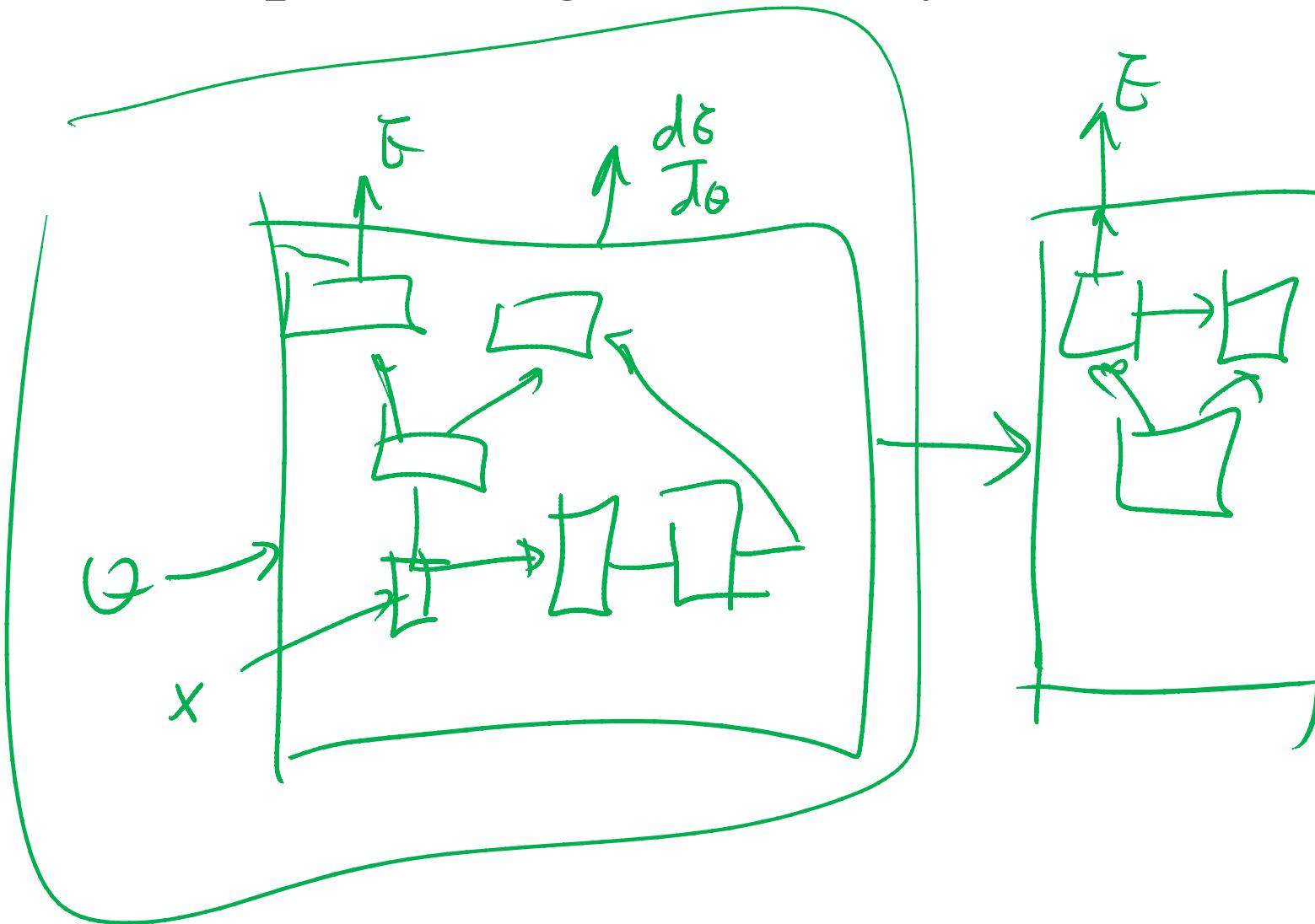
output of layer z δ^{l+1} deriv. of loss wrt layer's output



maxout



Deep learning: extremely flexible!



Next lecture

In the next lecture, we will look at a successful type of neural network that is very popular in speech and object recognition, known as a convolutional neural network.