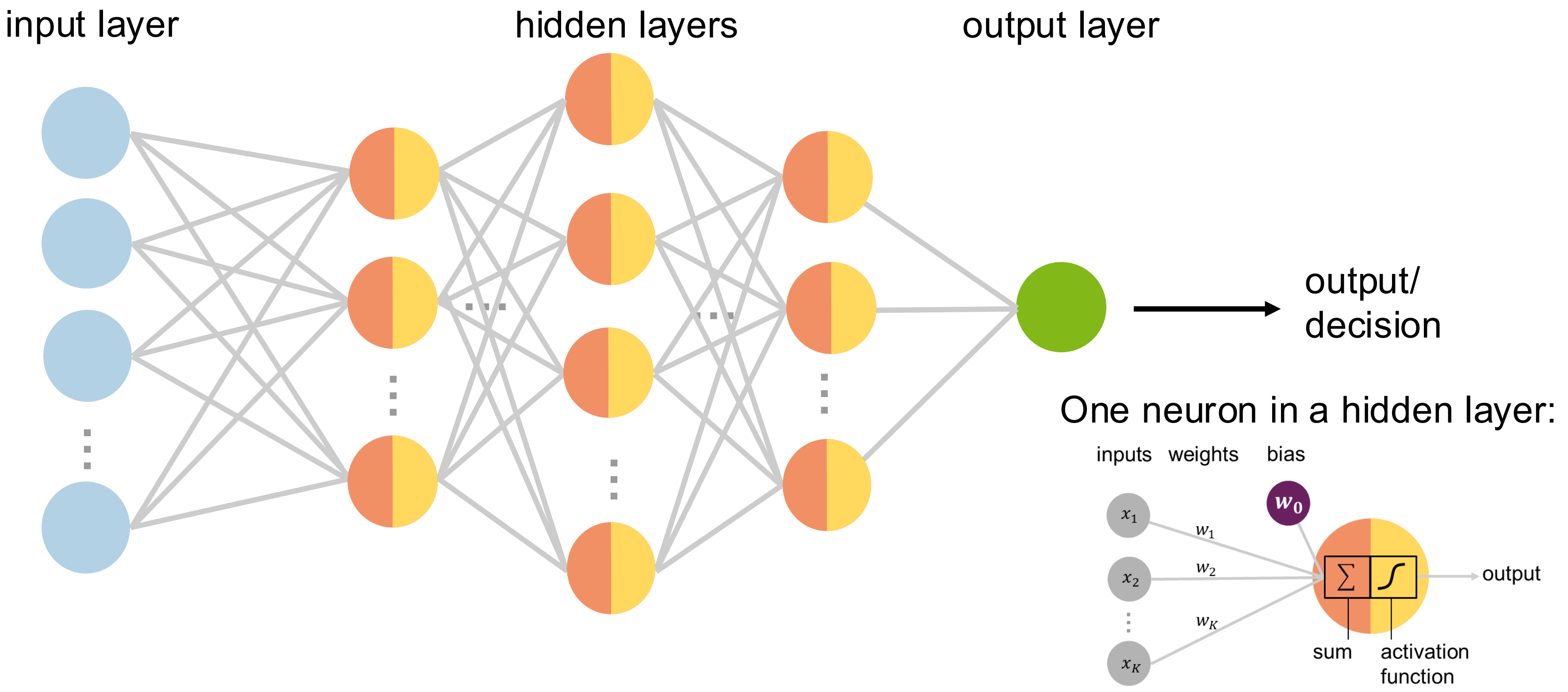
Zürcher Hochschule
für Angewandte Wissenschaften

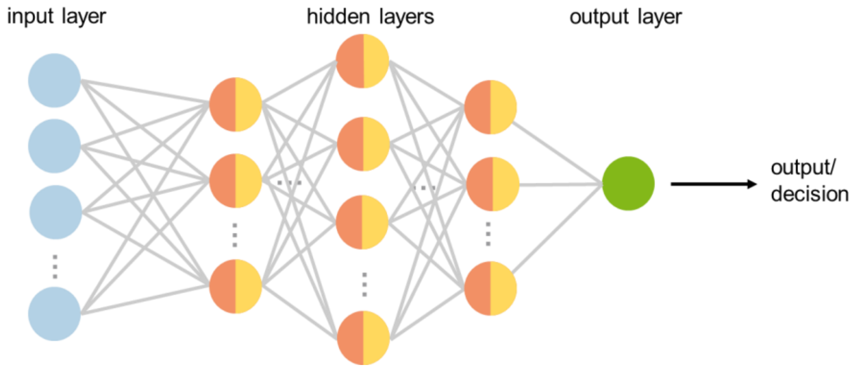
School of Engineering

Feed-forward neural networks

A Neural Network has an Input Layer, one or several Hidden Layers and an Output Layer



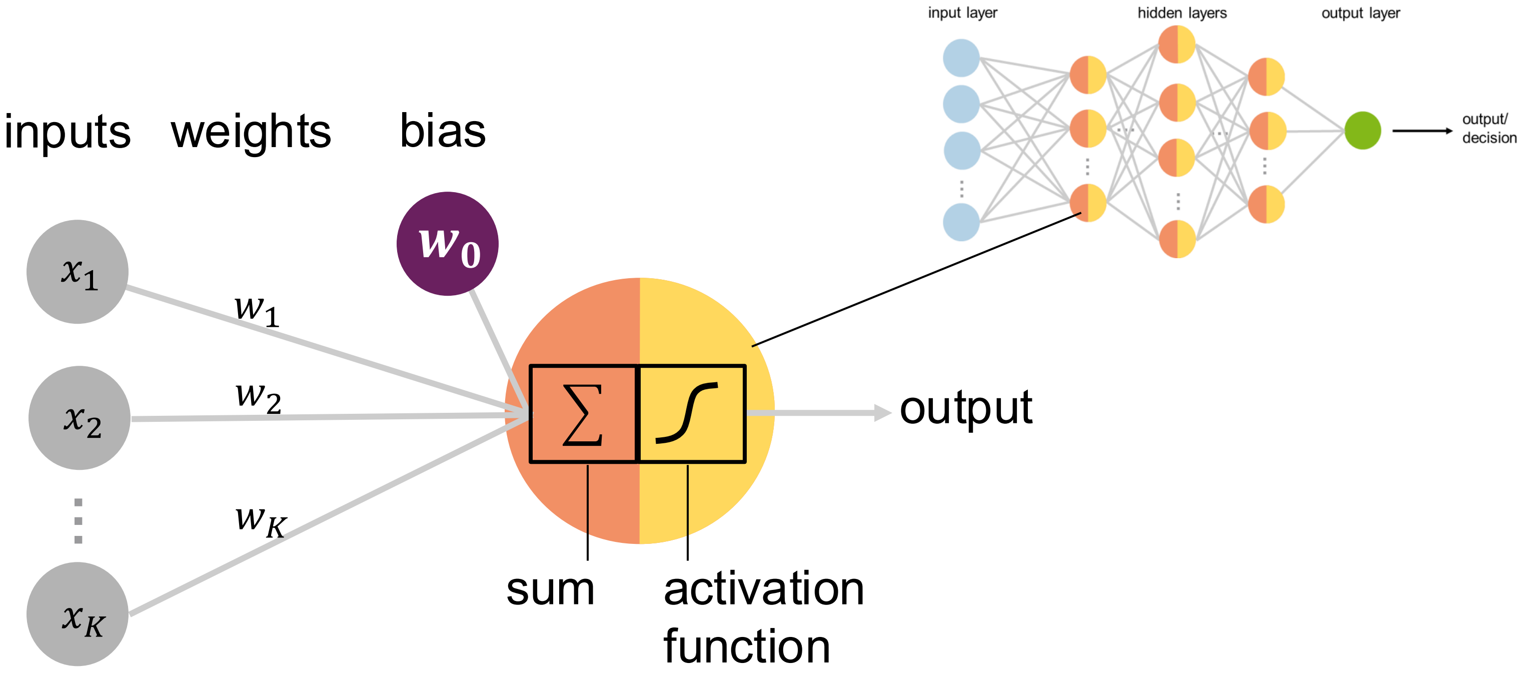
Feedforward Neural Network



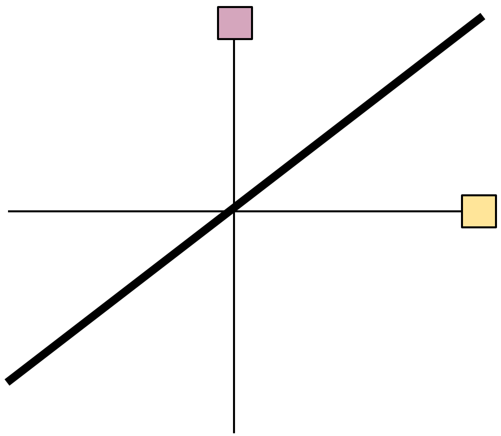
- The information moves in only **one direction:** forward from the input nodes, through the hidden nodes (if any) to the output nodes.
- Each neuron in one layer has directed connections to the neurons of the subsequent layer
- **No connections** between nodes within the **same layer**
- There are no cycles or loops in the network
- Vanilla Neural Network Structure: Subsequent layers are **fully connected**
- **Trainable Parameters:** Weights connecting the nodes and biases

A neuron

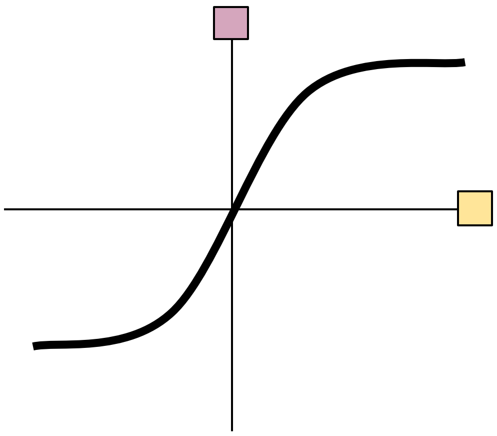
… applies an activation function to the weighted sum of its inputs



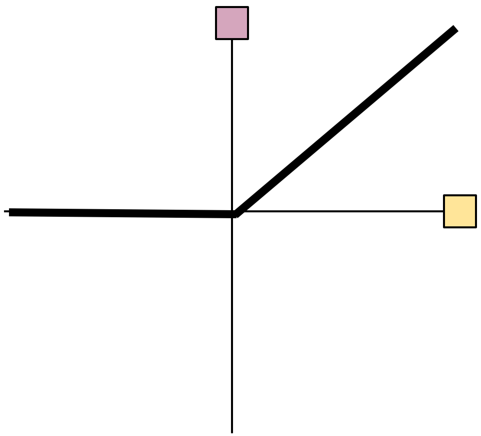
Activation functions



LINEAR
like linear regression
(only used in output
layer)

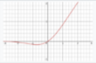
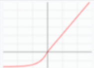
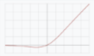
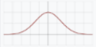


LOGISTIC /
SIGMOIDAL / TANH
Smooth, differentiable,
saturating functions



RECTIFIED
LINEAR (ReLU)
Cheap to
compute, popular
lately

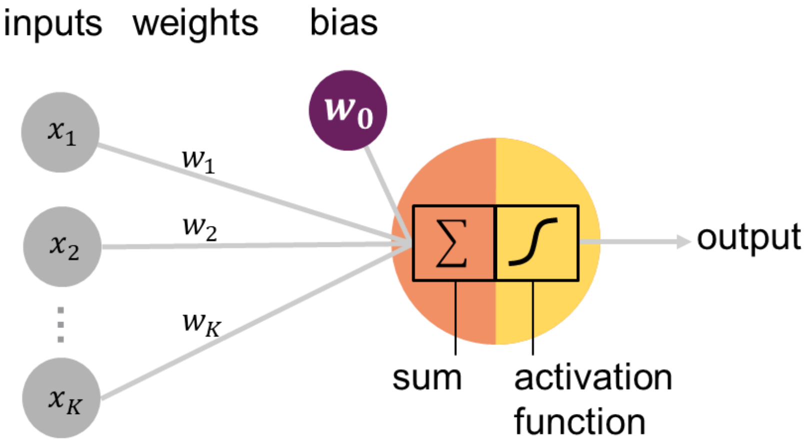
Activation functions

Name	Plot	Function, $g(x)$	Derivative of g, $g'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty, \infty)$	C^∞
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$	$g(x)(1-g(x))$	$(0, 1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - g(x)^2$	$(-1, 1)$	C^∞
Rectified linear unit (ReLU)[8]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$	C^0
Gaussian Error Linear Unit (GELU)[5]		$\frac{1}{2}x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = x\Phi(x)$	$\Phi(x) + x\phi(x)$	$(-0.17\ldots, \infty)$	C^∞
Softplus[9]		$\ln(1 + e^x)$	$\frac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞
Exponential linear unit (ELU)[10]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C^1 & \text{if } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$
Scaled exponential linear unit (SELU)[11]		$\lambda\begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$	$\lambda\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\lambda\alpha, \infty)$	C^0
Leaky rectified linear unit (Leaky ReLU)[12]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$(-\infty, \infty)$	C^0
Parametric rectified linear unit (PReLU)[13]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α	$\begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
Sigmoid linear unit (SiLU,[5] Sigmoid shrinkage,[14] SiL,[15] or Swish-1[16])		$\frac{x}{1+e^{-x}}$	$\frac{1 + e^{-x} + xe^{-x}}{(1+e^{-x})^2}$	$[-0.278\ldots, \infty)$	C^∞
Gaussian		e^{-x^2}	$-2xe^{-x^2}$	$(0, 1]$	C^∞

Source: https://en.wikipedia.org/wiki/Activation_function

Neuron with ReLu activation

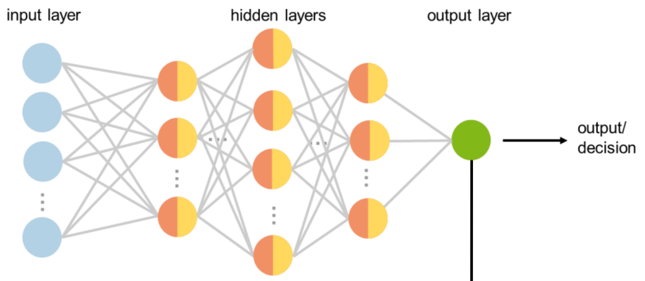
… applies an activation function to the weighted sum of its inputs



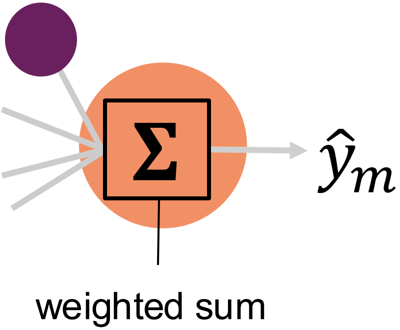
sum: $z = w_0 + w_1 x_1 + w_2 x_2 + \ \dots + w_K x_K$

output: $o = \text{act(z)} = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$

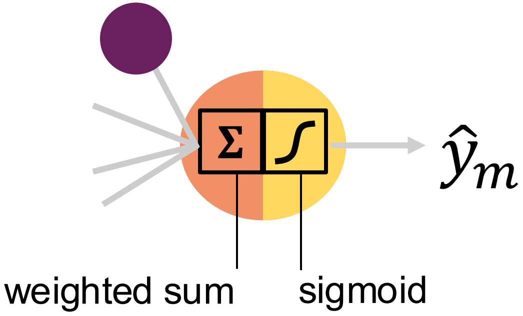
The Output Layer



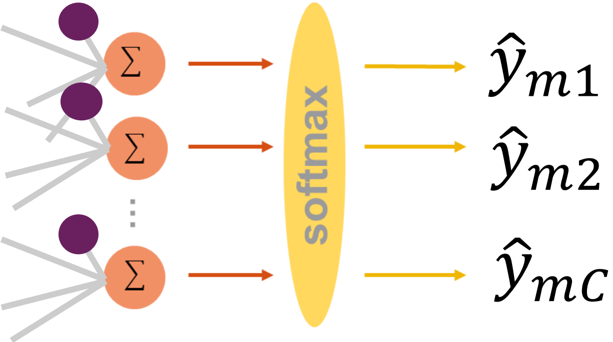
Regression: Linear Function



Binary Classification: Sigmoid



Multinomial Classification: Softmax



The output layer

Zürcher Hochschule
für Angewandte Wissenschaften

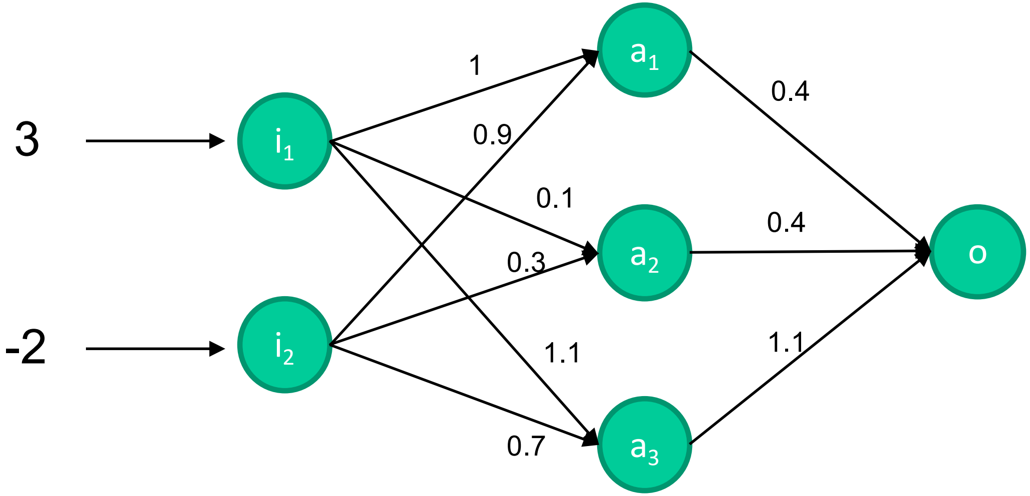
zh aw School of Engineering

Regression tasks:
- output is a single node, the predicted value

Classification with two classes:
- output: a single value between 0..1
- decision boundary e.g. at 0.5

Classification with many classes:
- number of output values = number of classes
- sum over all output values = 1
- each output value between 0..1
- decision: largest output value is predicted class

Example



ReLU Activation function for all hidden nodes:

$$\mathrm{act}(z_{a_i}) = \mathrm{ReLu}(z_{a_i}) = \begin{cases} z_{a_i} & \text{if } z_{a_i} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

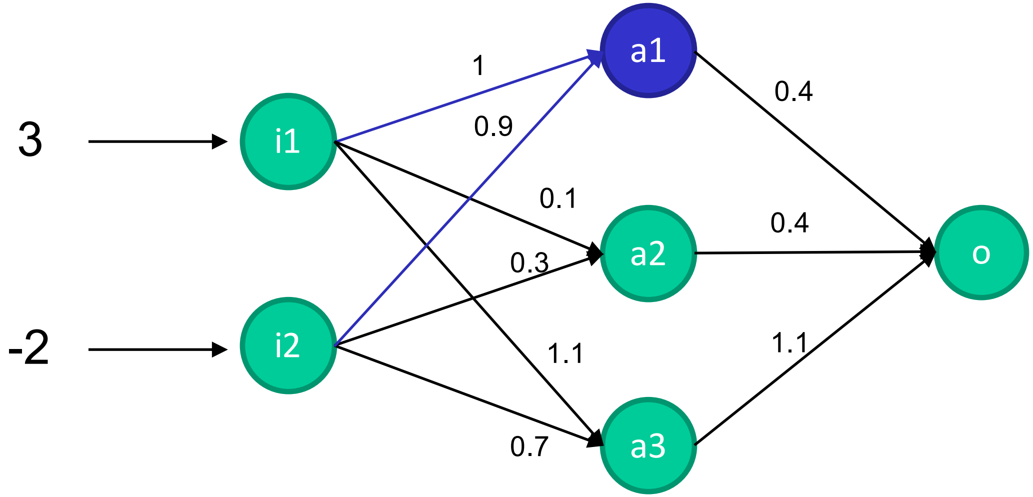
Sigmoid for output node

$$\mathrm{act}(z_o) = \sigma(z_o) = \frac{1}{1 + e^{-(z_o)}}$$

The bias (w_0) is not explicitly depicted. It is the same for all nodes:

$$\mathrm{w}_{\mathrm{a}_1,0} = \mathrm{w}_{\mathrm{a}_2,0} = \mathrm{w}_{\mathrm{a}_3,0} = w_{\mathrm{o},0} = 0.2$$

Example

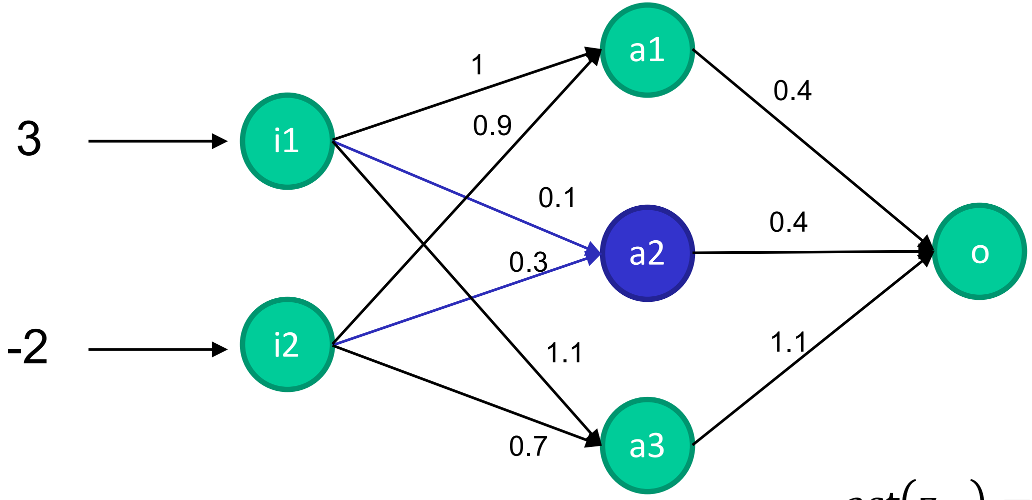


$$\mathrm{act}(z_{a_i}) = \mathrm{ReLu}(z_{a_i}) = \begin{cases} z_{a_i} & \text{if } z_{a_i} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The bias (w_0) for all nodes is the same: $\mathrm{w}_{a_1,0} = \mathrm{w}_{a_2,0} = \mathrm{w}_{a_3,0} = w_{o,0} = 0.2$

$$z_{a_1} = i_1 \cdot 1 + i_2 \cdot 0.9 + \mathrm{w}_{a1,0} = 3 \cdot 1 - 2 \cdot 0.9 + 0.2 = 1.4$$
$$a_1 = \mathrm{act}(z_{a_1}) = \mathrm{ReLu}(1.4) = 1.4$$

Example



$$\text{act}(z_{a_i}) = \text{ReLu}(z_{a_i}) \begin{cases} z_{a_i} & \text{if } z_{a_i} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

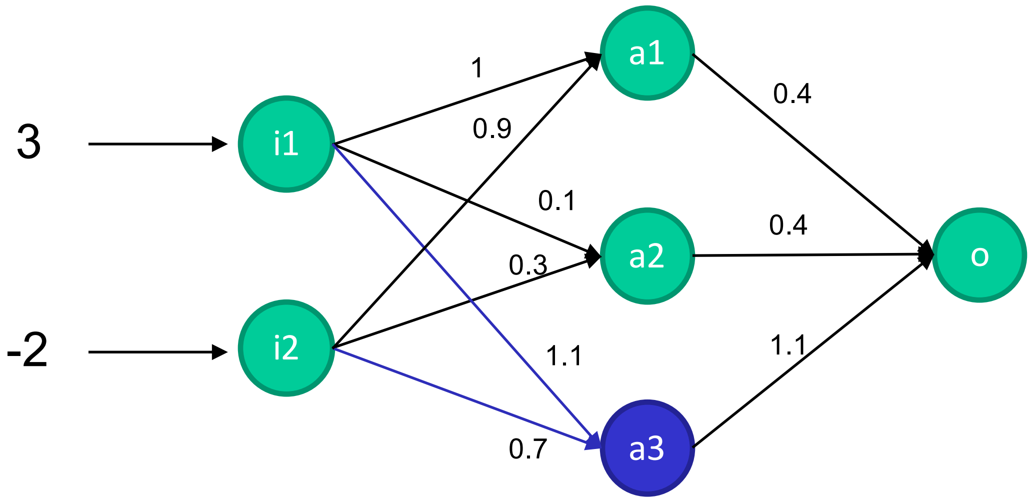
The bias (w_0) for all nodes is the same: $\mathrm{w}_{a_1,0} = \mathrm{w}_{a_2,0} = \mathrm{w}_{a_3,0} = w_{o,0} = 0.2$

$$z_{a_2} = i_1 \cdot 0.1 + i_2 \cdot 0.3 + w_{a2,0} = 3 \cdot 0.1 - 2 \cdot 0.3 + 0.2 = -0.1$$

$$a_2 = \text{act}(z_{a_2}) = \text{ReLu}(-0.1) = 0$$

Example

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering



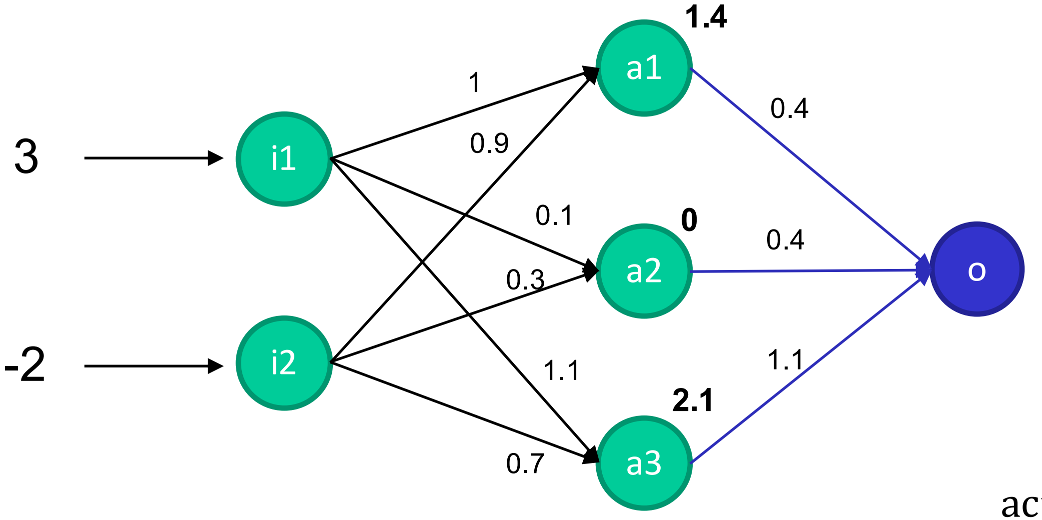
$$\mathrm{act}(z_{a_i}) = \mathrm{ReLu}(z_{a_i}) = \begin{cases} z_{a_i} & \text{if } z_{a_i} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The bias (w_0) for all nodes is the same: $\mathrm{w}_{\mathrm{a}_1,0} = \mathrm{w}_{\mathrm{a}_2,0} = \mathrm{w}_{\mathrm{a}_3,0} = w_{\mathrm{o},0} = 0.2$

$$z_{a_3} = i_1 \cdot 1.1 + i_2 \cdot 0.7 + w_{\mathrm{a}3,0} = 3 \cdot 1.1 - 2 \cdot 0.7 + 0.2 = 2.1$$

$$a_3 = \mathrm{act}(z_{a_3}) = \mathrm{ReLu}(2.1) = 2.1$$

Example



$$\mathrm{act}(z_o) = \sigma(z_o) = \frac{1}{1 + e^{-(z_o)}}$$

The bias (w_0) for all nodes is the same: $\mathrm{w}_{\mathrm{a}_1,0} = \mathrm{w}_{\mathrm{a}_2,0} = \mathrm{w}_{\mathrm{a}_3,0} = w_{\mathrm{o},0} = 0.2$

$$z_o = a_1 \cdot 0.4 + a_2 \cdot 0.4 + a_3 \cdot 2.1 + w_{\mathrm{o},0} = 1.4 \cdot 0.4 + 0 \cdot 0.4 + 2.1 \cdot 1.1 + 0.2 = 3.07$$

$$\hat{y} = \mathrm{act}(z_o) = \frac{1}{1 + e^{-(z_o)}} = 0.96$$

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

Training Neural Networks

The cost function for neural networs

Zürcher Hochschule für Angewandte Wissenschaften

zh aw School of Engineering

Regression: Average sum of squared residuals

$$J(\boldsymbol{W}) = \frac{1}{2M}\sum_{m=1}^{M}(y_m - \hat{y}_m)^2$$

Classification: Average Cross Entropy

- Binary (logistic):

$$\mathrm{J}(\boldsymbol{W}) = -\frac{1}{M}\sum_{m=1}^{M} y_m \log \hat{y}_m + (1 - y_m)\log(1 - \hat{y}_m)$$

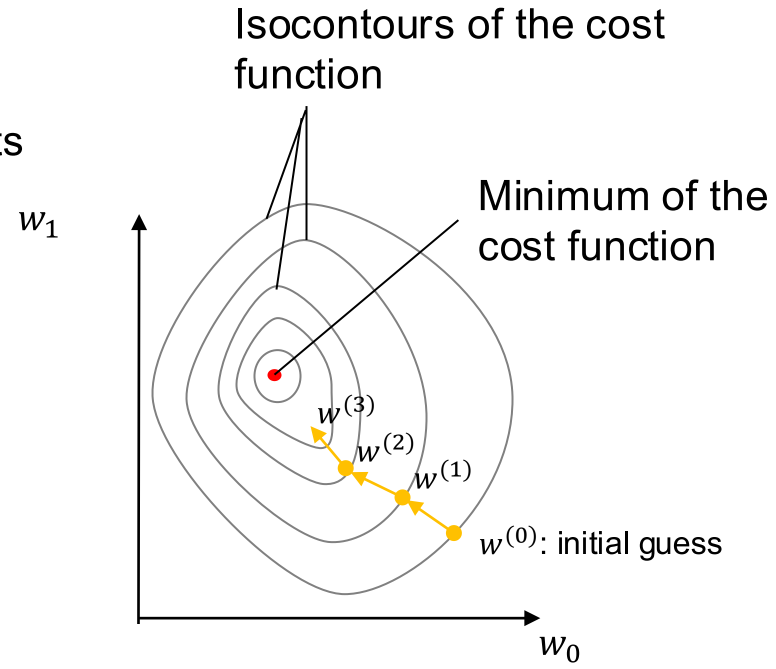
- Multinomial, i.e. multiclass (softmax):

$$\mathrm{J}(\boldsymbol{W}) = -\frac{1}{M}\sum_{m=1}^{M} y_{mk} \log \hat{y}_{mk} \text{ (where } k \text{ is the correct class index for sample } m\text{)}$$

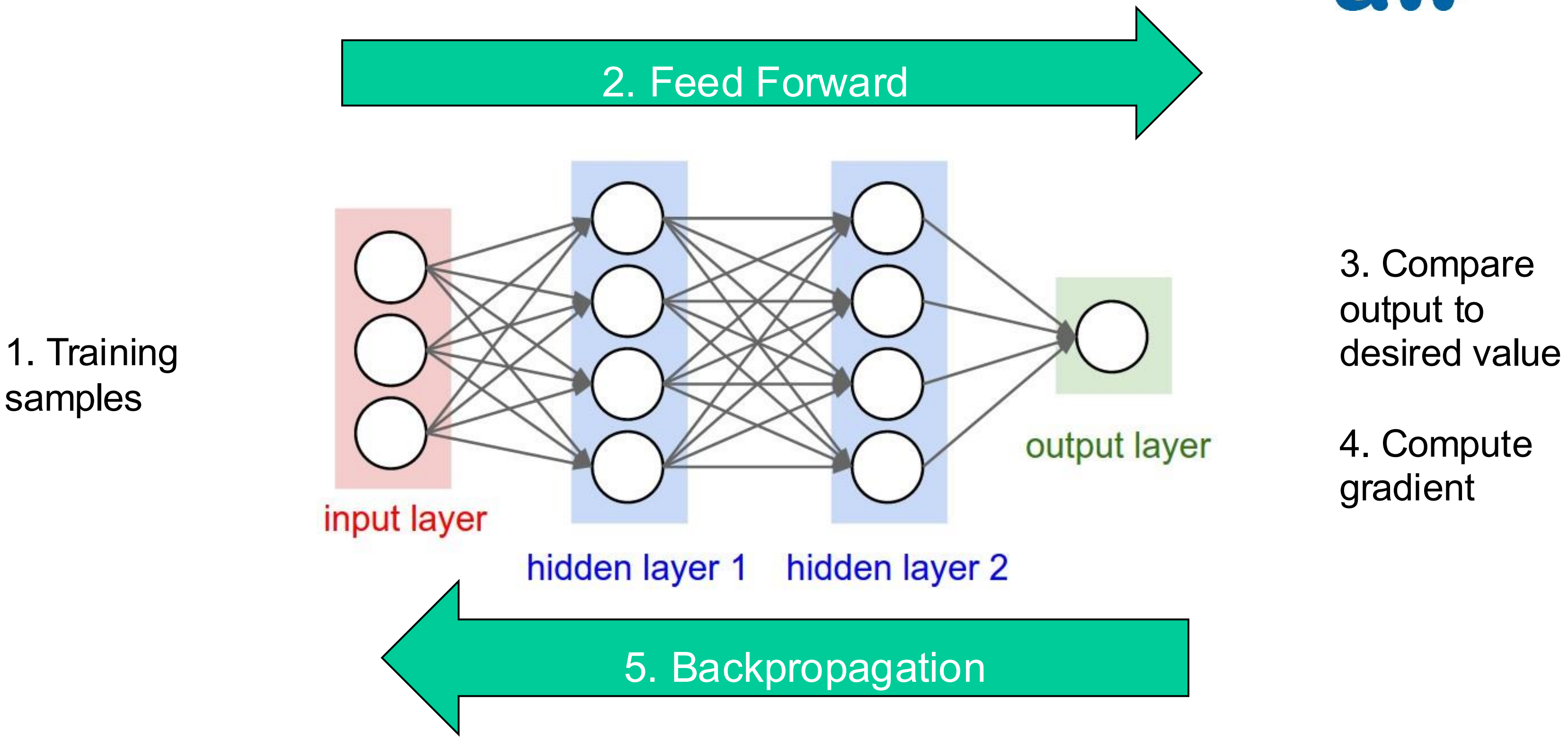
Minimisation by gradient descent

- Initial guess for the model parameters: the weights
- Repeat until stopping criterium is reached:
    1. compute **gradient** of the cost function with respect to parameters
    2. **adjust the parameters** in the opposite direction of the gradient by a small step, i.e. the **learning rate** α

$$w_{kl} \leftarrow w_{kl} - \alpha \frac{\partial J(\boldsymbol{W})}{\partial w_{kl}}$$

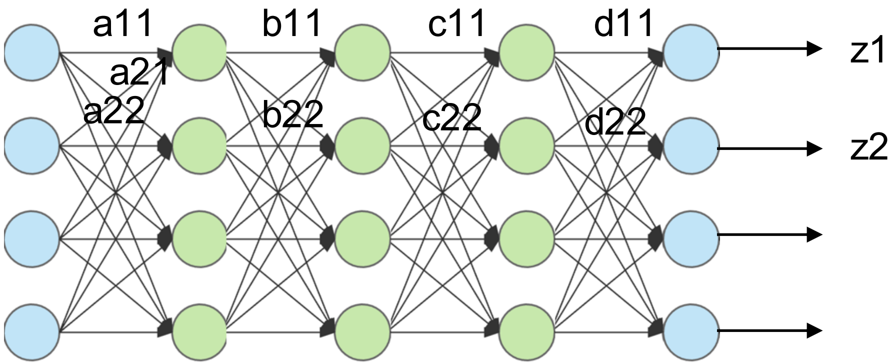


Backpropagation: efficient computation of the partial derivatives in gradient descent



https://www.pyimagesearch.com/wp-content/uploads/2016/08/simple_neural_network_header.jpg

Re-use of Partial Derivative Chains



$$\frac{\partial z1}{\partial a11} = \frac{\partial z1}{\partial d11} \frac{\partial d11}{\partial c11} \frac{\partial c11}{\partial b11} \frac{\partial b11}{\partial a11} + \dots$$

$$\frac{\partial z1}{\partial a21} = \dots + \frac{\partial z1}{\partial d11} \frac{\partial d11}{\partial c11} \frac{\partial c11}{\partial b11} \frac{\partial b11}{\partial a21} \dots$$

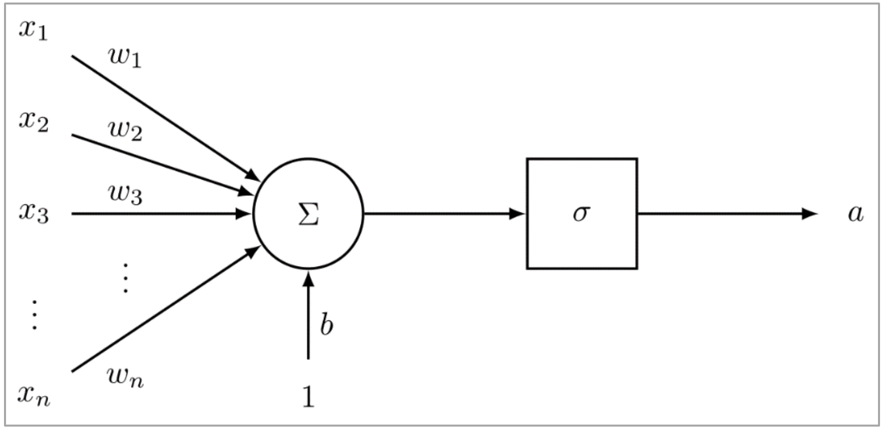
$$\frac{\partial z2}{\partial a11} = \dots + \frac{\partial z22}{\partial d12} \frac{\partial d12}{\partial c11} \frac{\partial c11}{\partial b11} \frac{\partial b11}{\partial a11} \dots$$

In each path for the partial derivatives, we can re-use sub-paths from previous calculations. This makes backpropagation efficient.

History:

- First described by Werbos 1974
- Re-discovered by Parker 1982 and **Rumelhart et al. 1986**

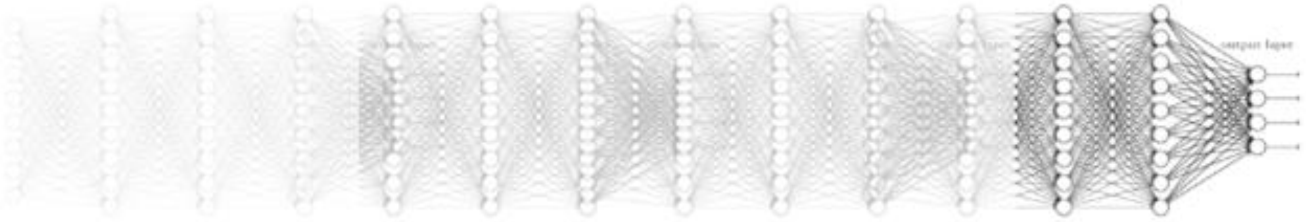
Key factor: Differentiability



If all components in a neural network are differentiable, we can efficiently compute the gradient of the cost function with respect to the weights and optimise using gradient descent

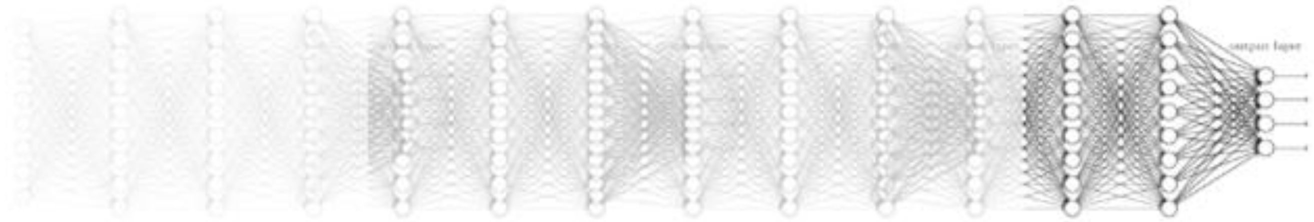
Disadvantages of Backpropagation

- Requires large amount of labelled training data
- Learning time is slow for multiple hidden layers
- Can get stuck in local optima (although not a big problem in high dimensions)
- **Vanishing Gradient Problem**



Vanishing Gradient Problem
Sepp Hochreiter 1991

- **Observation:** if we have many hidden layers, and if all partial derivatives are between -1 and 1 (sigmoid function), then multiplying them makes them exponentially small
- Thus, changes in the first layers will diminish since the gradient becomes extremely small
- Depths of a network can be easily up to 150 layers
- In practice, this results in very slowly changing weights, thus, very long training times
- Less of problem with ReLu (question: why?)
- Advanced network architectures avoid this problem



Zürcher Hochschule
für Angewandte Wissenschaften

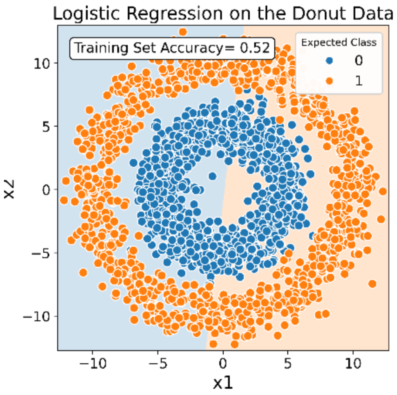
School of Engineering

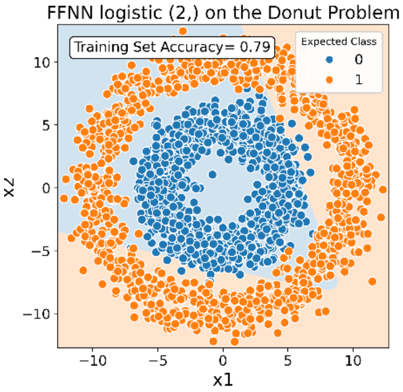
Neural networks for non-linear problems

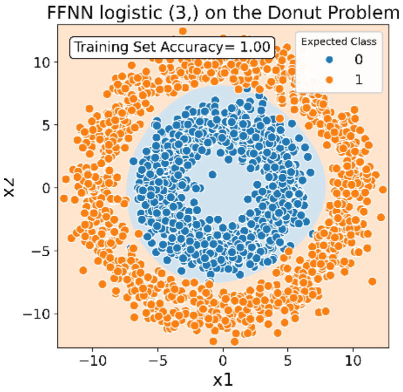
Non-linear decision boundaries with neural networks

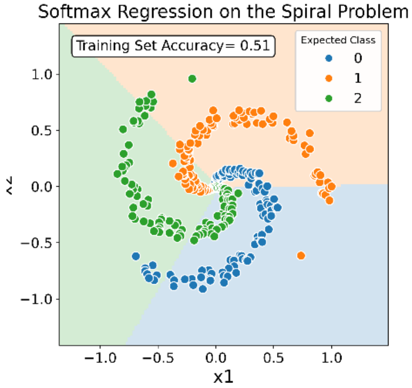
Zürcher Hochschule für Angewandte Wissenschaften

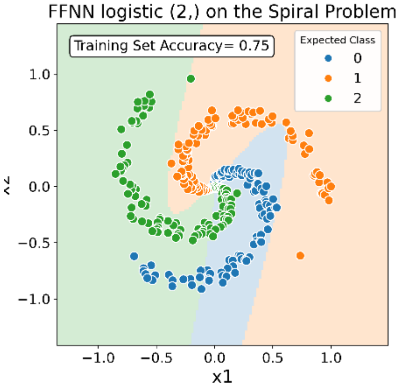
School of Engineering

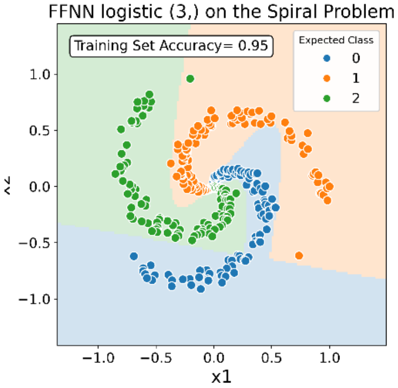






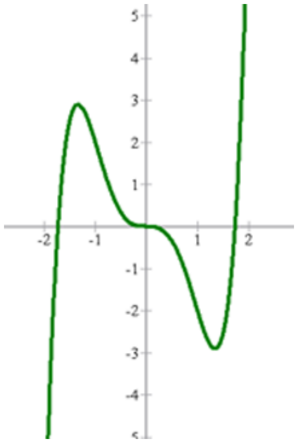


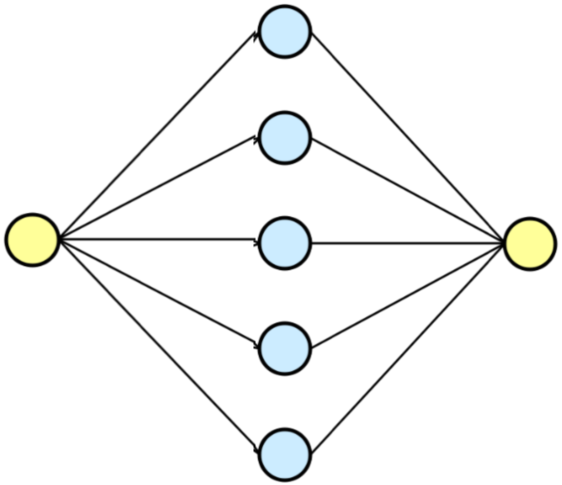




Universality Theorem
Hornik 1991

A neural net with one hidden layer and arbitrary number of neurons can approximate any given continuous function.

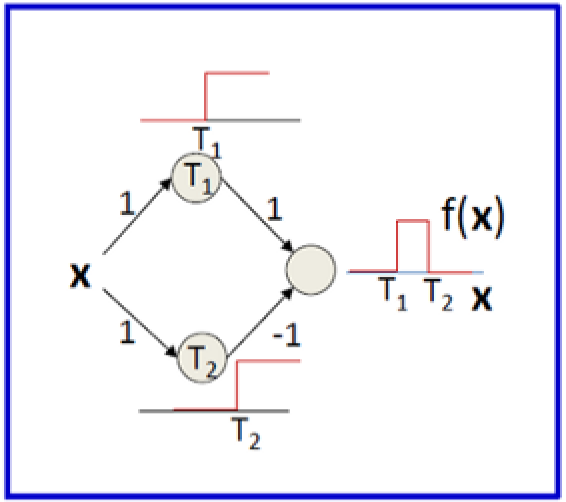


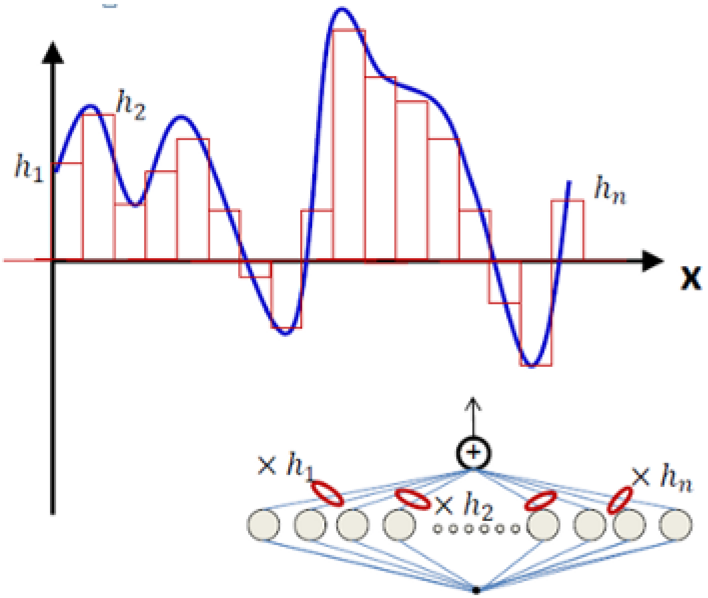


Proof Idea of Universality Theorem

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering

Idea: Cut a given function *f* in sufficient amount of small pieces, then use 2 neurons that model a partial linear function to approximate each of these pieces





https://medium.com/analytics-vidhya/neural-networks-and-the-universal-approximation-theorem