Neural Networks

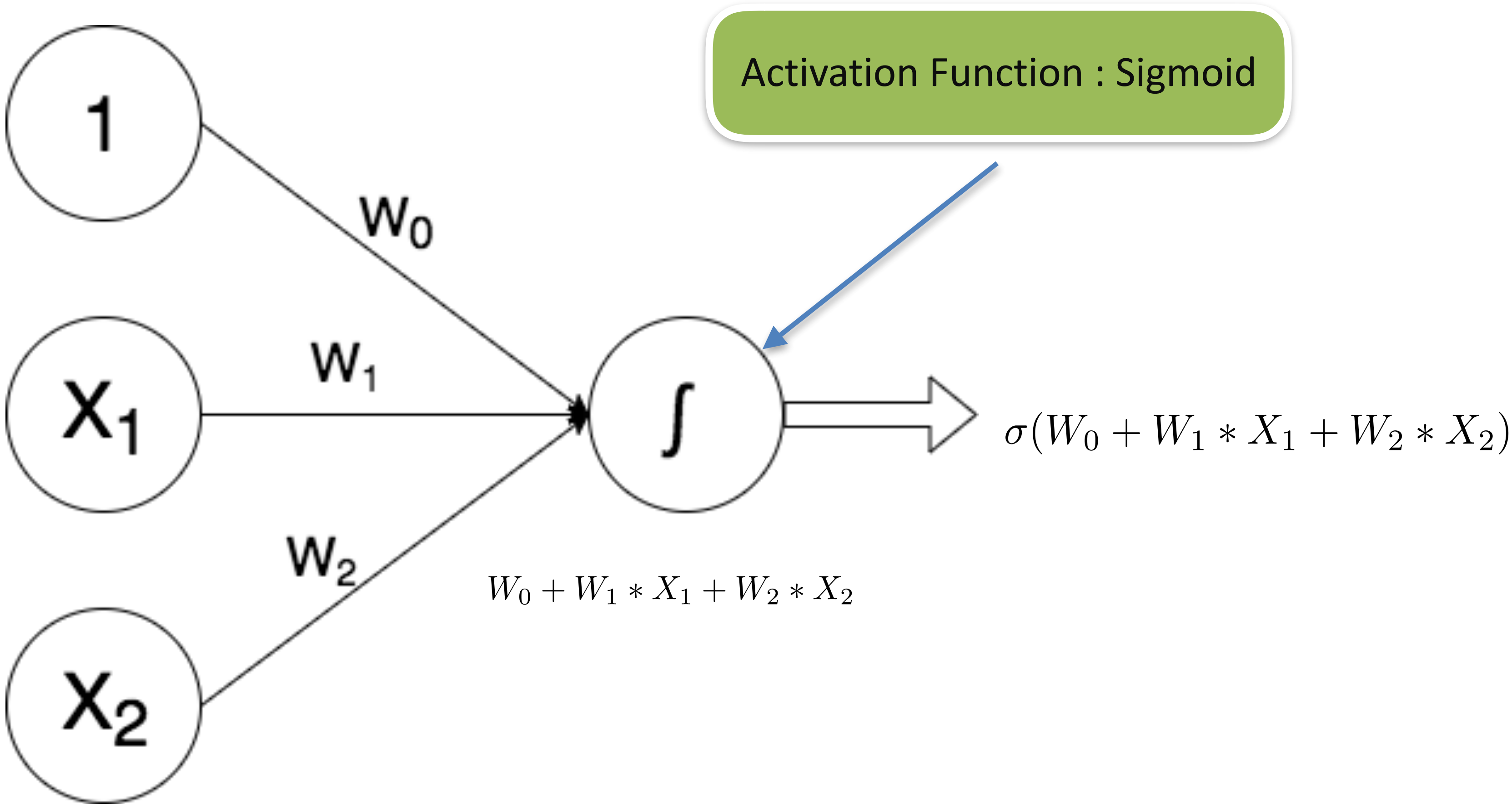
Agenda

Discussion Flow

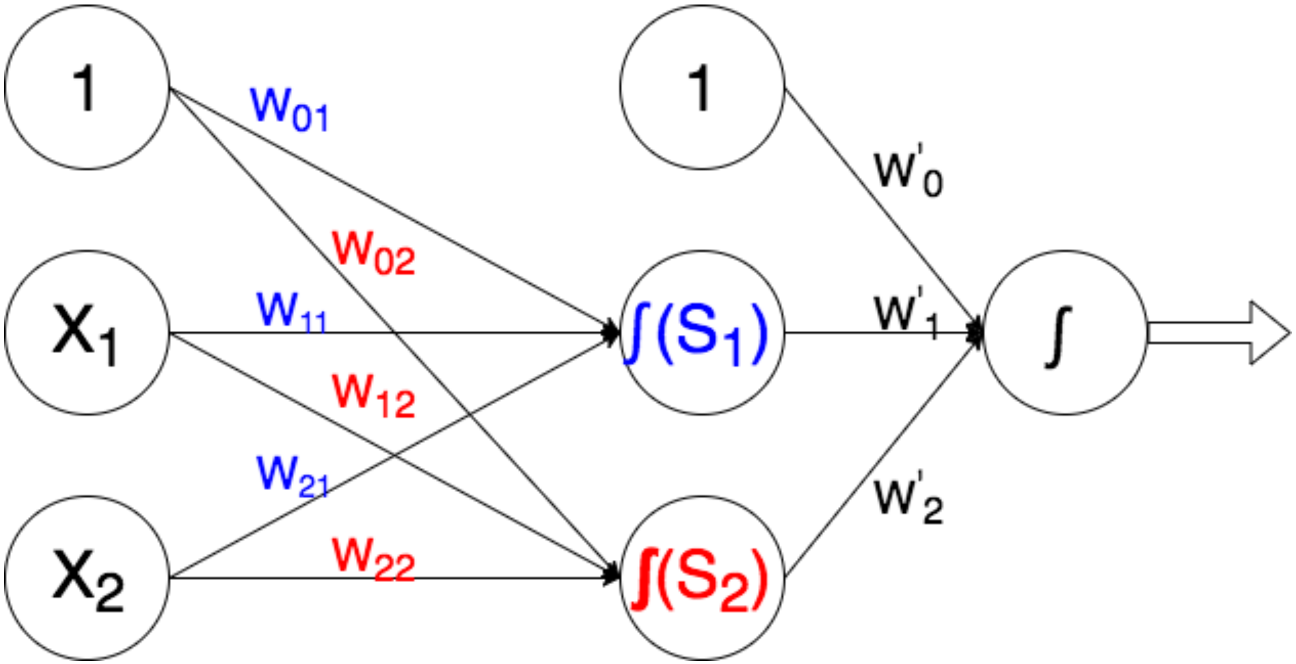
- Understanding Neural Network Representation
- Parameter Estimation with Gradient Descent
- Back Propagation
- Parameter Tuning with sklearn

Neural Network Representation

Logistic Regression as Neural Network



Adding a hidden layer



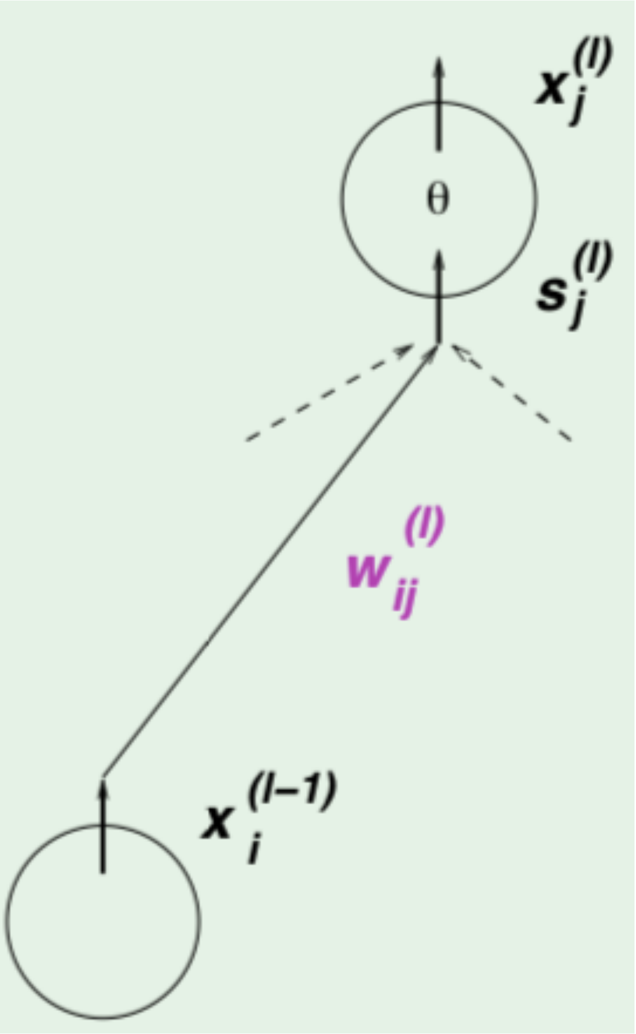
$$\sigma(W_0' + W_1' * \sigma(S_1) + W_2' * \sigma(S_2))$$

$$S_j = W_{0j} + W_{1j} * X_1 + W_{2j} * X_2$$

General Representation

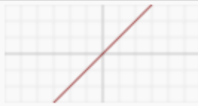
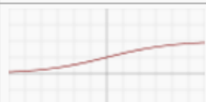
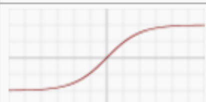
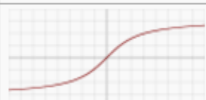
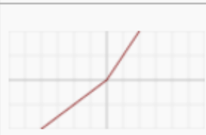
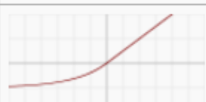
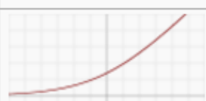
$$w_{ij}^{(l)} \quad \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ 1 \leq j \leq d^{(l)} & \text{outputs} \end{cases}$$

$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} \, x_i^{(l-1)}\right)$$



Reference : http://work.caltech.edu/slides/slides10.pdf

Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Gradient Descent & BackPropagation

Parameter Estimation

All the weights $\mathbf{w} = \{w_{ij}^{(l)}\}$ determine $h(\mathbf{x})$

Error on example $(\mathbf{x}_n, y_n)$ is

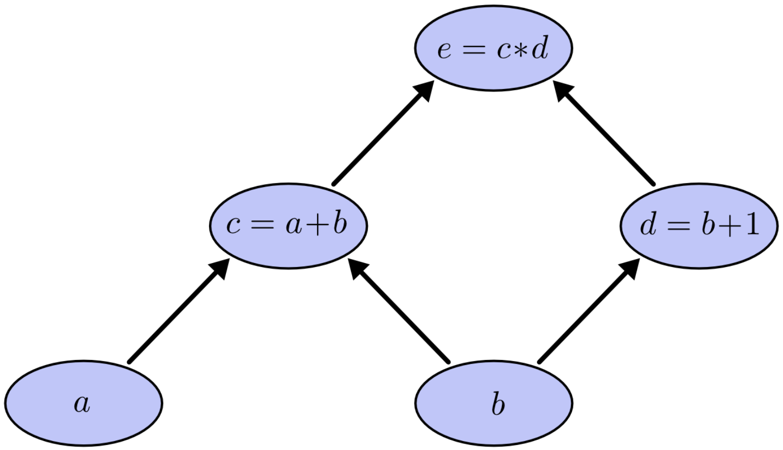
$$\mathbf{e}\left(h(\mathbf{x}_n), y_n\right) = \mathbf{e}(\mathbf{w})$$

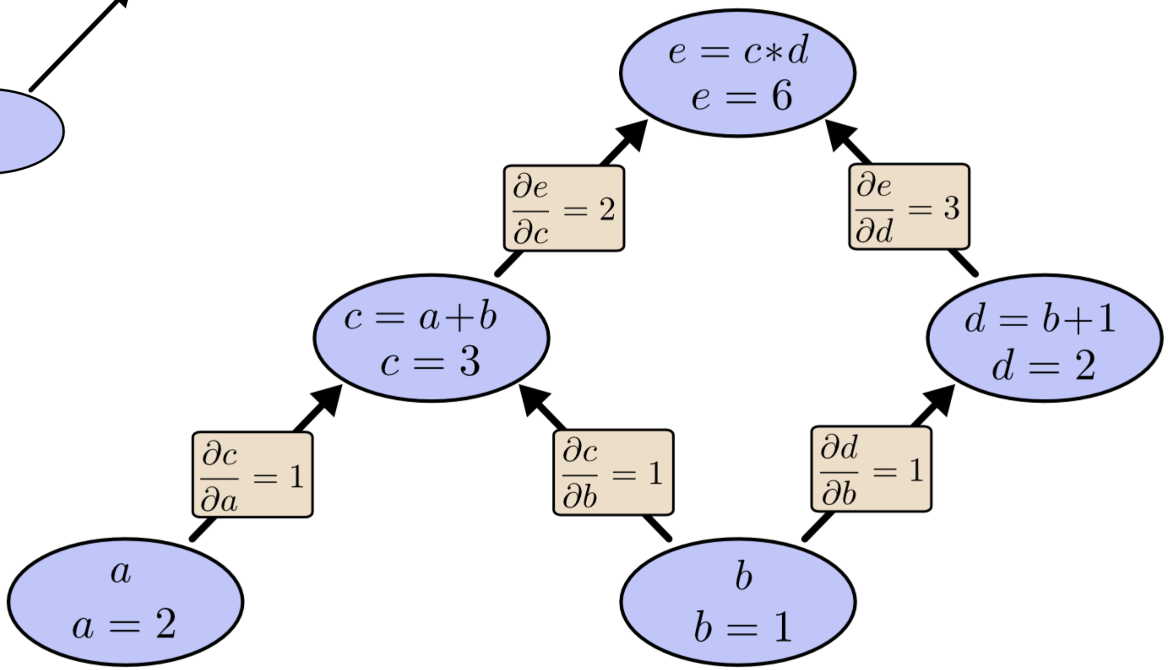
To implement SGD, we need the gradient

$$\nabla \mathbf{e}(\mathbf{w}): \quad \frac{\partial\ \mathbf{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}} \quad \text{for all } i, j, l$$

Reference : http://work.caltech.edu/slides/slides10.pdf

Back-Propagation





Reference : http://colah.github.io/posts/2015-08-Backprop/

Back Propagation for NN

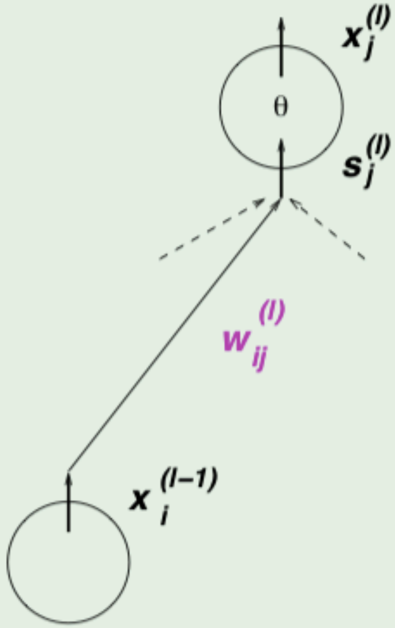
Computing $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}}$

We can evaluate $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}}$ one by one: analytically or numerically

A trick for efficient computation:

$$\frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}} = \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_j^{(l)}} \times \frac{\partial\ s_j^{(l)}}{\partial\ w_{ij}^{(l)}}$$

We have $\dfrac{\partial\ s_j^{(l)}}{\partial\ w_{ij}^{(l)}} = x_i^{(l-1)}$ We only need: $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_j^{(l)}} = \delta_j^{(l)}$



Reference : http://work.caltech.edu/slides/slides10.pdf

Contd..

δ for the final layer

$$\delta_j^{(l)} = \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_j^{(l)}}$$

For the final layer $l = L$ and $j = 1$:

$$\delta_1^{(L)} = \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_1^{(L)}}$$

$$\mathrm{e}(\mathbf{w}) = (\ x_1^{(L)} - y_n)^2$$

$$x_1^{(L)} = \theta(s_1^{(L)})$$

$$\theta'(s) = 1 - \theta^2(s)$$ for the tanh

Reference : http://work.caltech.edu/slides/slides10.pdf

Contd..

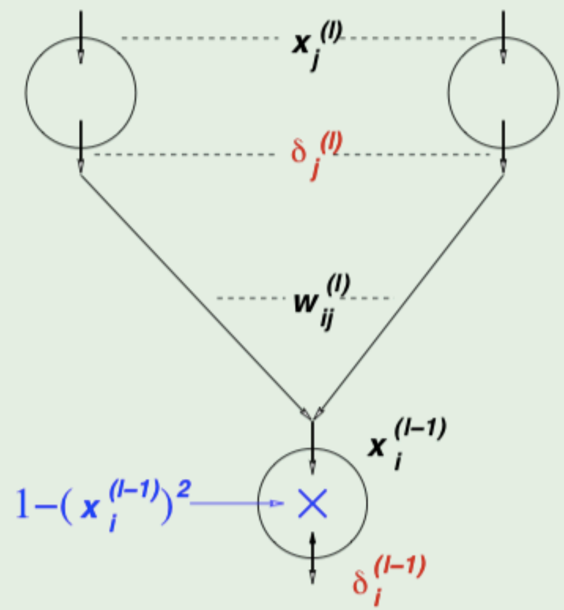
Back propagation of δ

$$\delta_i^{(l-1)} = \frac{\partial\, \mathbf{e}(\mathbf{w})}{\partial\, s_i^{(l-1)}}$$

$$= \sum_{j=1}^{d^{(l)}} \frac{\partial\, \mathbf{e}(\mathbf{w})}{\partial\, s_j^{(l)}} \times \frac{\partial\, s_j^{(l)}}{\partial\, x_i^{(l-1)}} \times \frac{\partial\, x_i^{(l-1)}}{\partial\, s_i^{(l-1)}}$$

$$= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)})$$

$$\delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)}\, \delta_j^{(l)}$$



Reference : http://work.caltech.edu/slides/slides10.pdf

EDVANCER EDUVENTURES

Contd..

Initialize all weights $w_{ij}^{(l)}$ at random

for $t = 0, 1, 2, \ldots$ do

  Pick $n \in \{1, 2, \cdots, N\}$

  Forward: Compute all $x_j^{(l)}$

  Backward: Compute all $\delta_j^{(l)}$

  Update the weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta\, x_i^{(l-1)} \delta_j^{(l)}$

  Iterate to the next step until it is time to stop

Return the final weights $w_{ij}^{(l)}$

Reference : http://work.caltech.edu/slides/slides10.pdf

Lets see it in action in Python