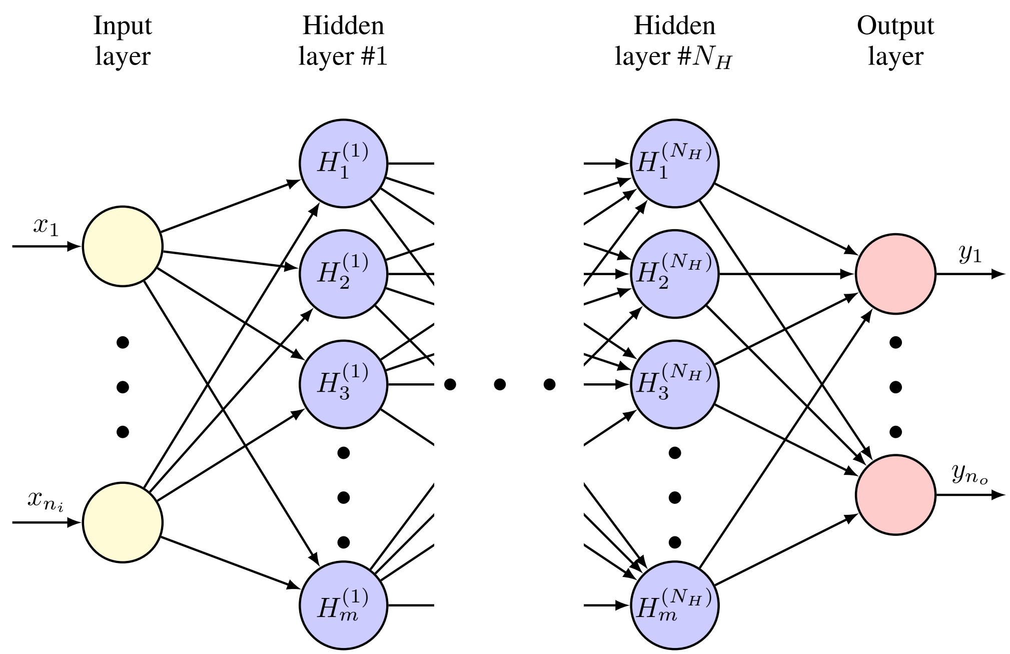
Neural Network

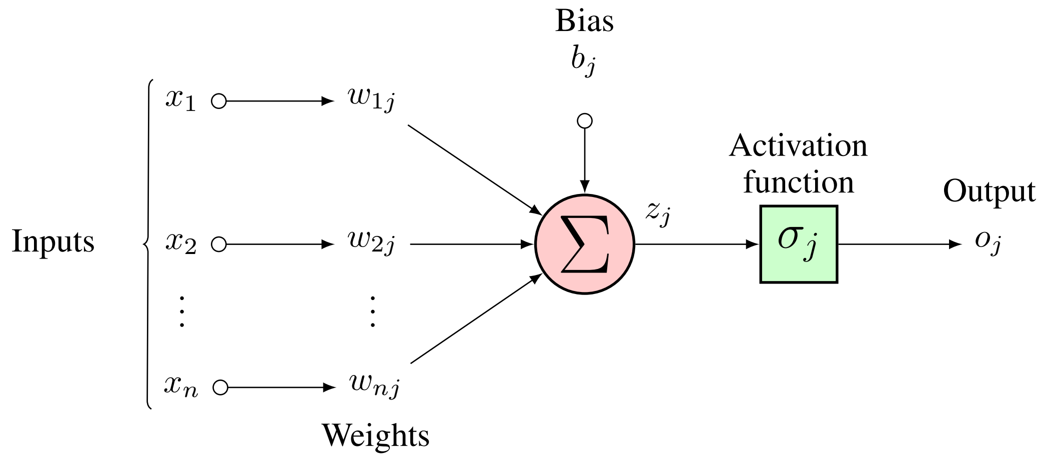
- Neural networks are used as universal approximator of functional relationship $y = f(x)$.
- **Feed-forward Neural Network**:



- The universal approximation theorem states that feed-forward neural networks with at least one hidden layer and large enough number of neurons, and differentiable activation functions, can approximate any continuous function on a compact support.

Neural Network

- A closer look at one hidden unit known as the neuron:



- For a single neuron:

$$o_j = \sigma_j \left(\sum_{i=1}^{n} w_{ij} x_i + b_j \right)$$

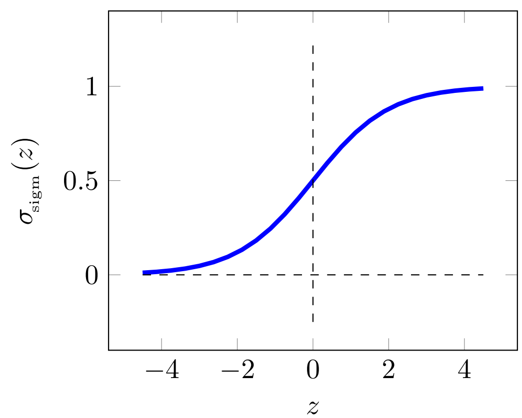
- For the full network:

$$y = \sum_{p=0}^{n_o} w_{0p} \left[\sigma_p^{(o)} \left(\ldots \sigma_k^{(2)} \left(\sum_{j=1}^{n_2} w_{kj}^{(2)} \left[\sigma_j^{(1)} \left(\sum_{i=1}^{n_1} w_{ij}^{(1)} x_i + b_j^{(1)} \right) \right] + b_k^{(2)} \right) \ldots \right) \right] + b_0$$

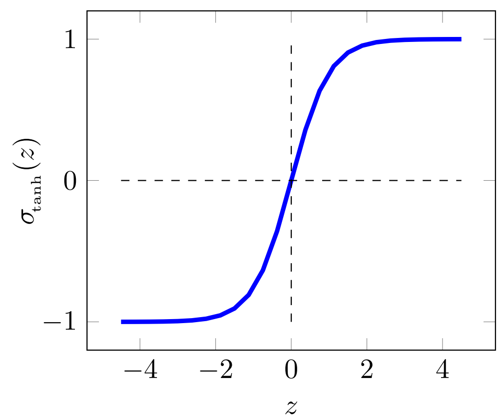
- In matrix form: $\boxed{y = \mathbf{W}_0 \left(\boldsymbol{\sigma} (\ldots \boldsymbol{\sigma} \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right) \ldots) \right) + b_0}$

Activation Function

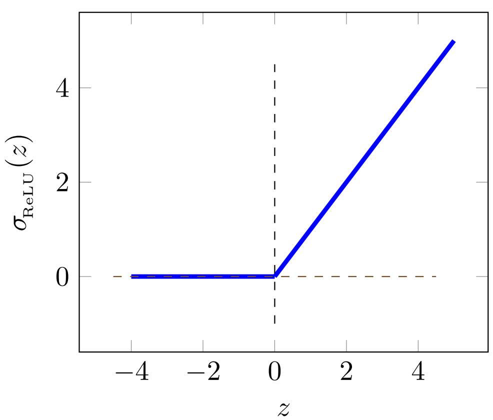
- Sigmoid: $\sigma_{sigm}(z) = \frac{1}{1+e^{-z}}$



- Tanh: $\sigma_{tanh}(z) = \tanh(z)$

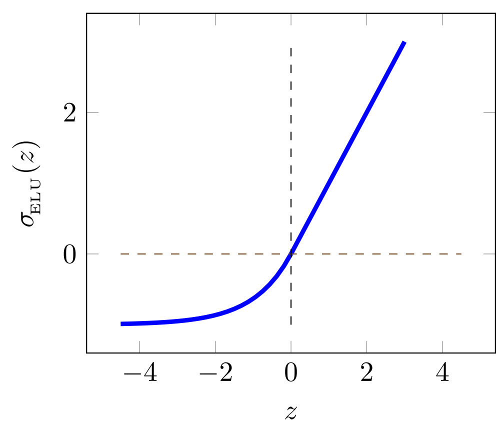


- Rectified Linear Unit (ReLU): $\sigma_{ReLU}(z) = \max(0, z)$



- Exponential Linear Unit (ELU):

$$\sigma_{ELU}(z) = \begin{cases} z & \text{for } z > 0, \\ \alpha(e^{z} - 1) & \text{for } z \leq 0, \end{cases}$$



Training of Neural Network

- The parameters of a neural network: $\boldsymbol{\theta} = \left[\{\mathbf{W}\}_{i=0}^{N_H}, \{\mathbf{b}\}_{i=0}^{N_H} \right]$.
- These parameters are selected by minimizing the error between the prediction of the network and measured data for a training dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$. This procedure is known as the training of the network.
- A common loss function used for training is the mean squared error (MSE) given by

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - y_{\mathrm{pred}}(\mathbf{x}_i) \right)^2$$

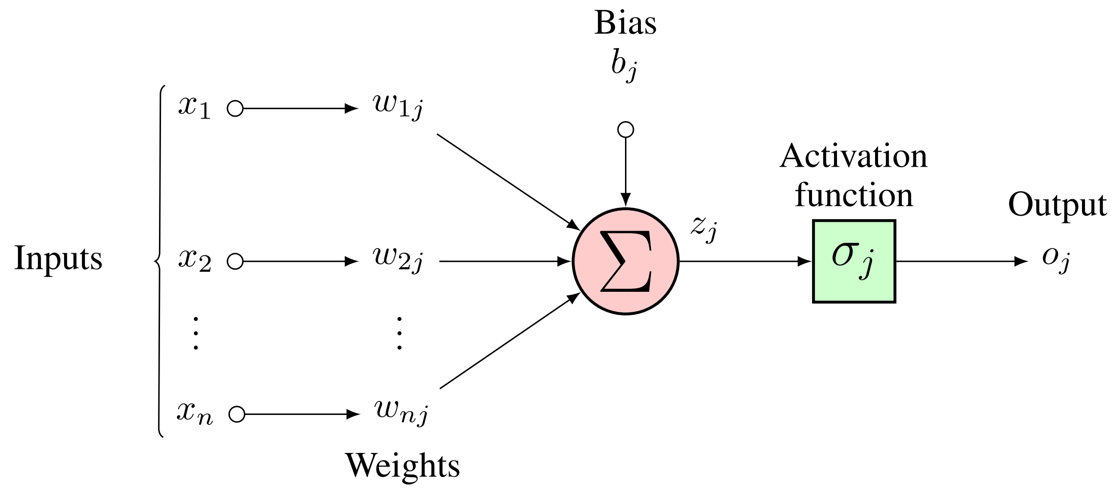
- **Stochastic gradient descent (SGD)** is an efficient strategy to update them

$$\boxed{\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \underset{\text{learning rate}}{\eta} \frac{\partial J(\boldsymbol{\theta}^{(k)})}{\partial \boldsymbol{\theta}}}$$

  - At every iteration, only a small batch of training data is used to estimate the gradients and update the parameters.

Backpropagation

- Backpropagation is used to estimate the derivative $\frac{\partial J}{\partial \boldsymbol{\theta}}$



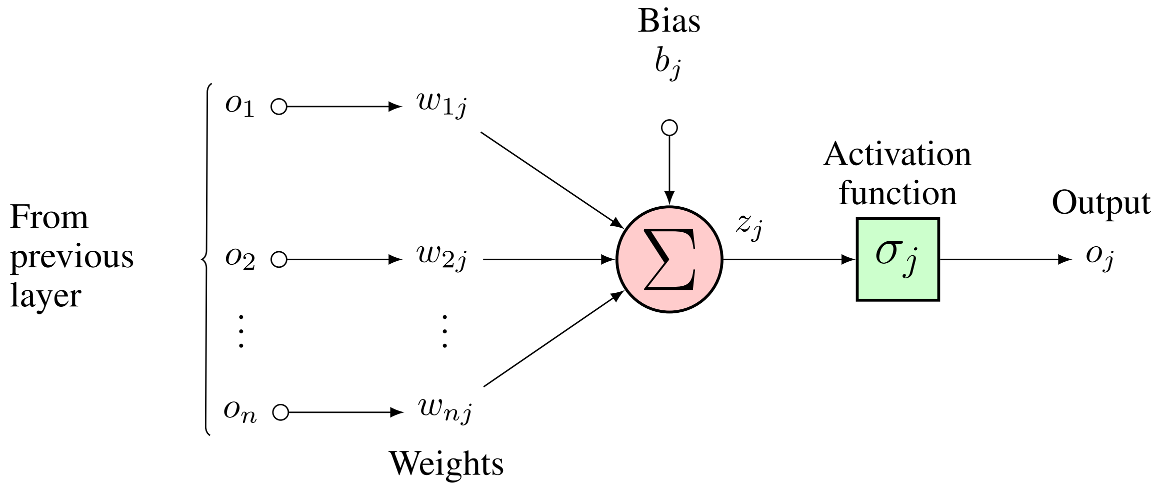
- Use chain rule from your Calculus course

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial J}{\partial o_j}\frac{\partial o_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}$$

- For just this one neuron
  - Third term: $z_j = \sum_{i=1}^{n} w_{ij}x_i + b_j \Rightarrow \frac{\partial z_j}{\partial w_{ij}} = x_i$
  - Second term: $\frac{\partial o_j}{\partial z_j}$ depends on the activation function used.

    For Sigmoid activation function $\frac{\partial o_j}{\partial z_j} = o_j(1 - o_j)$
  - First term: For $J = \frac{1}{2}(y_{\text{target}} - o_j)^2$, we have $\frac{\partial J}{\partial o_j} = o_j - y_{\text{target}}$

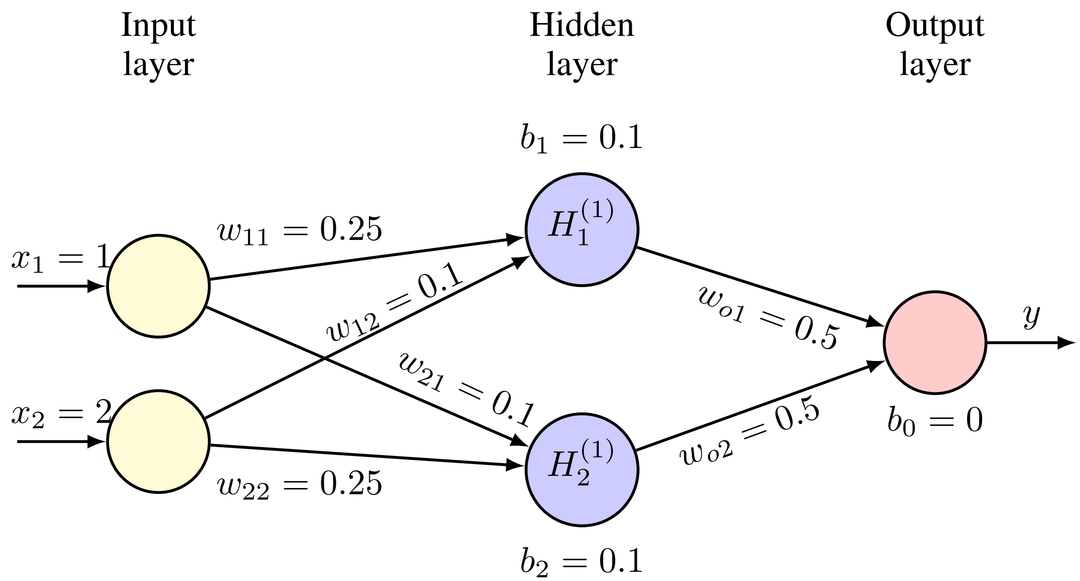
Backpropagation



- Backpropagation: $$\boxed{\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial o_j}\frac{\partial o_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}}$$
- For a neuron inside a hidden layer
  - Third term: $z_j = \sum_{i=1}^{n} w_{ij} o_i + b_j \Rightarrow \frac{\partial z_j}{\partial w_{ij}} = o_i$ from previous layer
  - Second term: $\frac{\partial o_j}{\partial z_j}$ depends on the activation function used.
  - First term:

$$\frac{\partial J}{\partial o_j} = \sum_p \left(\frac{\partial J}{\partial z_p}\frac{\partial z_p}{\partial o_j} \right) = \sum_p \left(\frac{\partial J}{\partial o_p}\frac{\partial o_p}{\partial z_p}\frac{\partial z_p}{\partial o_j} \right) = \sum_p \left(\frac{\partial J}{\partial o_p}\frac{\partial o_p}{\partial z_p} w_{pj} \right)$$

An Example



- Hidden layer:

$$\mathbf{W}_1 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 0.25 & 0.1 \\ 0.1 & 0.25 \end{bmatrix}; \quad \mathbf{b}_1 = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$$

  Activation function: Sigmoid.

- Output layer:

$$\mathbf{W}_0 = \begin{bmatrix} w_{o1} & w_{o2} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}; \quad b_0 = 0$$

- $y_{\mathrm{target}} = 1; \quad J = \frac{1}{2}\left(y_{\mathrm{target}} - y\right)^2$
- Neural network prediction: $y = \mathbf{W}_0(\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)) + b_0 = 0.6512$

An Example

- Neural network prediction: $y = \mathbf{W}_0(\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)) + b_0$
- Backpropagation:

$$\frac{\partial J}{\partial w_{11}} = \frac{\partial J}{\partial o_1}\frac{\partial o_1}{\partial z_1}\frac{\partial z_1}{\partial w_{11}}$$

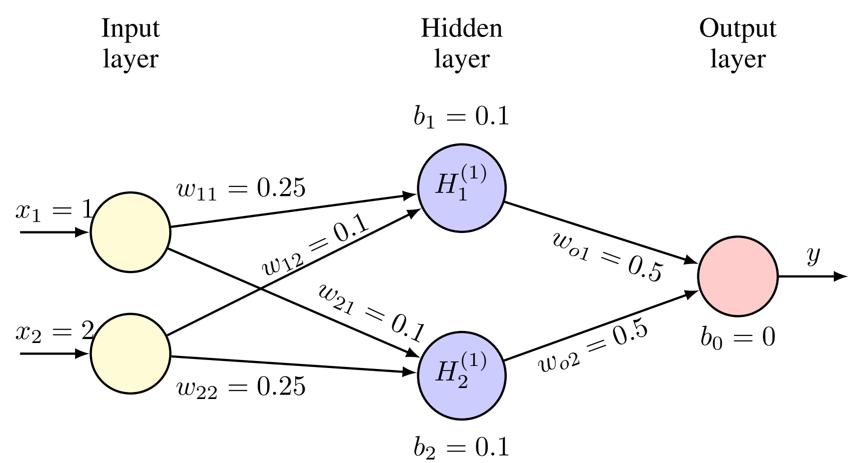
$$\frac{\partial z_1}{\partial w_{11}} = x_1 = 1$$

$$\frac{\partial o_1}{\partial z_1} = o_1(1 - o_1) = 0.2320$$

$$y = \sum_{i=1}^{2} w_{oi} o_i + b_0 = 0.6512$$

$$J = \frac{1}{2}(y_{\text{target}} - y)^2$$

$$\Rightarrow \frac{\partial J}{\partial o_1} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial o_1} = (y - y_{\text{target}})w_{o1} = -0.1744$$



$$\frac{\partial J}{\partial w_{11}} = \frac{\partial J}{\partial o_1}\frac{\partial o_1}{\partial z_1}\frac{\partial z_1}{\partial w_{11}} = -0.0405$$