Back Propagation (1)

1. input and output neurons

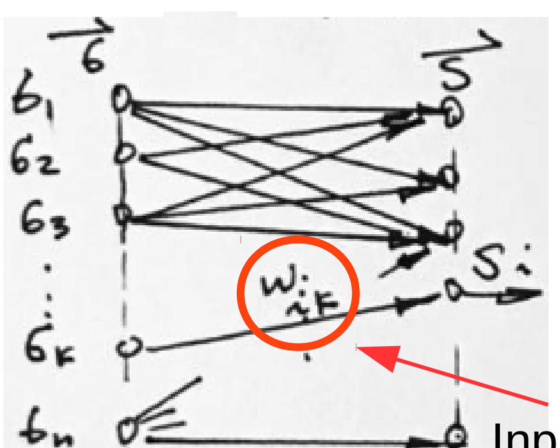
Input

$$\vec{\sigma} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_n \end{bmatrix}_{n\times 1}$$

Output

$$\vec{S} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{n_o} \end{bmatrix}_{n_o\times 1}$$

The architecture



Input: k
output: i

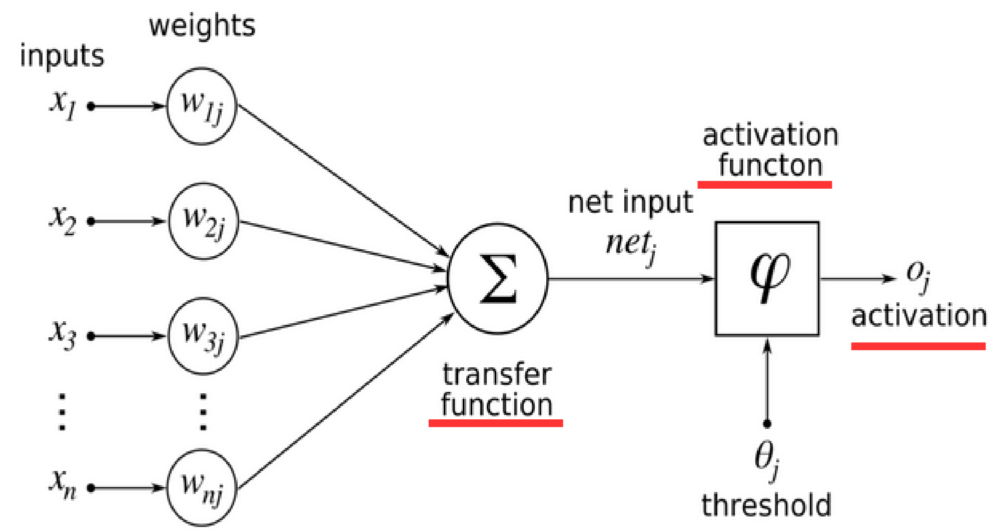
2. weights w_ik

i for output

k for input

Other popular notation

$$Y = \sum (weight * input) + bias$$



3. Activation function

$$\boxed{S_i = f\left(\sum_k \omega_{ik} \sigma_k\right)} \qquad \ldots (1)$$

Note the activation function f(.) and the bias (offset can be written in the unified summation term)

Back Propagation (2)

4. Transfer function h_i

let

$$h_i = \sum_k w_{ik} \sigma_k \quad \ldots (2)$$

Index i for the output neuron

Neuron output function S_i

$$S_i = f(h_i) = f[h_i(w_{ik})] \quad \ldots (2\text{-}1)$$

Neuron output is tied to activation function f(.), transfer function h_i and weights w_ik

5. Error at each neuron output (the difference between the true output Zeta (desired true output) and the current output S_i) at the experiment Mu

$$\zeta_i^\mu - S_i^\mu \quad \ldots (2\text{-}2)$$

6. total error for all output neuron I and all experiments Mu

$$D = \frac{1}{2} \sum_\mu \sum_i (\zeta_i^\mu - S_i^\mu)^2 \quad \ldots (3)$$

7. minimize the error wrt to w_ik

$$\frac{\partial D}{\partial w_{ik}} = \frac{\partial}{\partial w_{ik}} \cdot \frac{1}{2} \sum_\mu \sum_i [\zeta_i^\mu - f(h_i^\mu)]^2$$

$$= \sum_\mu [\zeta_i^\mu - f(h_i^\mu)] f'(h_i^\mu) \frac{\partial h_i}{\partial w_{ik}}$$

8. Training the NN by updating the weights

$$w_{ik}(t+1) = w_{ik}(t) + \delta w_{ij}(t) \quad \ldots (4)$$

where

$$\delta w_{ij}(t) = -\epsilon \frac{\partial D}{\partial w_{ij}} \quad \ldots (5)$$

Back Propagation (3)

9. computation of the derivatives to update the weight

$$\begin{pmatrix} \frac{\partial D}{\partial w_{i1}} \\ \frac{\partial D}{\partial w_{i2}} \\ \vdots \\ \frac{\partial D}{\partial w_{in}} \end{pmatrix} = \begin{pmatrix} [\xi_i^{\mu} - f(h_i^{\mu})] f' \frac{\partial h_i^{\mu}}{\partial w_{i1}} \\ [\xi_i^{\mu} - f(h_i^{\mu})] f' \frac{\partial h_i^{\mu}}{\partial w_{i2}} \\ \vdots \\ [\xi_i^{\mu} - f(h_i^{\mu})] f' \frac{\partial h_i^{\mu}}{\partial w_{in}} \end{pmatrix}$$

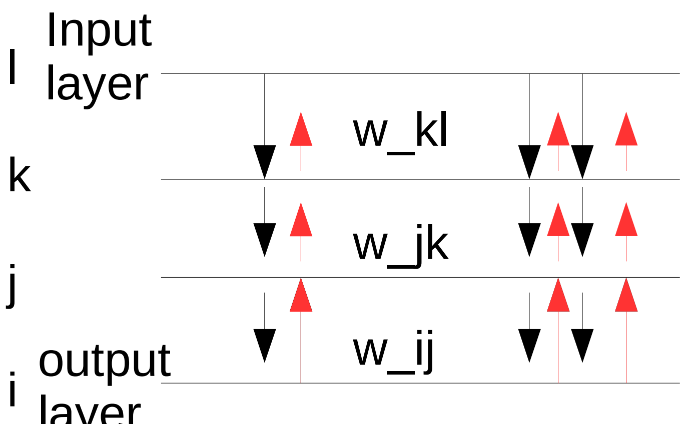
Update sequence back to the front:

w_ij → w_jk → w_kl

The state of the output functions for the layers i (output) and j input:

$$S_i = f(h_i) \qquad h_i = \sum_j w_{ij} S_j - \theta_i$$

10. feed forward NN (4 layers)



Black for the input; red for the back prop training direction

Harry Li, Ph.D.

Back Propagation (4)

11. chain rule, updating the weights (training)

For layer i

$$\frac{\partial D}{\partial w_{ij}} = \sum_{\mu} \left[\zeta_i^{\mu} - f(h_i^{\mu})\right] f'(h_i^{\mu}) \frac{\partial h_i^{\mu}}{\partial w_{ij}}$$

For layer j

$$\frac{\partial D}{\partial w_{jk}} = \sum_{\mu} \sum_{i} \left[\zeta_i^{\mu} - f(h_i^{\mu})\right] f'(h_i^{\mu}) \frac{\partial h_i^{\mu}}{\partial S_j} \cdot \frac{\partial S_j}{\partial w_{jk}}$$

For layer k

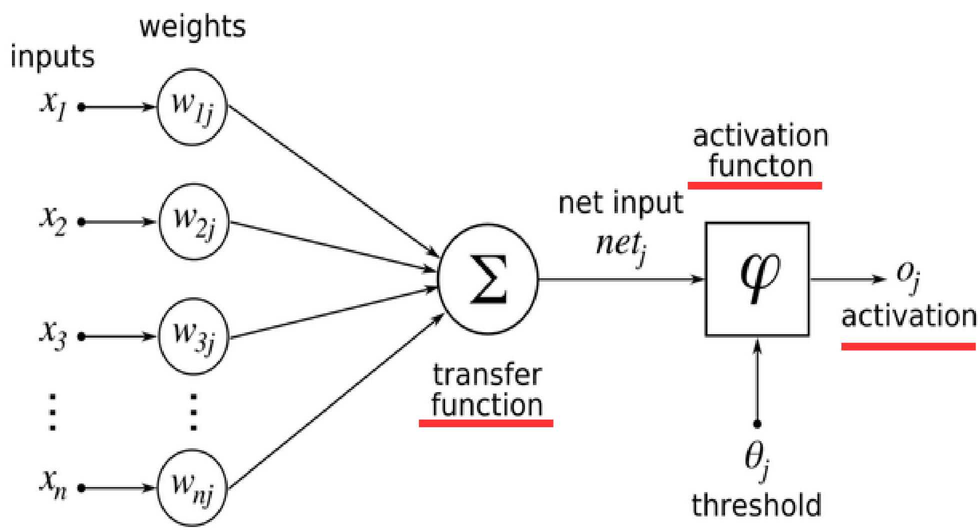
$$\frac{\partial D}{\partial w_{kl}} = \sum_{\mu} \sum_{i} \sum_{j} \left[\zeta_i^{\mu} - f(h_i^{\mu})\right] f'(h_i^{\mu}) \frac{\partial h_i^{\mu}}{\partial S_j} \cdot \frac{\partial S_j}{\partial S_k} \cdot \frac{\partial S_k}{\partial w_{kl}}$$

Note: the training described here all related to the derivative to the activation function, f'(.). So selection of the activation function is important and will be discussed in details next.

Activation Functions

Definition: for single neuron for the purpose of generating the output of the neuron.

$$Y = \sum (weight * input) + bias$$



Type: over hundreds proposed, we will focus On the following 4 types, Sigmoid, Softmax, Tanh, ReLU, (SSTR)

Characteristics and Comparison:

Sigmoid $$\sigma(x) = \frac{1}{1+e^{-x}}$$

Tanh $$tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$$

ReLU $$f(x) = max(0, x)$$

Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1, 2, \ldots, K$$

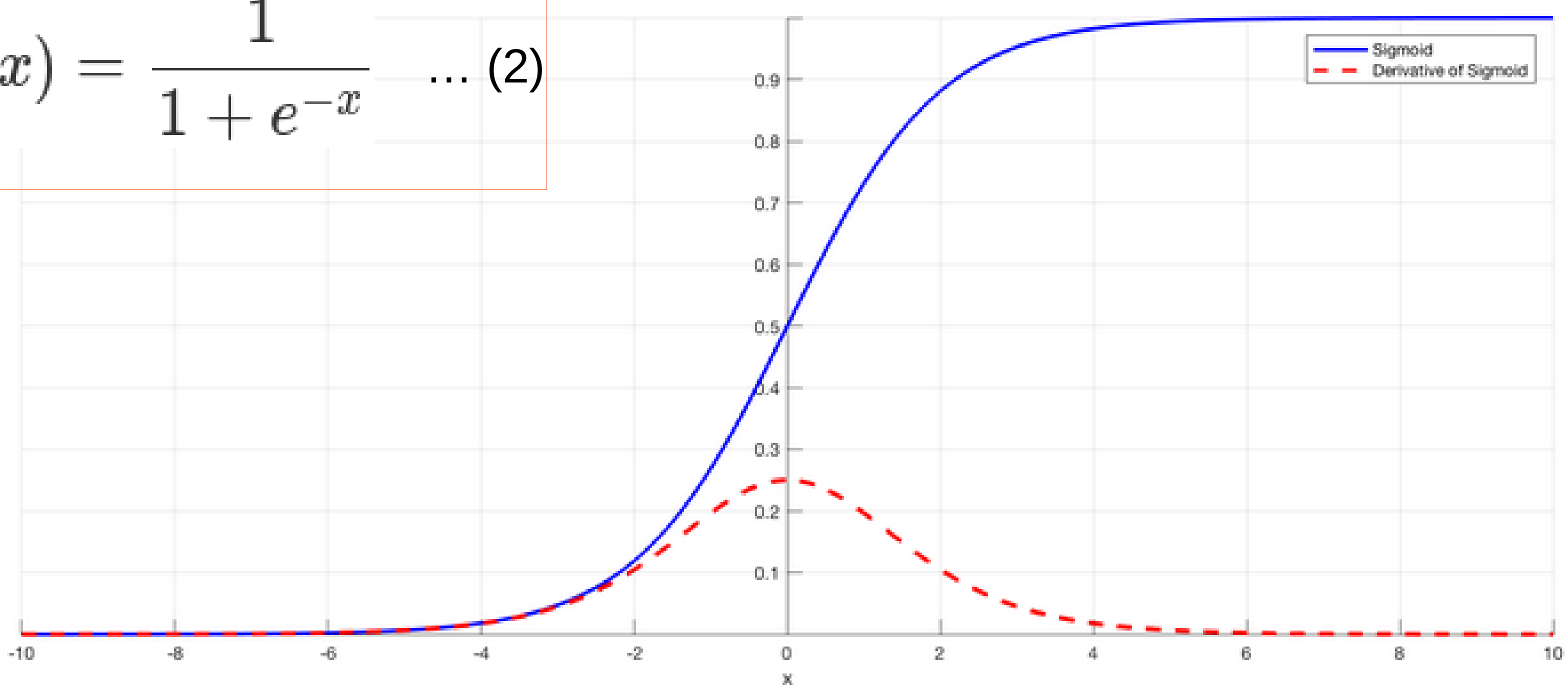
Sigmoid Functions

https://isaacchanghau.github.io/post/activation_functions/

$$\boxed{\sigma(x) = \frac{L}{1 + e^{-k(x - x_0)}}} \quad \ldots (1)$$

Logistic function in general as in equation (1)

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \ldots (2)$$

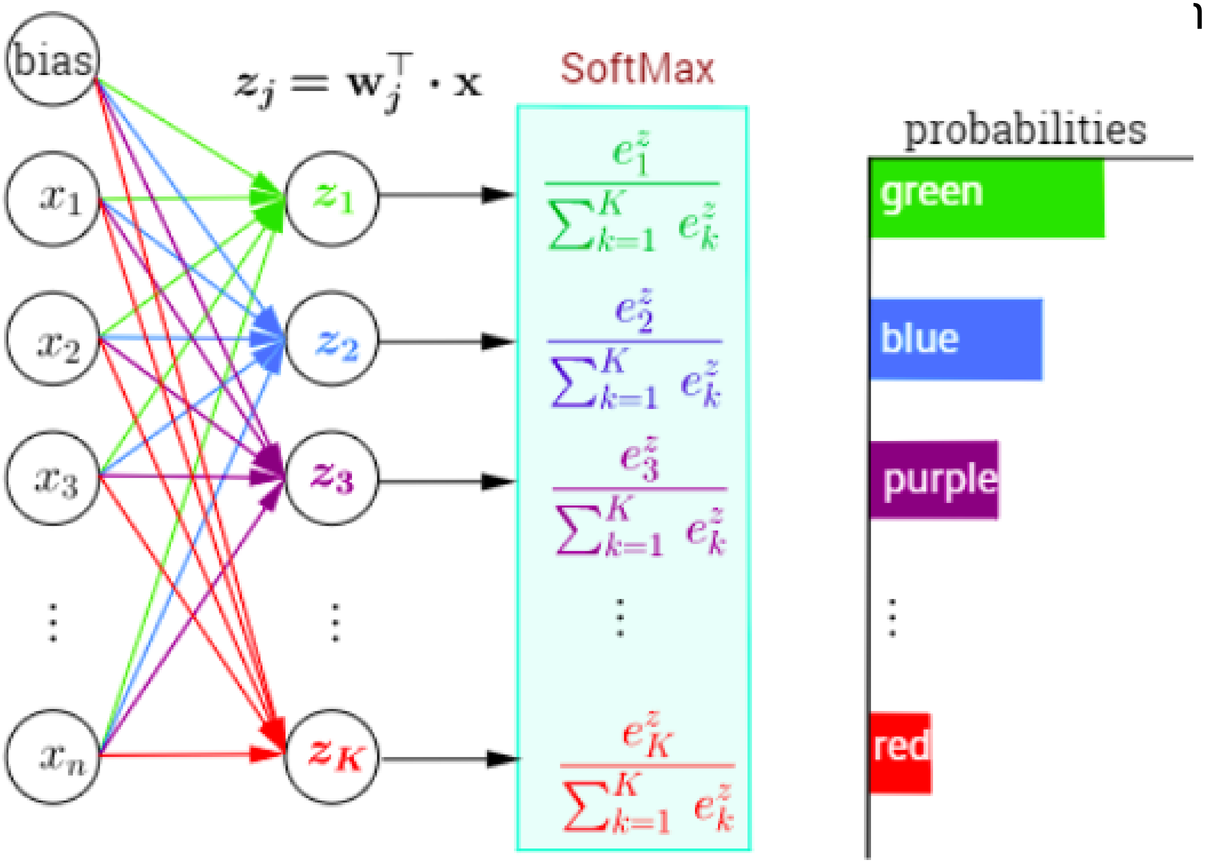


Softmax Functions

https://isaacchanghau.github.io/post/activation_functions/

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1, 2, \ldots, K \quad \ldots (1)$$

used in various multiclass classification methods, such as multinomial logistic regression, multiclass linear discriminant analysis, naive Bayes classifiers, ıd artificial neural networks.



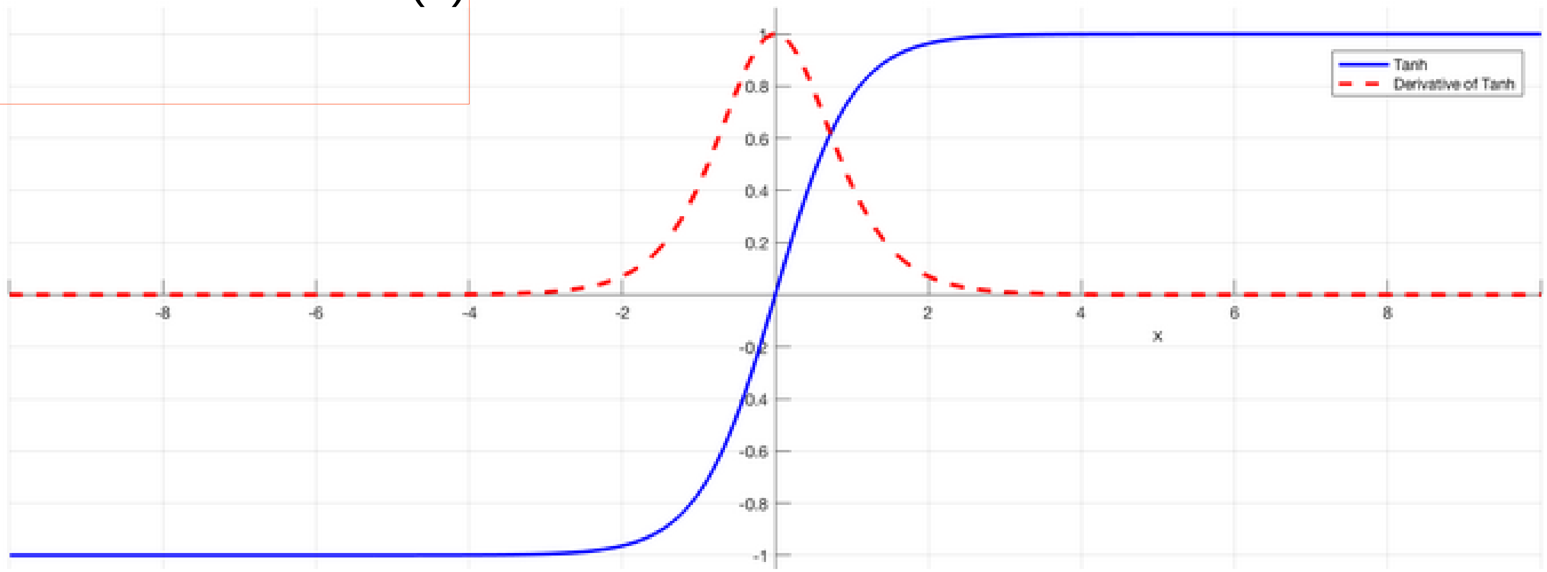
tanh Functions

https://isaacchanghau.github.io/post/activation_functions/

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$ … (1)

The derivative (red curve)

… (2)



Relu Functions

https://isaacchanghau.github.io/post/activation_functions/

$$\boxed{f(x) = max(0, x)} \quad \ldots (1)$$

One Relu example (green)

$$f(x) = \ln(1 + e^x) \quad \ldots (2)$$

Its derivative:

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