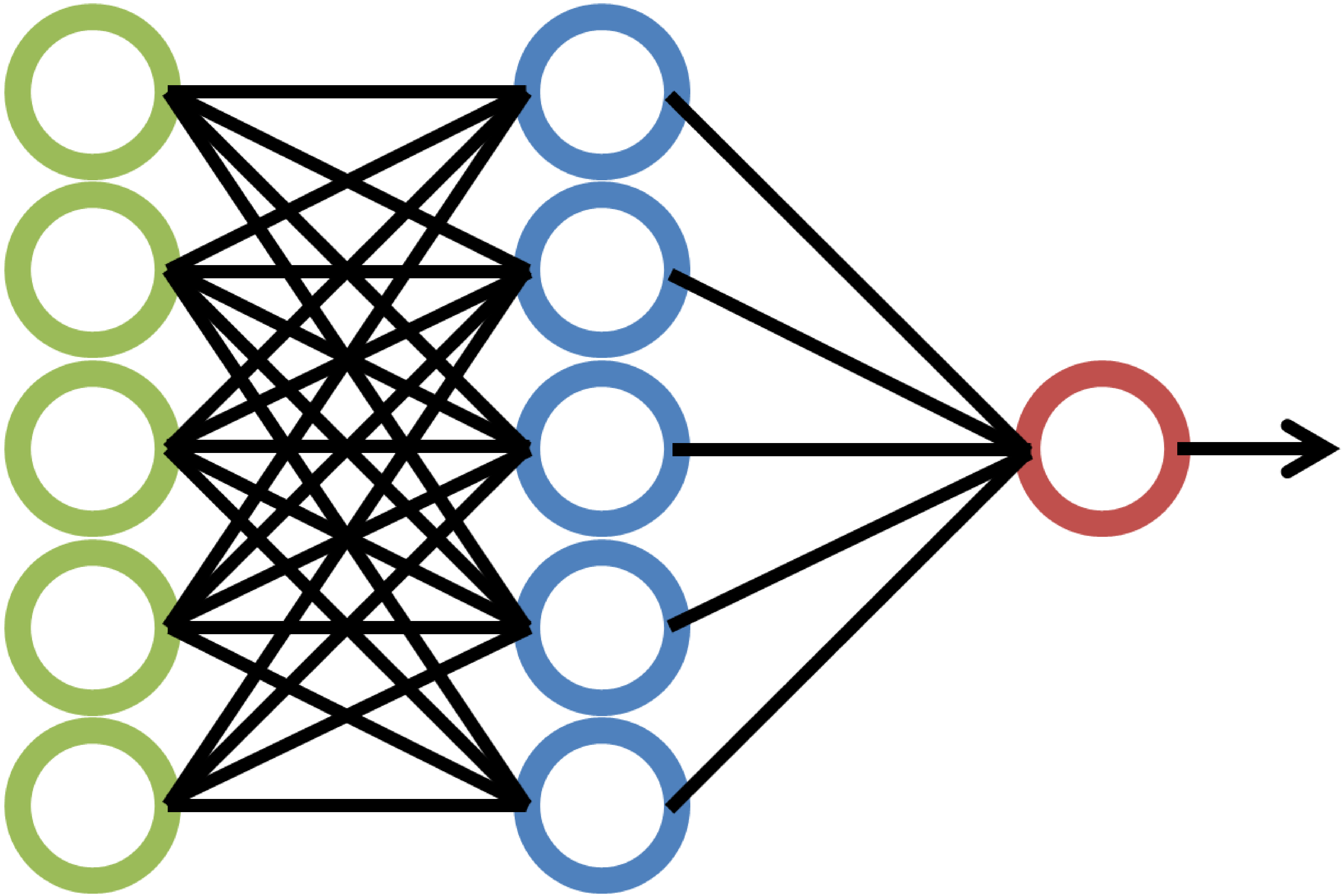
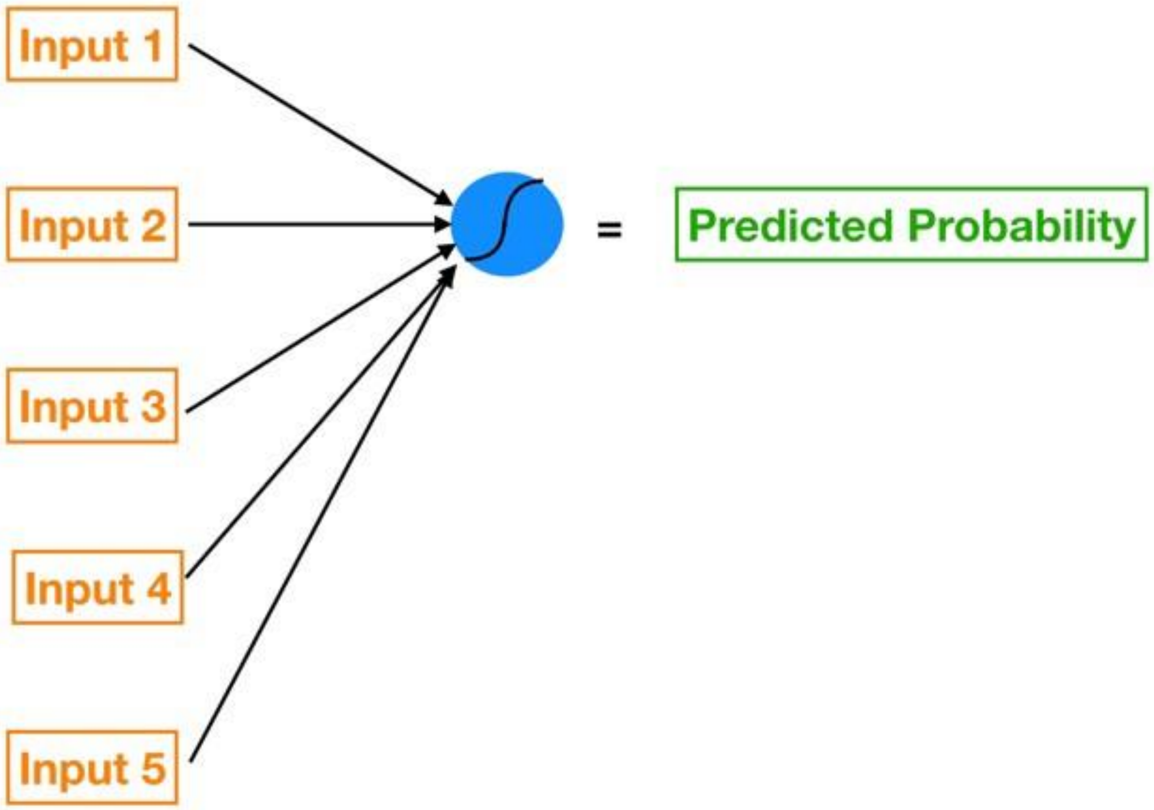
# Neural Networks

BY MG ANALYTICS



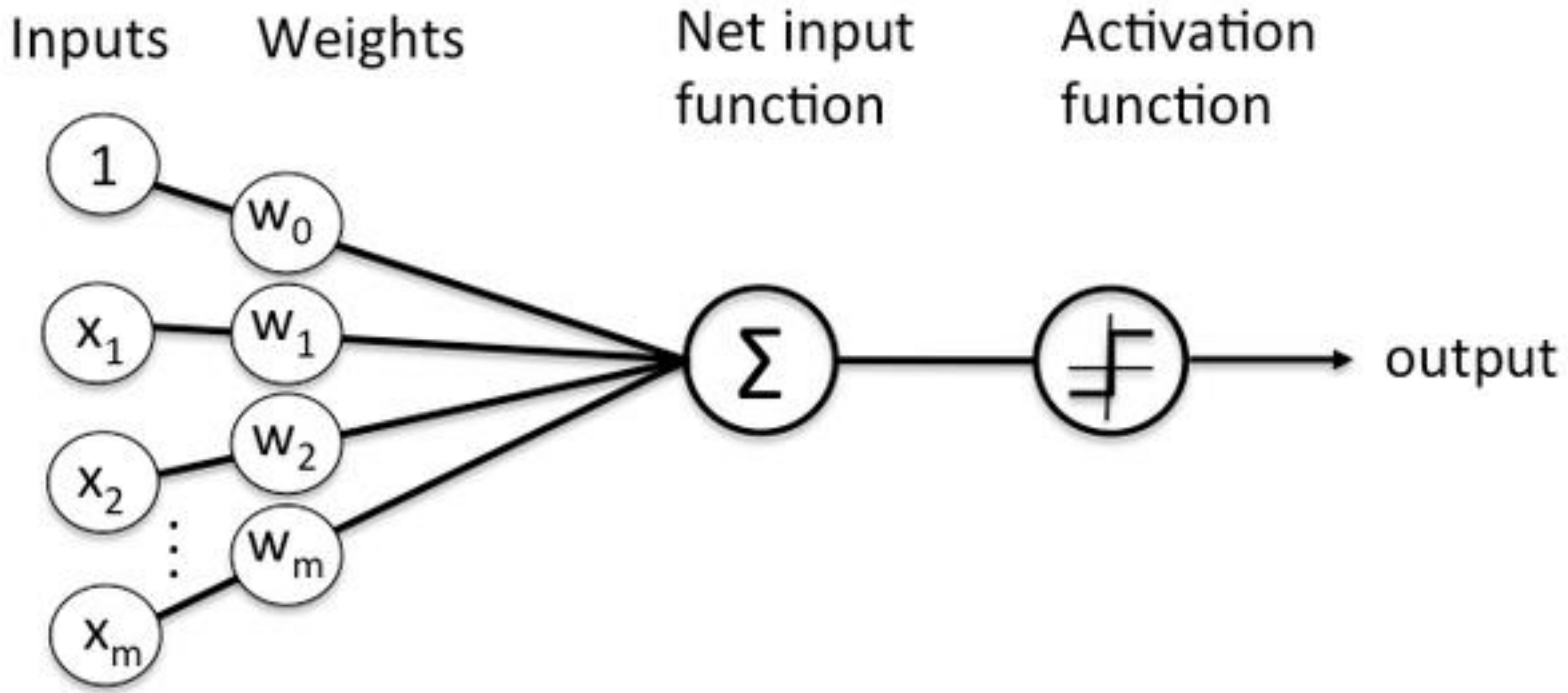

Input 1
Input 2
Input 3
Input 4
Input 5
=
Predicted Probability

# Neural networks

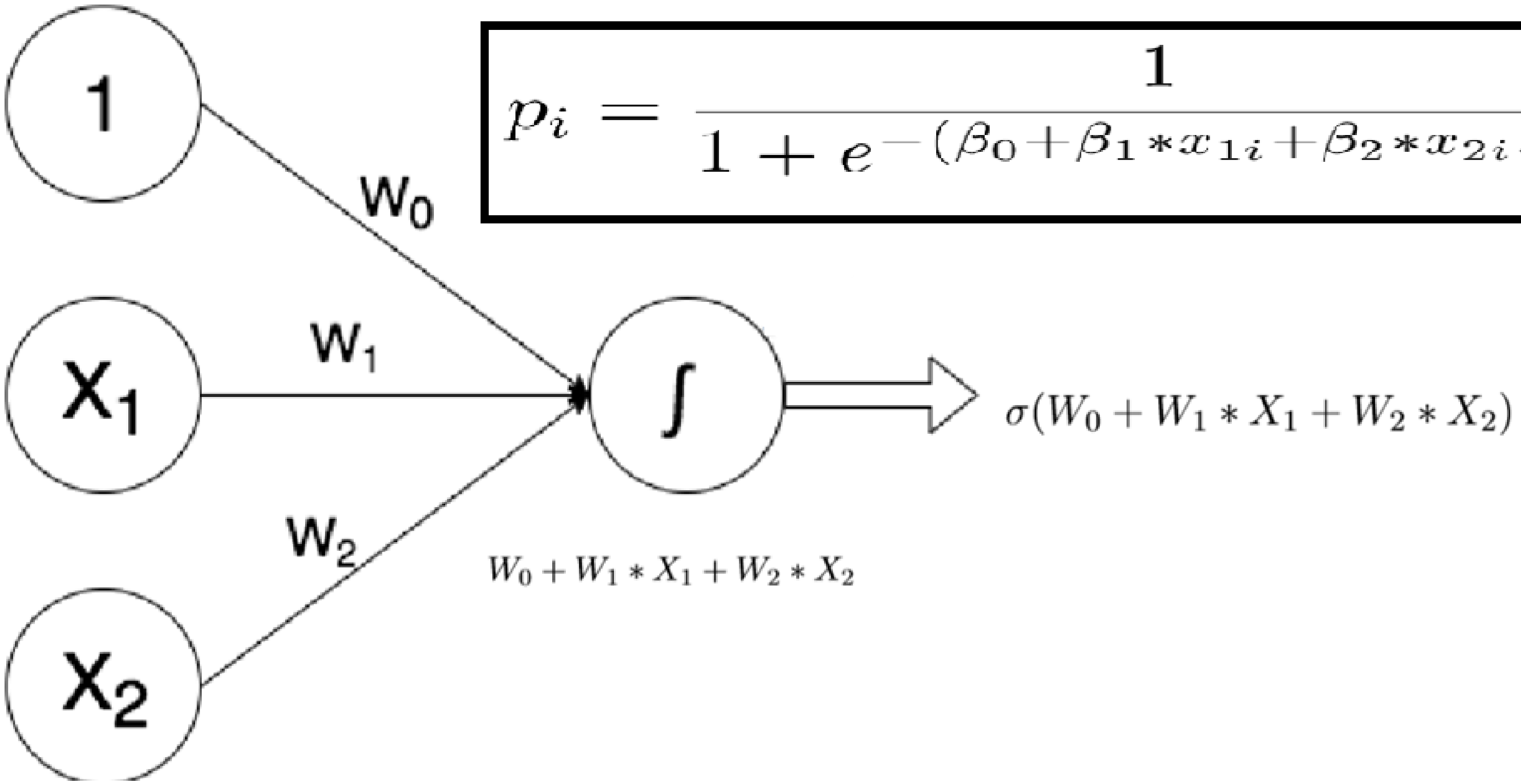


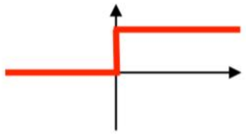
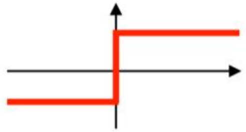
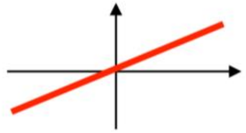
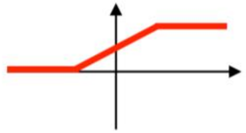
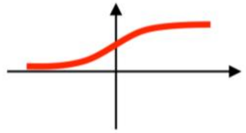
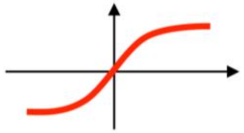
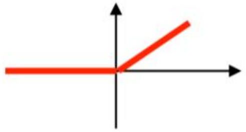
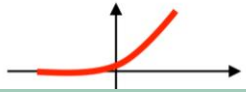
- Neural networks is an algorithm inspired by the neurons in our brain.
- It is designed to recognize patterns in complex data, and often performs the best when recognizing patterns in audio, images or video.
- A neural network simply consists of neurons (also called nodes).
- Then each neuron holds a number, and **each connection holds a weight.**
- Activation functions are usually non linear transforming functions which transform linear function into a non linear form which is capable of capturing complex patterns.

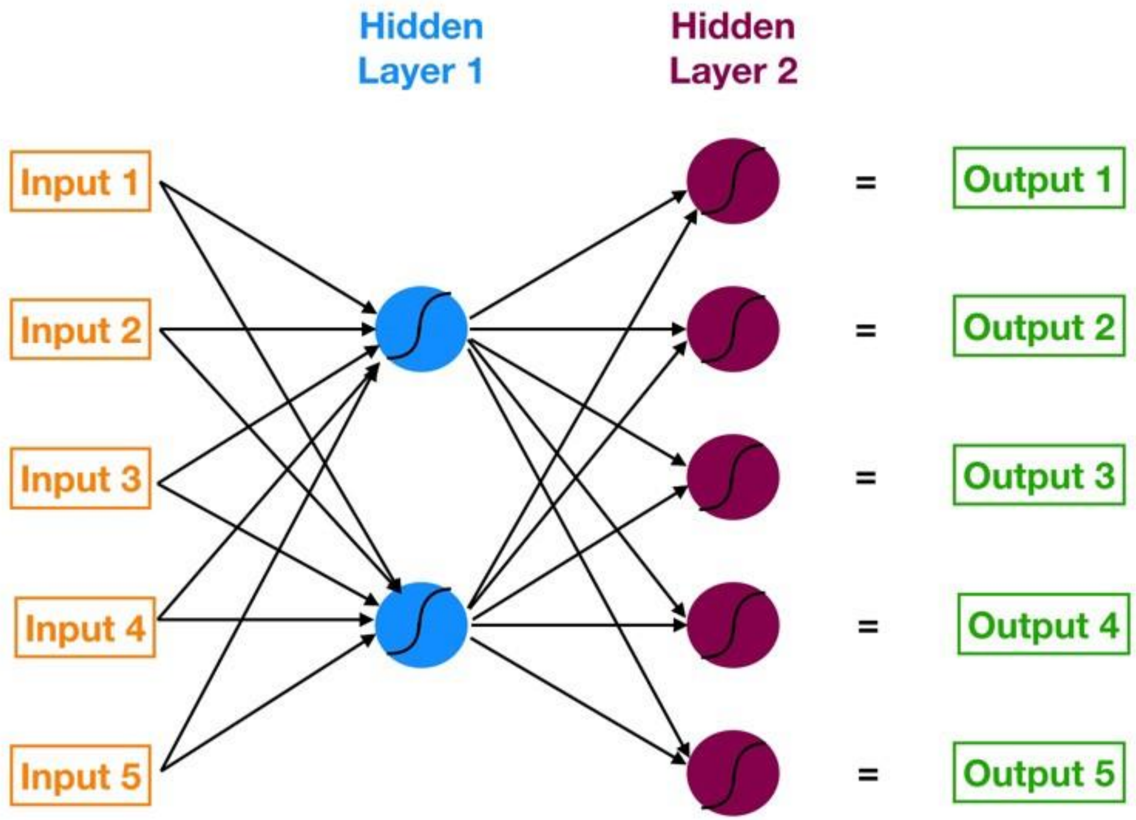
MG ANALYTICS

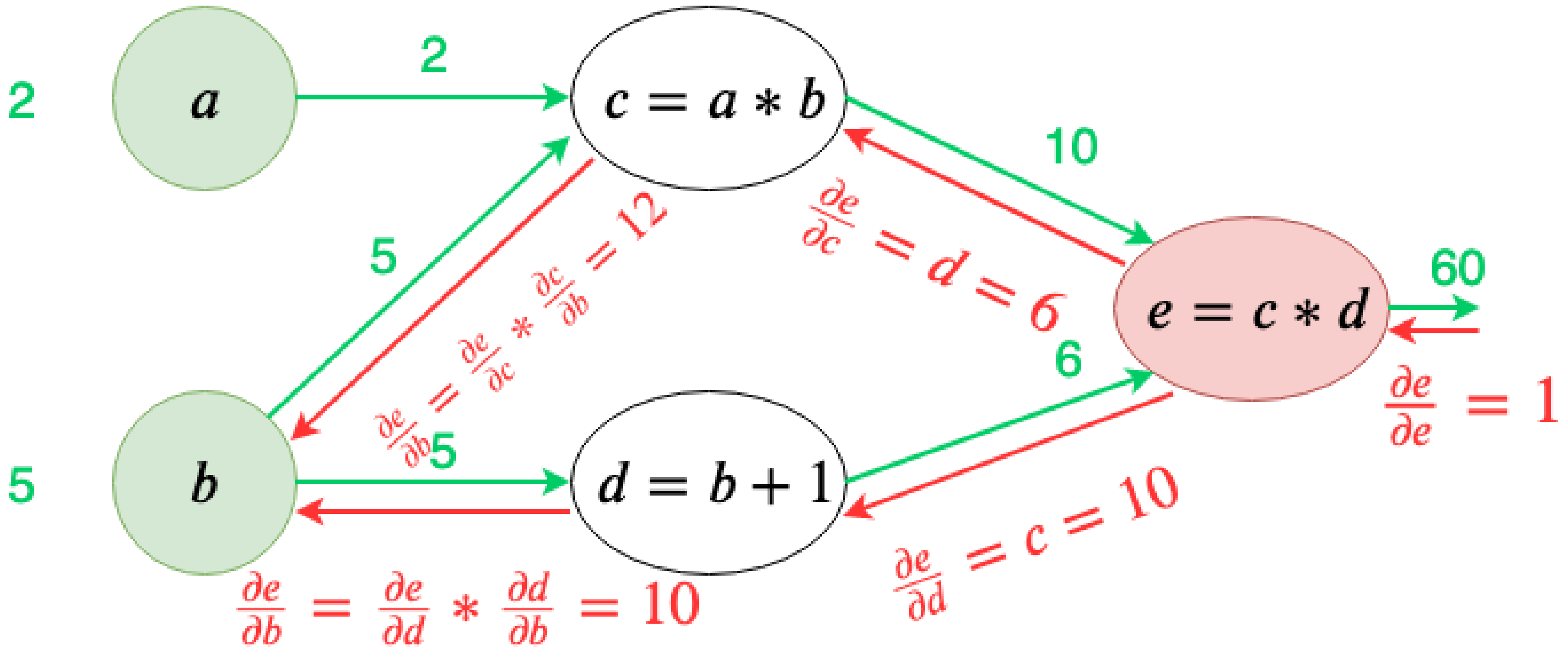

Inputs
Weights
Net input function
Activation function
1
$w_0$
$x_1$
$w_1$
$x_2$
$w_2$
$x_m$
$w_m$
$\Sigma$
output

$$\boxed{p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * x_{1i} + \beta_2 * x_{2i} \ldots)}}}$$



| Activation function | Equation | Example | 1D Graph |
| --- | --- | --- | --- |
| Unit step (Heaviside) | $\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$ | Perceptron variant |  |
| Sign (Signum) | $\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$ | Perceptron variant |  |
| Linear | $\phi(z) = z$ | Adaline, linear regression |  |
| Piece-wise linear | $\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$ | Support vector machine |  |
| Logistic (sigmoid) | $\phi(z) = \frac{1}{1 + e^{-z}}$ | Logistic regression, Multi-layer NN |  |
| Hyperbolic tangent | $\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ | Multi-layer Neural Networks |  |
| Rectifier, ReLU (Rectified Linear Unit) | $\phi(z) = max(0, z)$ | Multi-layer Neural Networks |  |
| Rectifier, softplus | $\phi(z) = \ln(1 + e^z)$ | Multi-layer Neural |  |


Hidden
Layer 1
Hidden
Layer 2
Input 1
Input 2
Input 3
Input 4
Input 5
=
=
=
=
=
Output 1
Output 2
Output 3
Output 4
Output 5



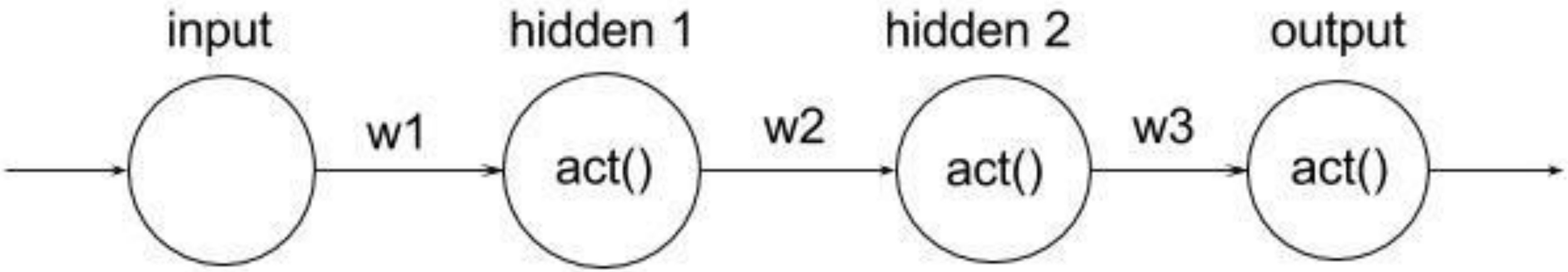

# Chain Rule

If $y = h(x) = g(u)$, where $u = f(x)$, then

$$\boxed{\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}}$$

# Backpropagation

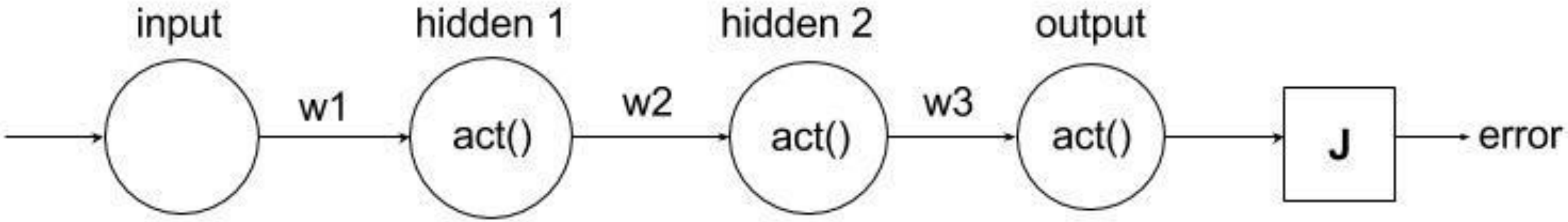
- Backpropagation is for calculating the gradients efficiently.
- We always start from the output layer and propagate backwards, updating weights and biases for each layer.
- adjust the weights and biases throughout the network, so that we get the desired output in the output layer.



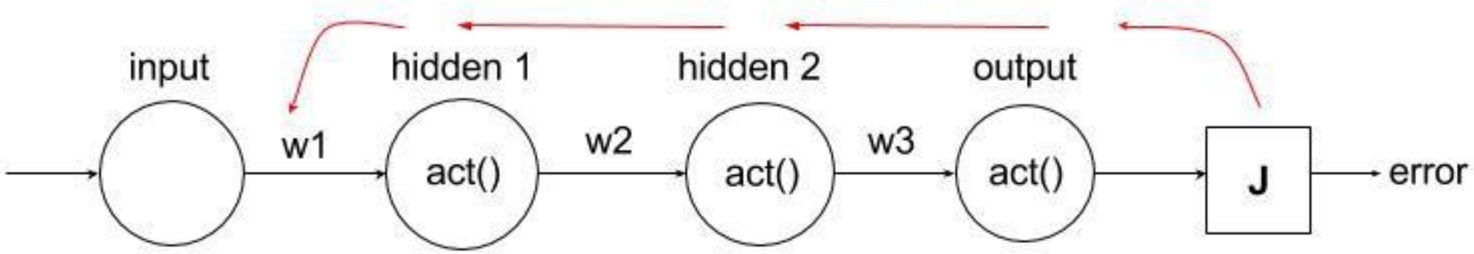

$$output = act(w3 * hidden2)$$
$$hidden2 = act(w2 * hidden1)$$
$$hidden1 = act(w1 * input)$$

$$output = act(w3 * act(w2 * act(w1 * input)))$$

$$\frac{\partial}{\partial w1} output = \frac{\partial}{\partial hidden2} output * \frac{\partial}{\partial hidden1} hidden2 * \frac{\partial}{\partial w1} hidden1$$




$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$



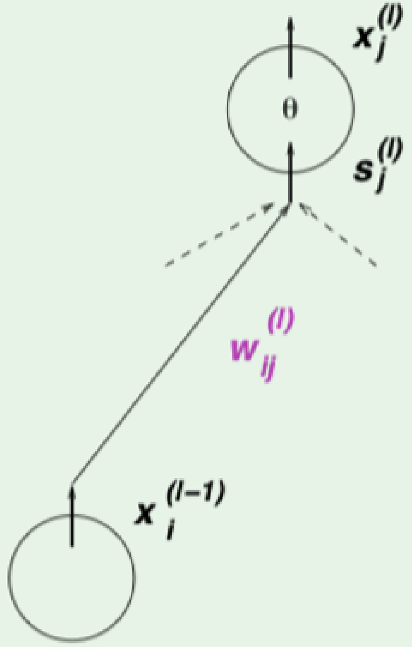
## Computing $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}}$

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



$\delta$ for the final layer

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

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

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

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

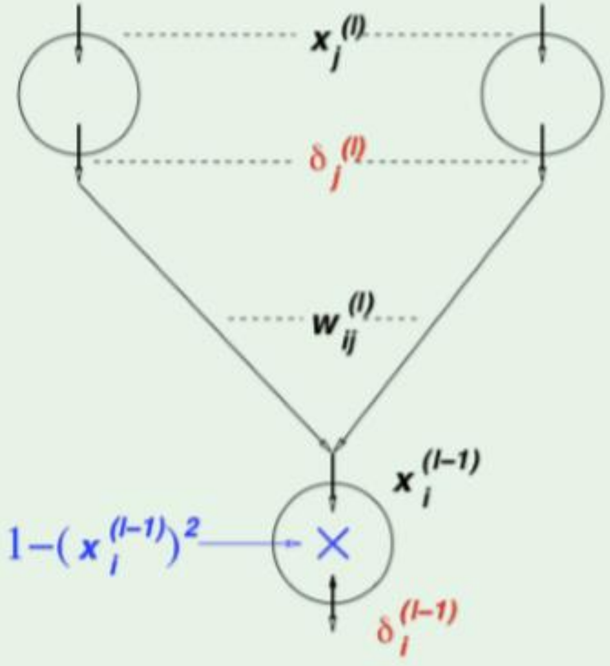
$$x_1^{(L)} = \theta(s_1^{(L)})$$

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

## Back propagation of $\delta$

$$\begin{aligned}
\delta_i^{(l-1)} &= \frac{\partial\, \mathbf{e}(\mathbf{w})}{\partial\, s_i^{(l-1)}} \\
&= \sum_{j=1}^{d^{(l)}} \frac{\partial\, \mathbf{e}(\mathbf{w})}{\partial\, s_j^{(l)}} \times \frac{\partial\, s_j^{(l)}}{\partial\, x_i^{(l-1)}} \times \frac{\partial\, x_i^{(l-1)}}{\partial\, s_i^{(l-1)}} \\
&= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)})
\end{aligned}$$

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



- Repeat :
  - Initialize weights to a small random number and let all biases be 0
  - **Forward** pass for next sample in mini-batch and calculate activations.
  - **Backward** pass calculate gradients and update gradient vector by iteratively propagating backwards through the neural network.
  - Update weights and biases based on the gradient vector calculated from averaging over the mini-batch.