Artificial Neural Networks

Kate Farrahi

Some Additional Online References

Neural Networks (+ Deep Learning)

- Michael Nielson's online book http://neuralnetworksanddeeplearning.com/chap1.html

- Deep Learning by Ian Goodfellow, Y. Bengio, and A. Courville

Backpropagation:

Step by Step example by Matt Mazur: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Activation functions:

- http://cs231n.github.io/neural-networks-1/#actfun

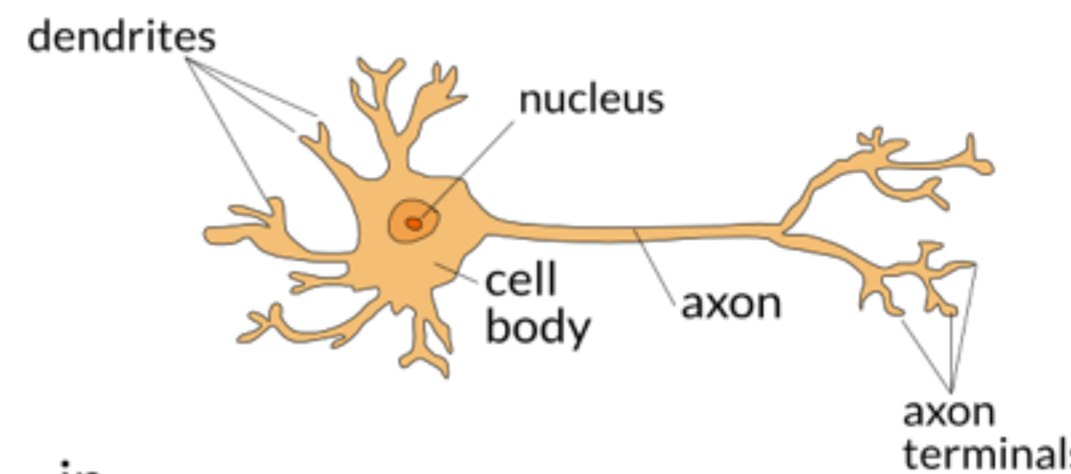
The Human Brain

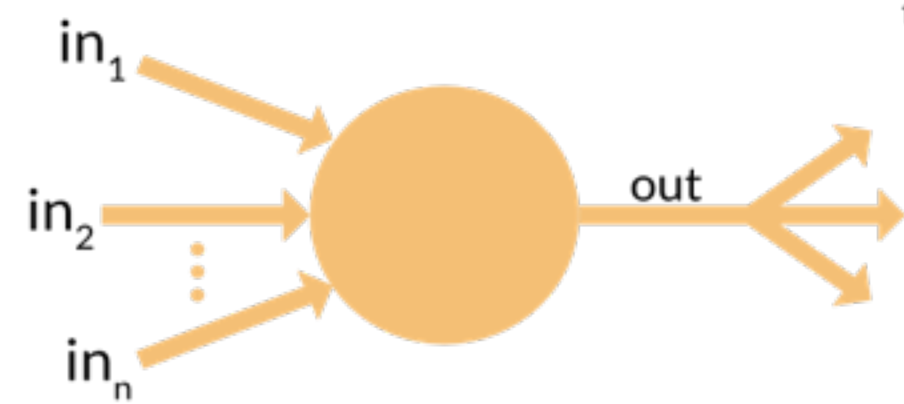
- Highly complex, non-linear, and parallel "computer"
- Structural constituents: neurons
- The structure of the brain is extremely complex and not fully understood
- Billions of nerve cells (neurons) and trillions of interconnections in the human brain
- Scientists tried to mimic the brain's behaviour in proposing the artificial neural network (ANN)
- The human brain is the inspiration for ANNs though we cannot say ANNs actually replicate the brain's behaviour very well, they are extremely simplified

The Neuron

- The basic ingredient of any ANN is the artificial neuron
- They are named and modelled after their biological counterparts - the neurons in the human brain
- input from the dendrites
- output from the axons

biological neuron





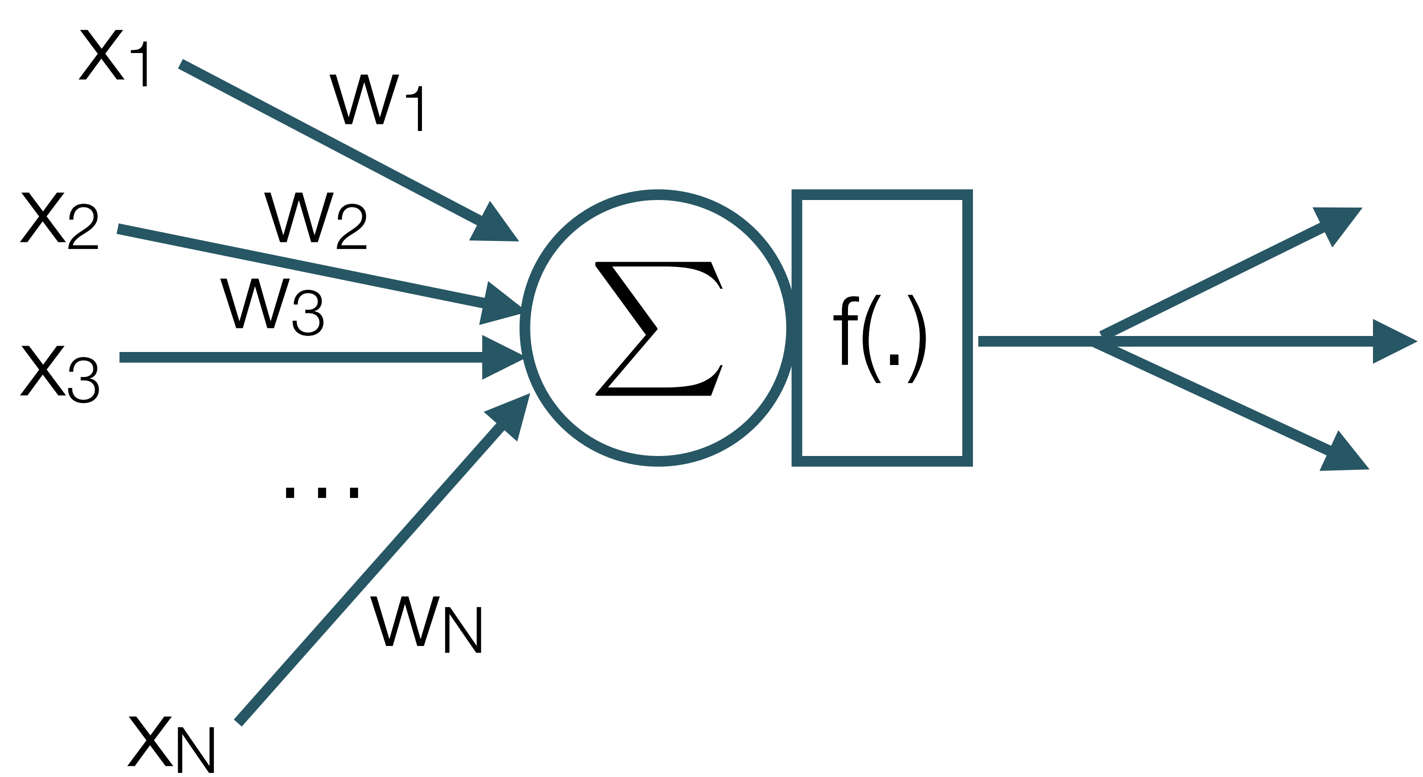
artificial neuron

The Artificial Neuron

- The basic unit that builds up ANNs has N inputs that are connected to the neuron with varying strengths of connection that we represent as weights, $w_1 \dots w_N$
- The signal that goes into the neuron is

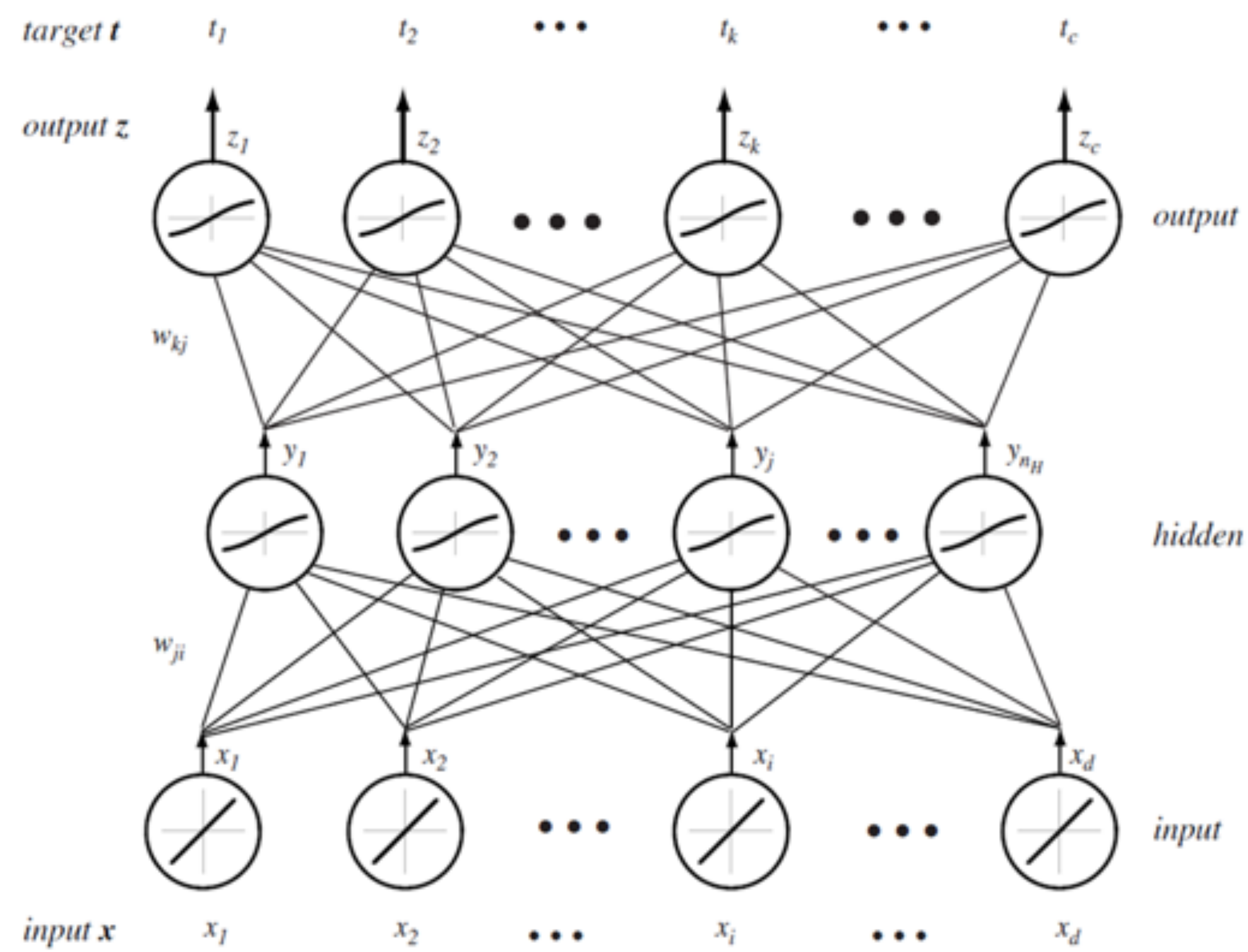
$$x_1 \cdot w_1 + x_2 \cdot w_2 + ...x_N \cdot w_N$$

- We want a decision out of the neuron (e.g. 1 or 0), therefore there must be a non-linear unit after the summation before outputting from the neuron



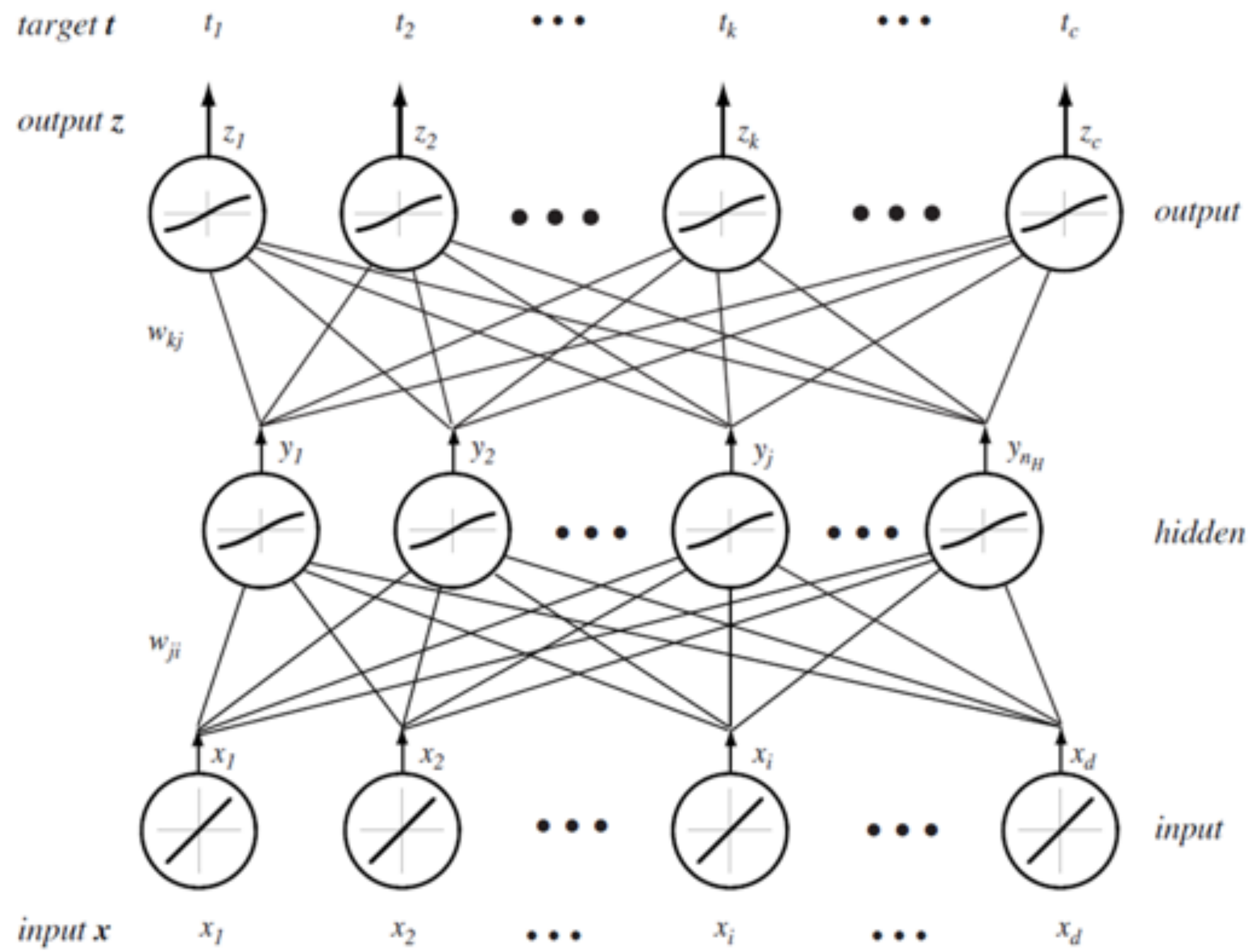
The Multilayer Perceptron - Architecture

- MLP is a class of *feedforward* artificial neural networks (ANNs)
- MLPs are fully connected
- MLPs consist of three or more layers of nodes
- 1 input layer, 1 output layer, 1 or more hidden layers
- d-n_H-c fully connected three-layer network



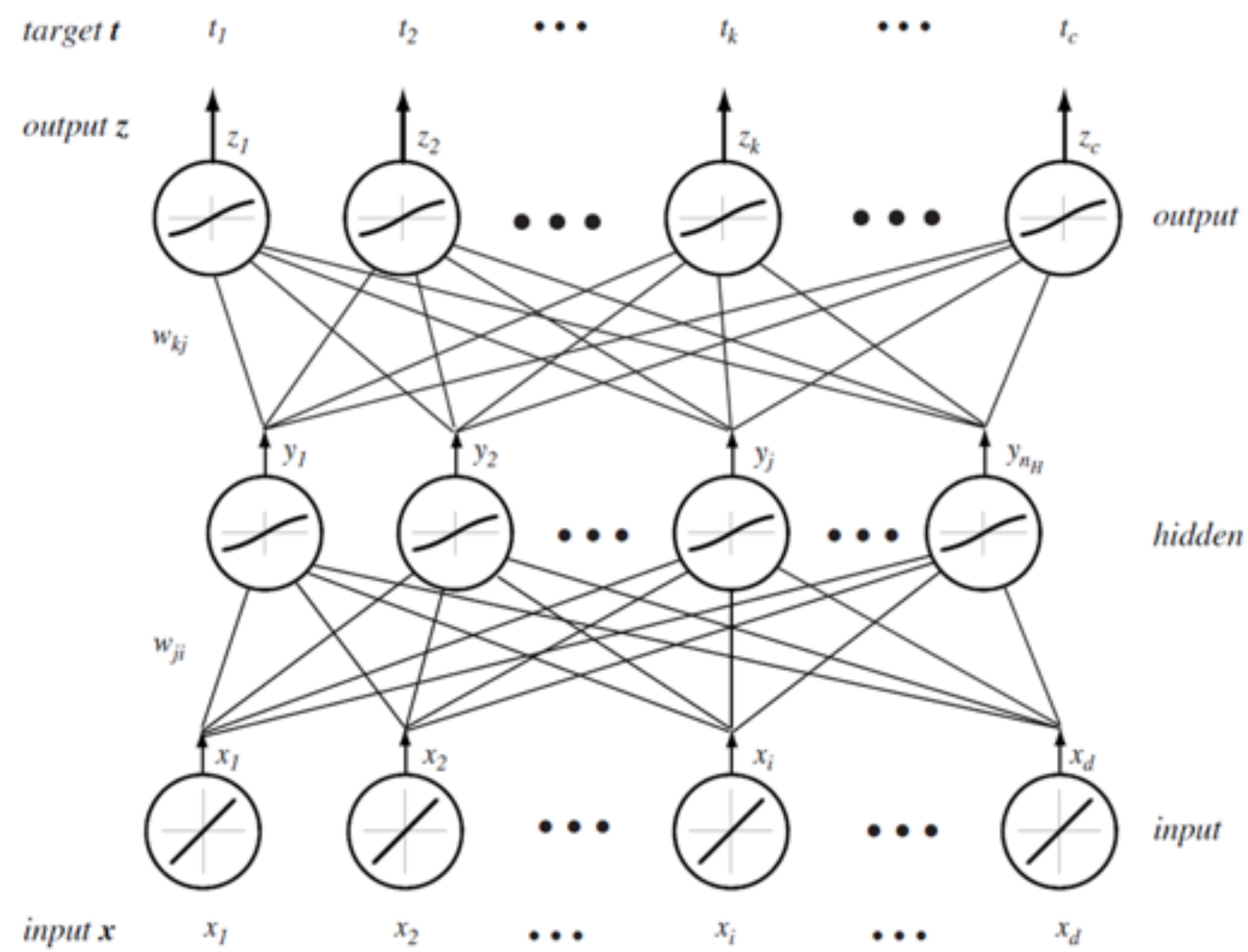
The Multilayer Perceptron - Input Layer

- d-dimensional input **x**
- no neurons at the input layer - "input units"
- each input unit simply emits the input x_i



The Multilayer Perceptron - Hidden Layer

- n_H neurons in the hidden layer
- each neuron in the hidden layer uses a non-linear activation function
- Weight w_{ji} denotes the input-to-hidden layer weights at the hidden unit j

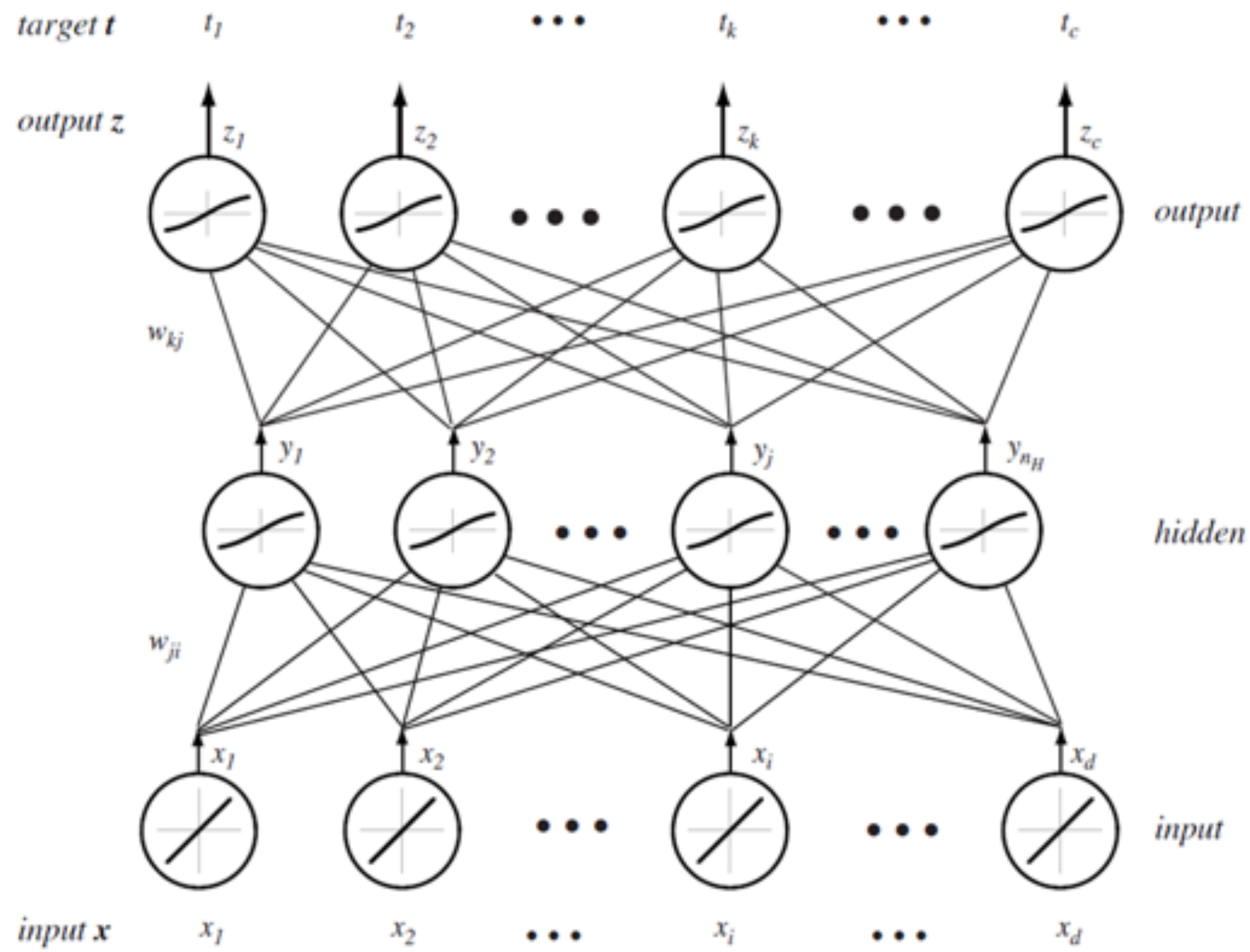


The Multilayer Perceptron - Output Layer

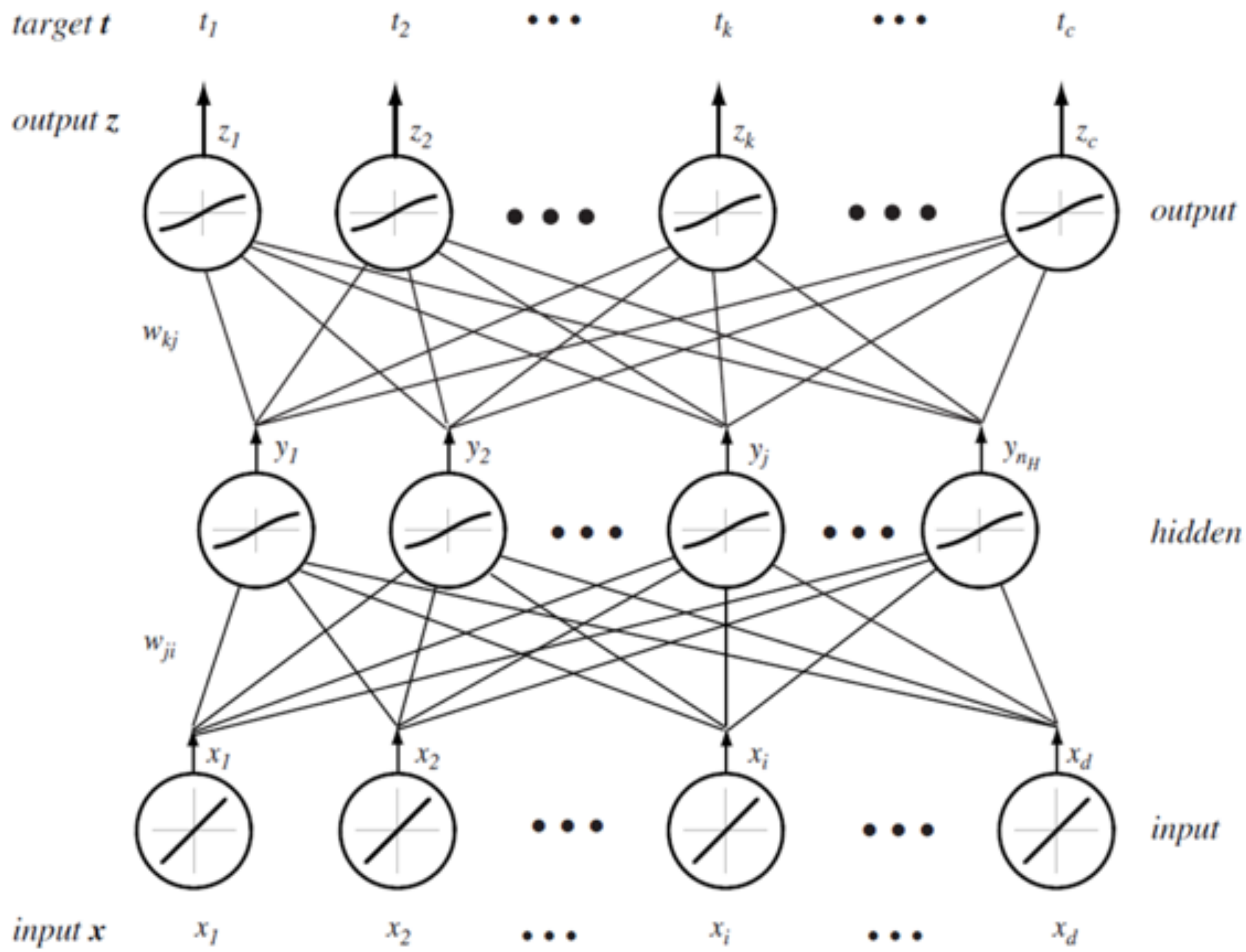
- c neurons in the output layer

- each neuron in the output layer also uses a non-linear activation function

- c and the activation function at the output layer related to the problem you are trying to solve - more details later

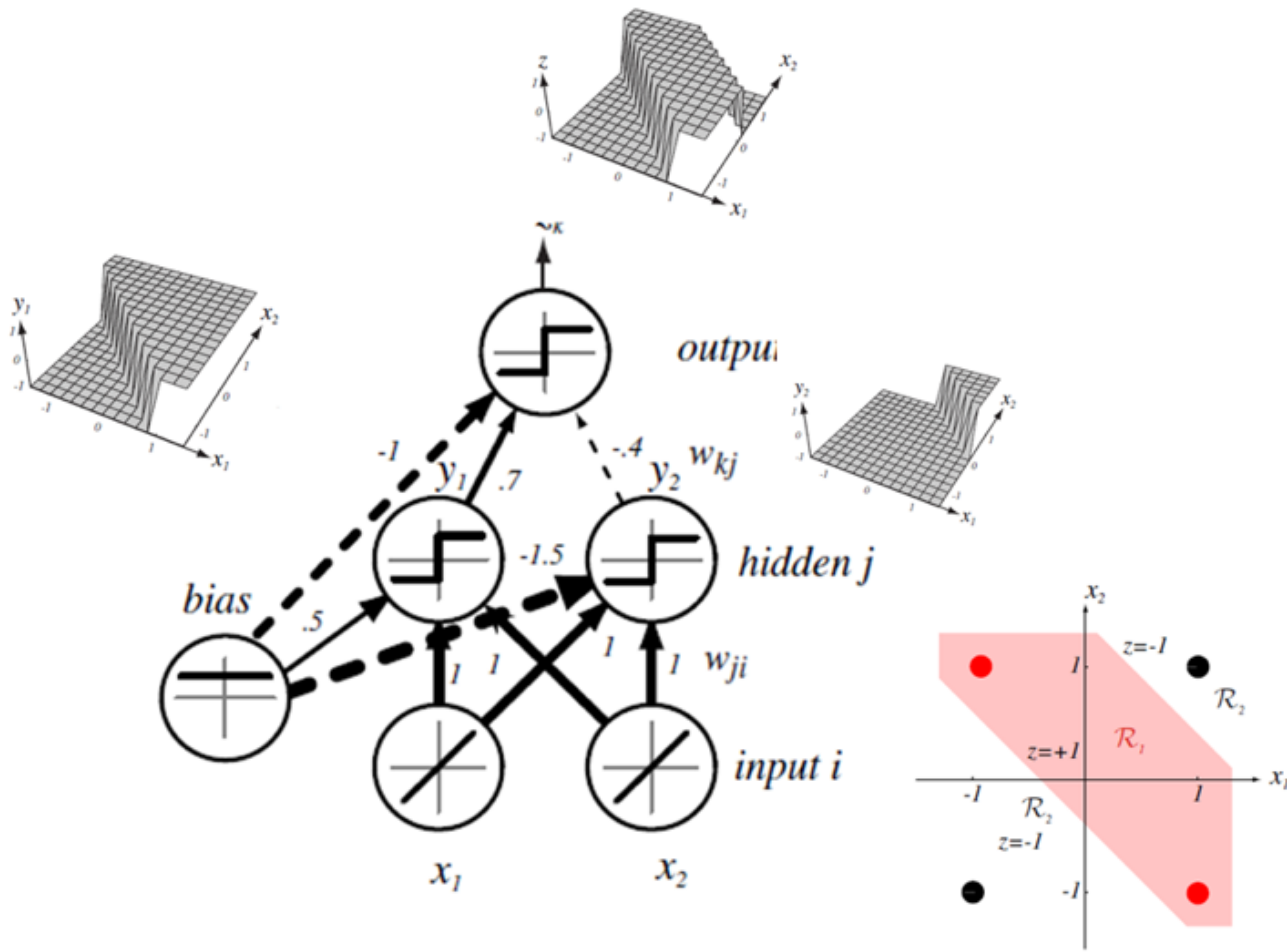


The Multilayer Perceptron - Output Layer



$$g_k(\mathbf{x}) = f(\sum_{j=1}^{n_H} w_{jk} f(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}) + w_{k0})$$

XOR Using a MLP



XOR using MLP with sign function as activation

- Weighted sum of inputs - net activation

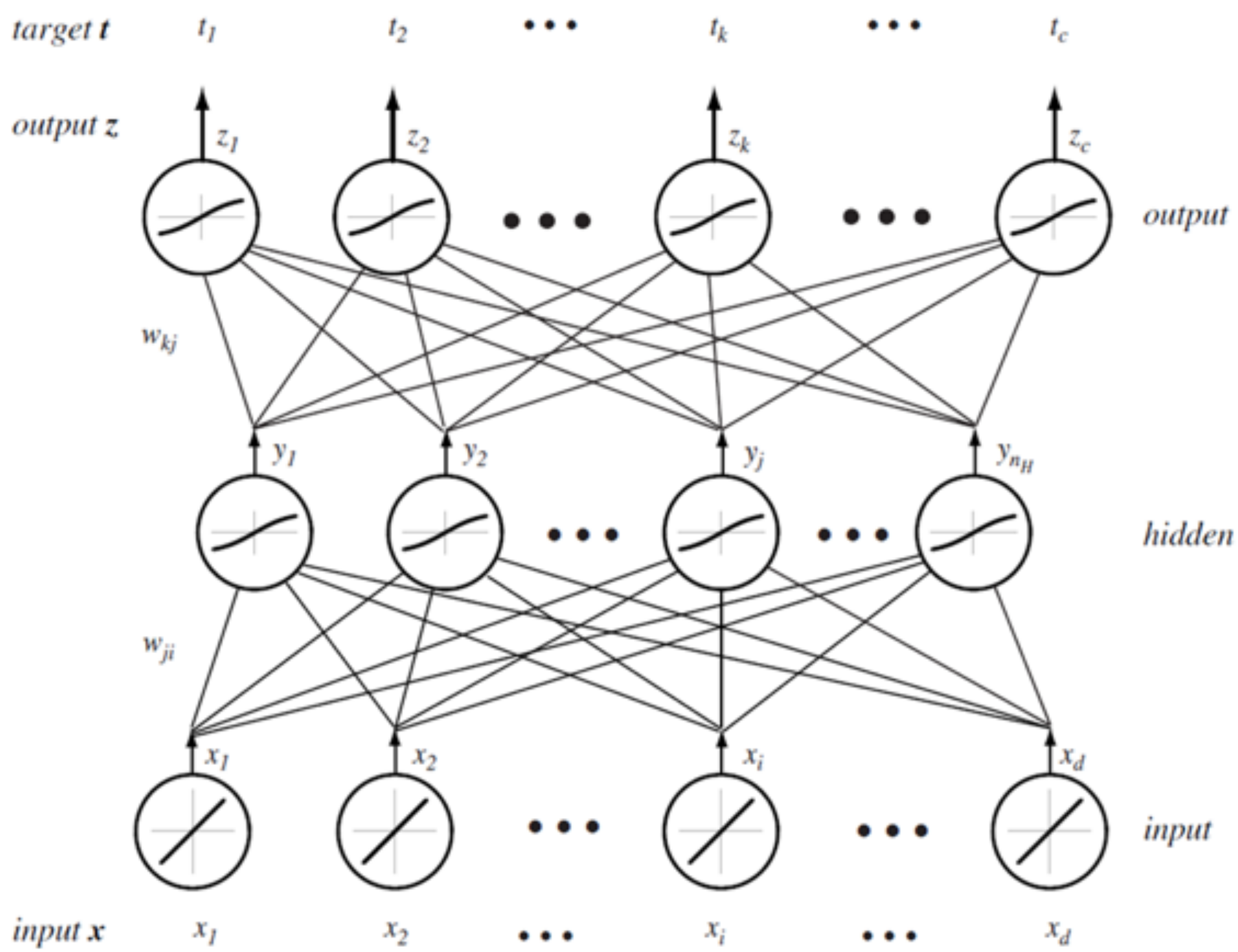
$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0}$$

- index i indexes the units of the input layer

- index j indexes the units in the hidden layer

- Squashed by a nonlinearity $y_j = f(net_j)$

The Multilayer Perceptron

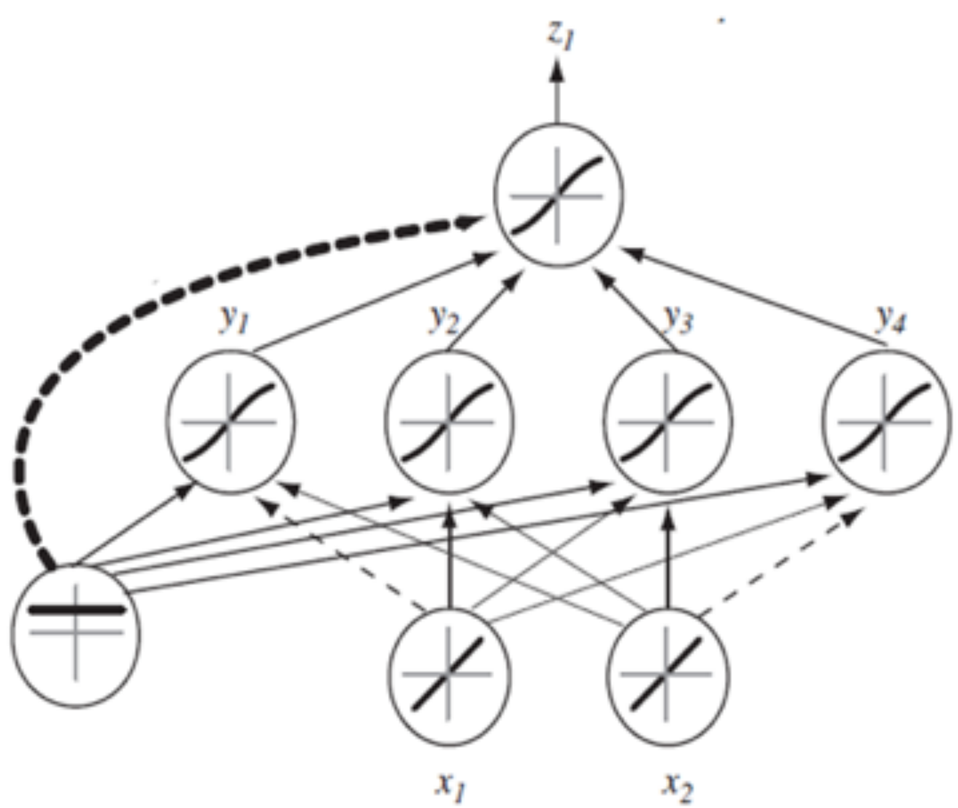


$$g_k(\mathbf{x}) = f(\sum_{j=1}^{n_H} w_{jk} f(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}) + w_{k0})$$

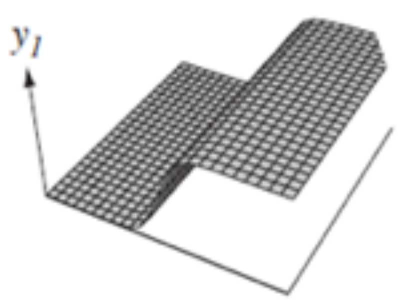
Expressive Power of Multilayer Networks

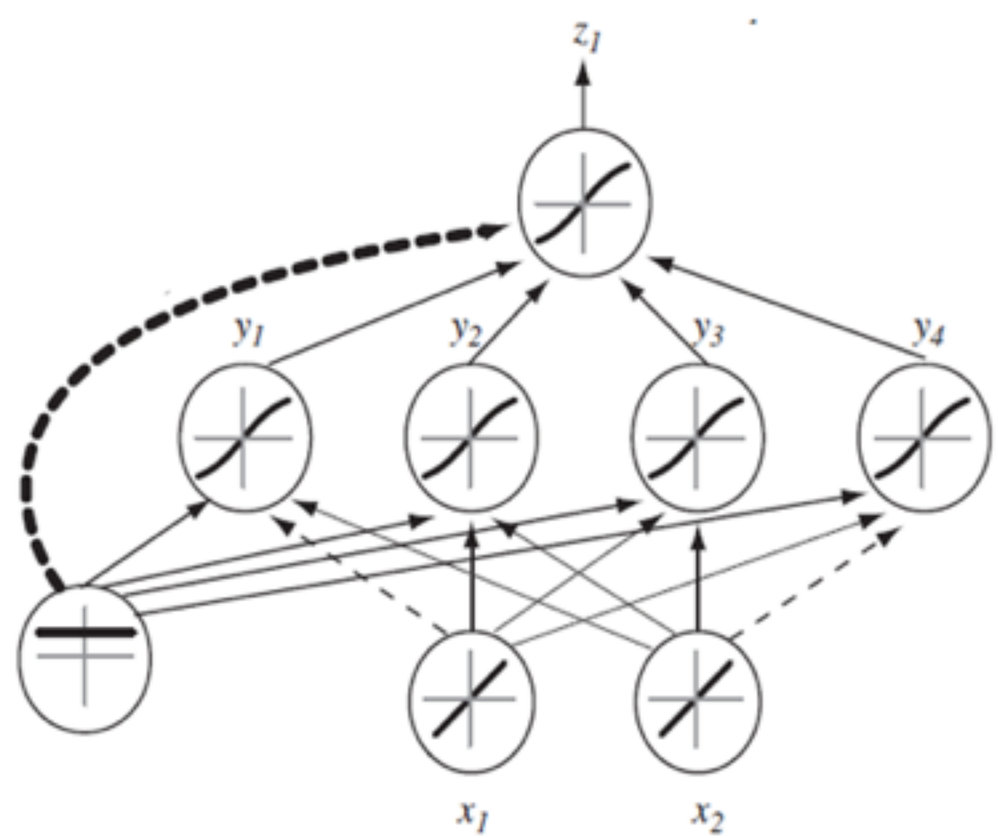
- Any continuous function from input to output can be implemented in a three-layer network, given sufficient number of hidden units n_H, proper nonlinearities, and weights.

MLP Constructing Complex Functions

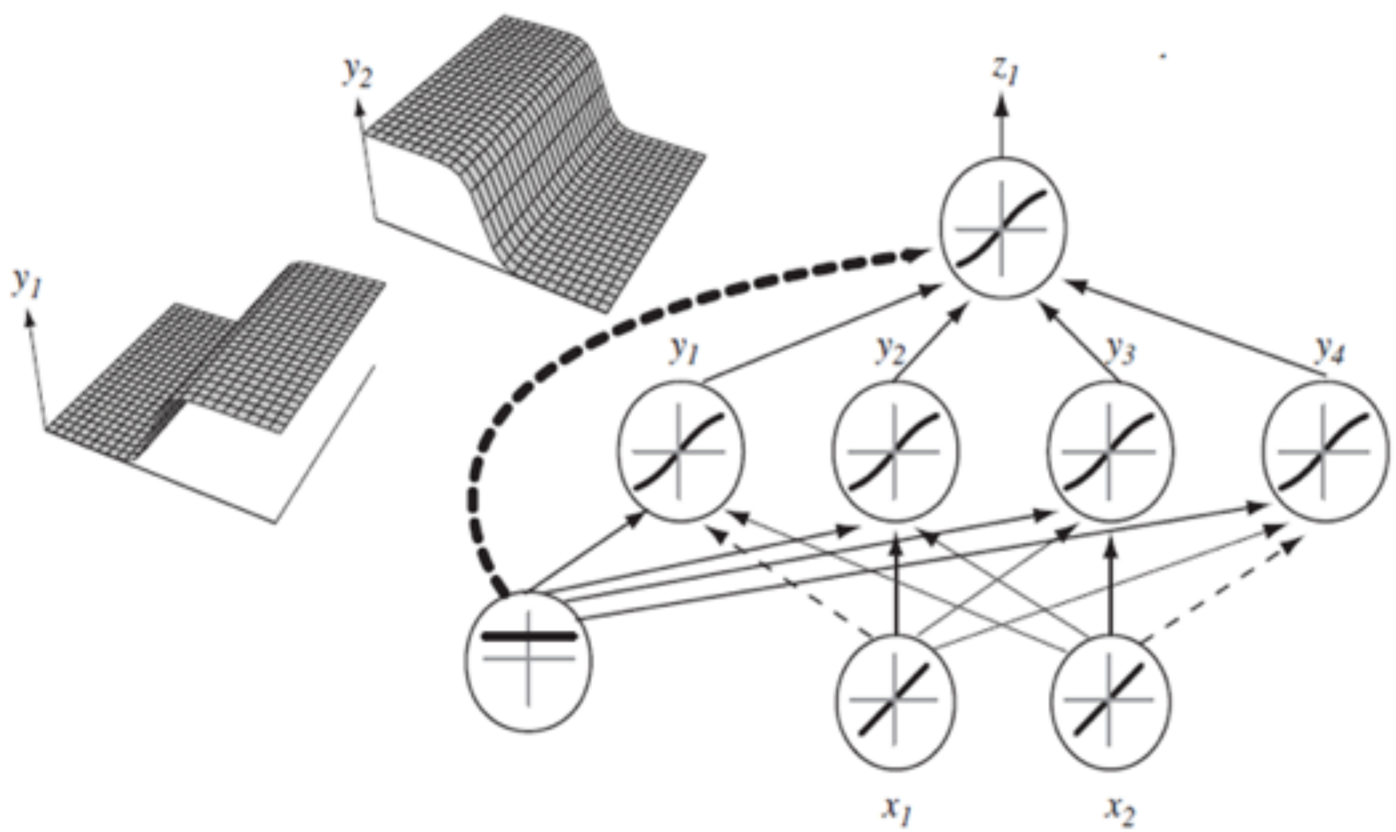


MLP Constructing Complex Functions

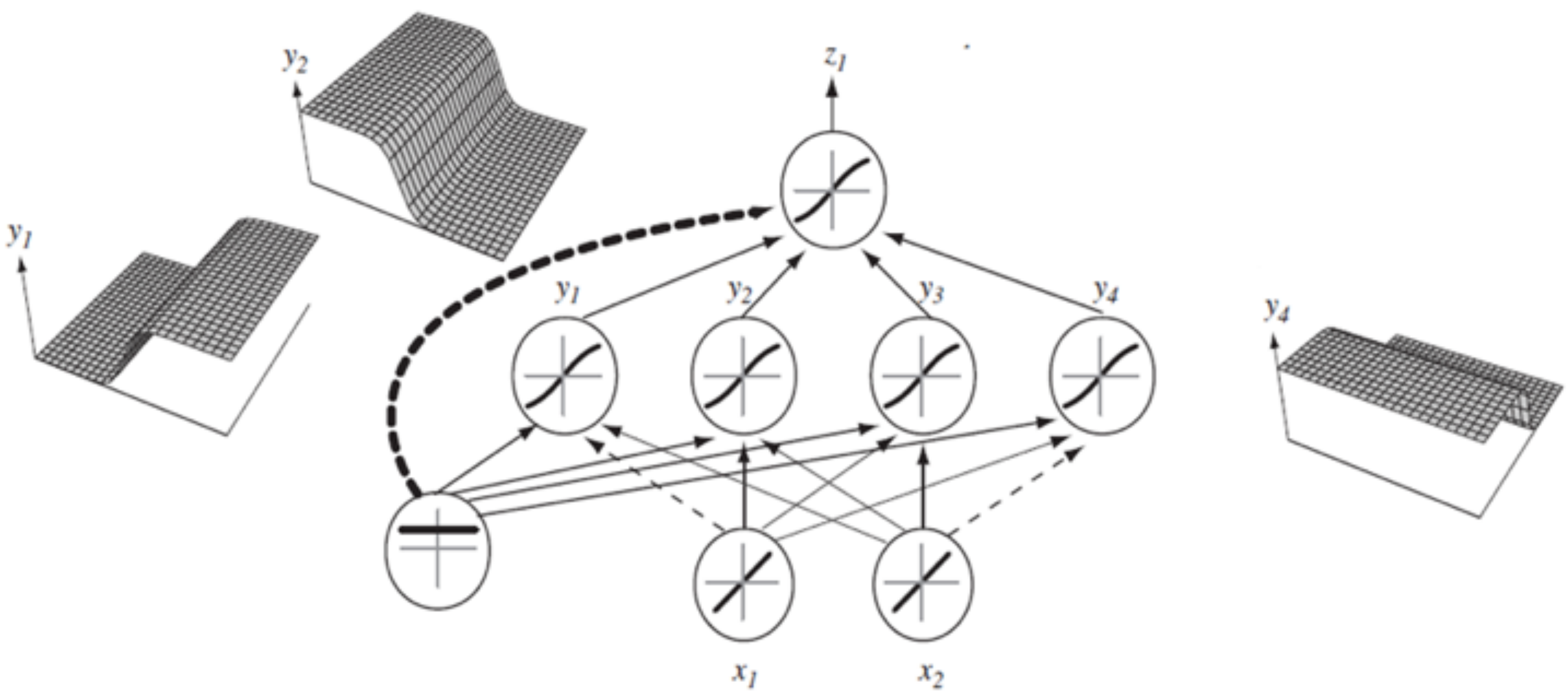




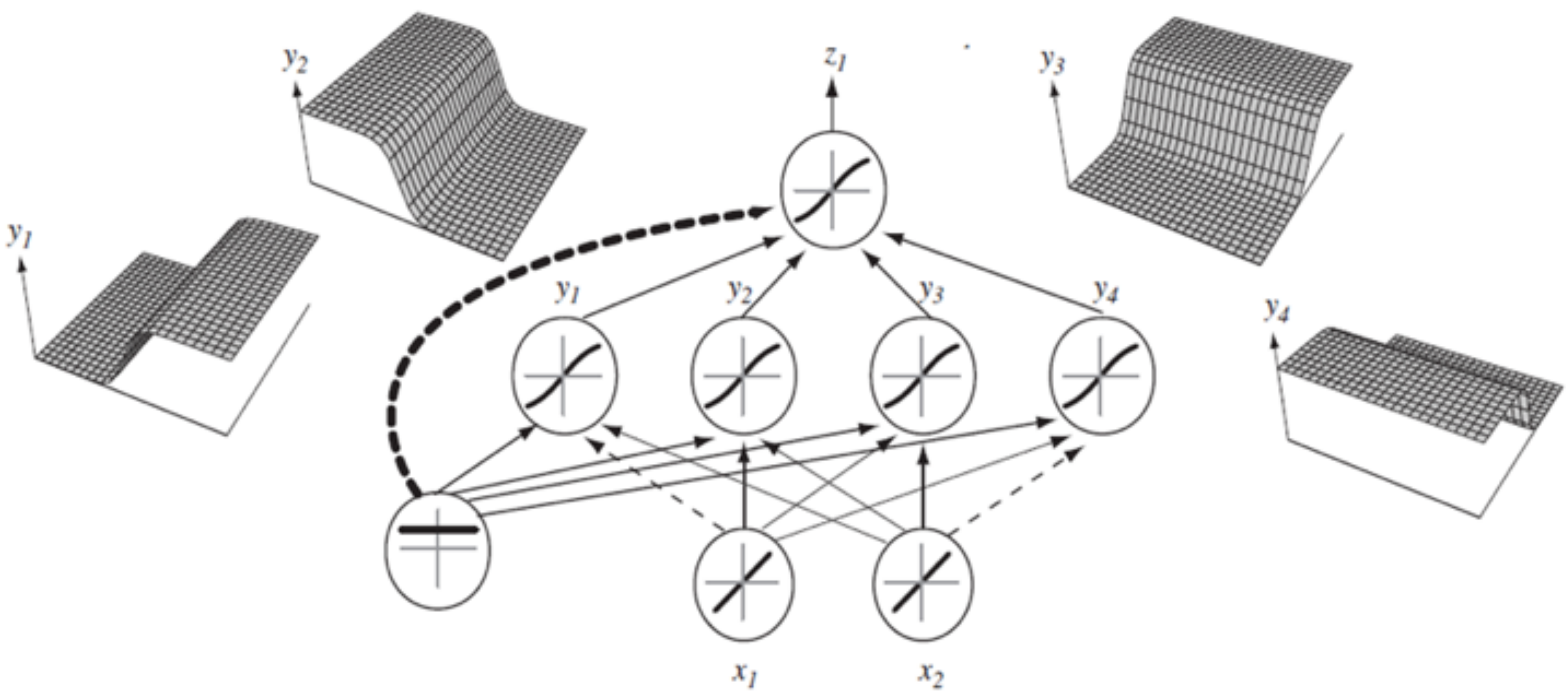
MLP Constructing Complex Functions



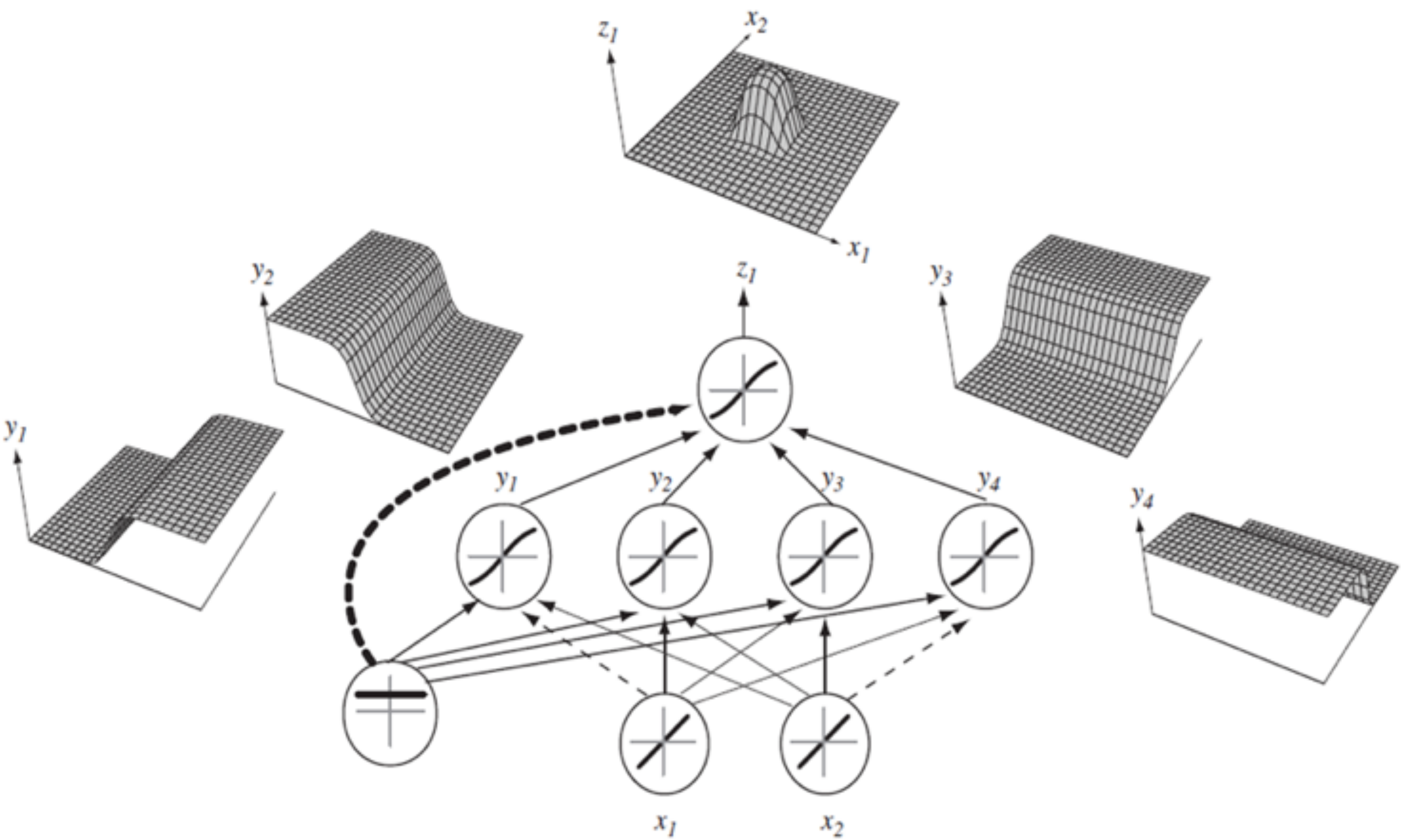
MLP Constructing Complex Functions



MLP Constructing Complex Functions



MLP Constructing Complex Functions



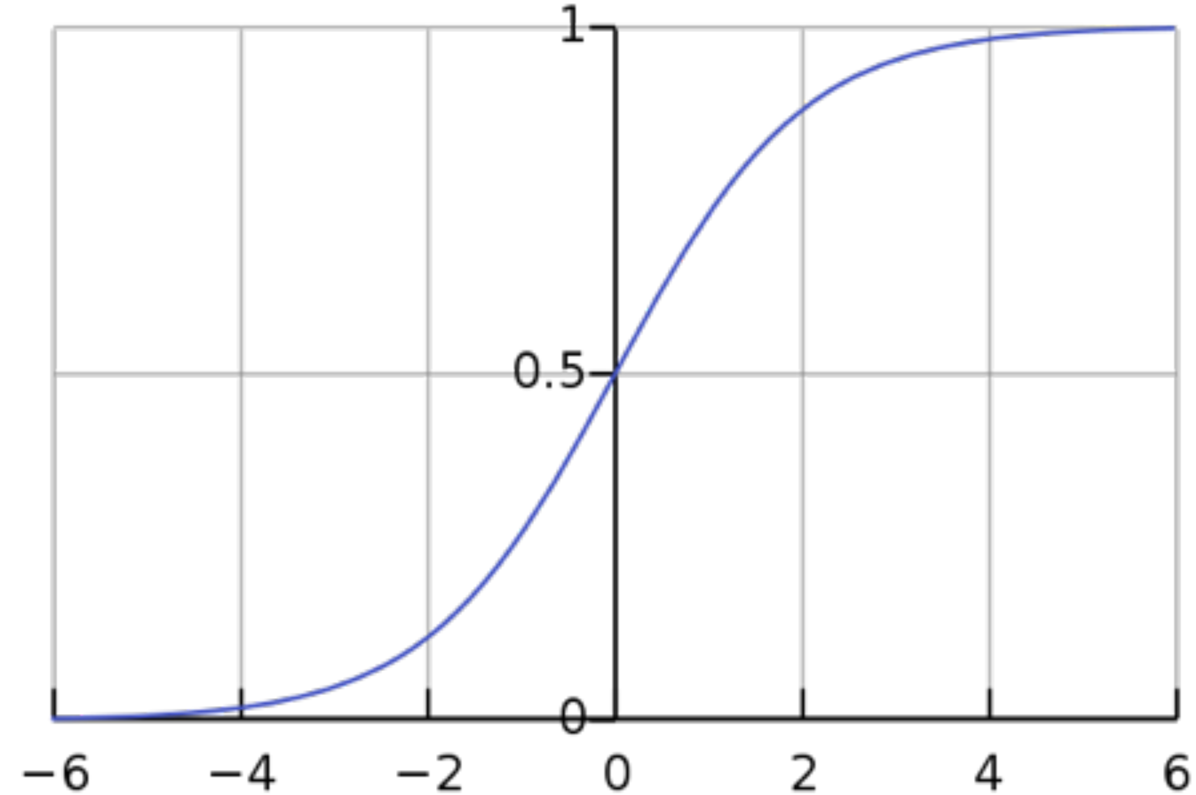
Activation Functions

- The activation function in a neural network is a function used to transform the activation level of a unit (neuron) into an output signal.
- The activation function essentially divides the original space into typically two partitions, having a "squashing" effect.
- The activation function is usually required to be a non-linear function.
- The input space is mapped to a different space in the output.
- There have been many kinds of activation functions proposed over the years (640+), however, the most commonly used are the **Sigmoid**, **Tanh**, ReLU, and Softmax.

The Logistic (or Sigmoid) Activation Function

- The sigmoid function is a special case of a logistic function given by f(x) and the plot below
- non-linear (slope varies)
- continuously differentiable
- monotonically increasing
- NB: e is the natural logarithm

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



Sigmoid Function - Derivative

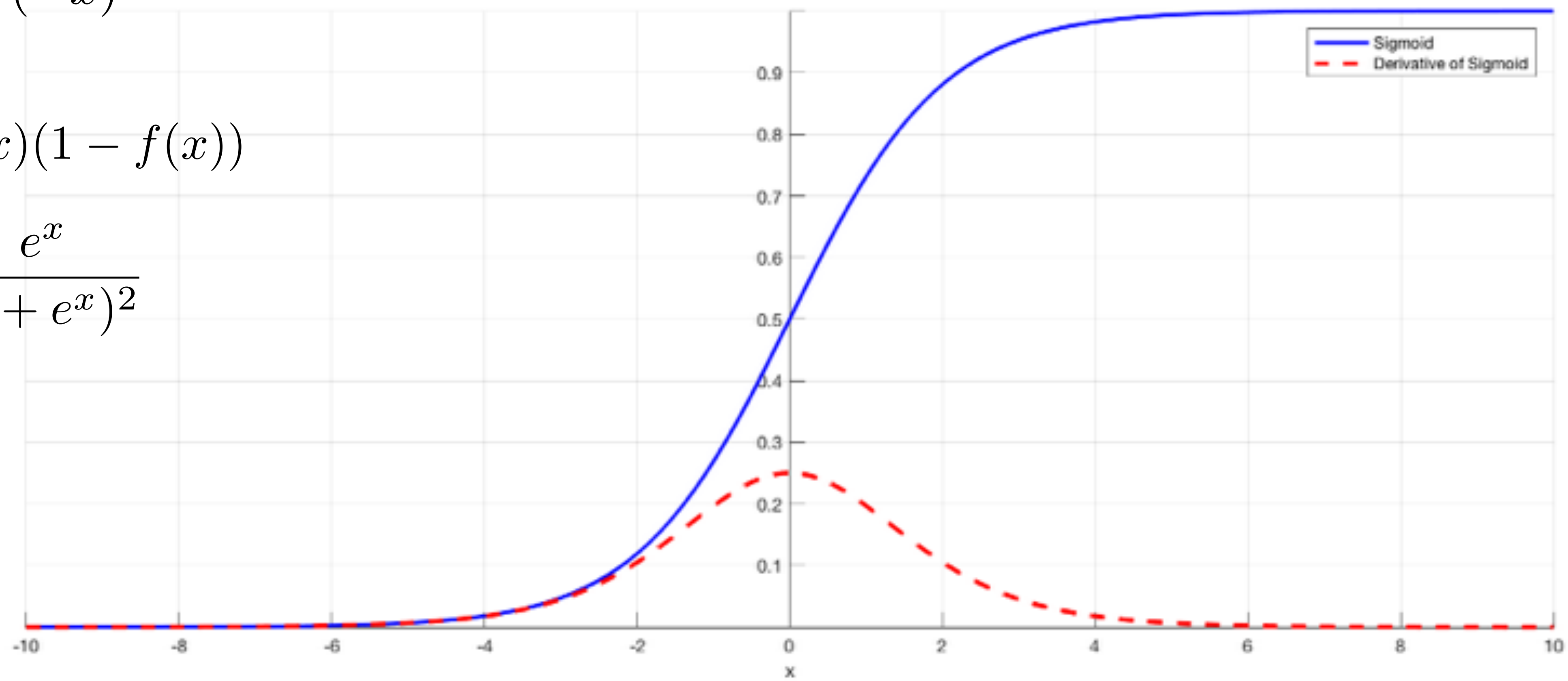
- The sigmoid function has an easily calculated derivative which is used in the back propagation algorithm

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

$$1 - f(x) = f(-x)$$

$$\frac{d}{dx}f(x) = f(x)(1-f(x))$$

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

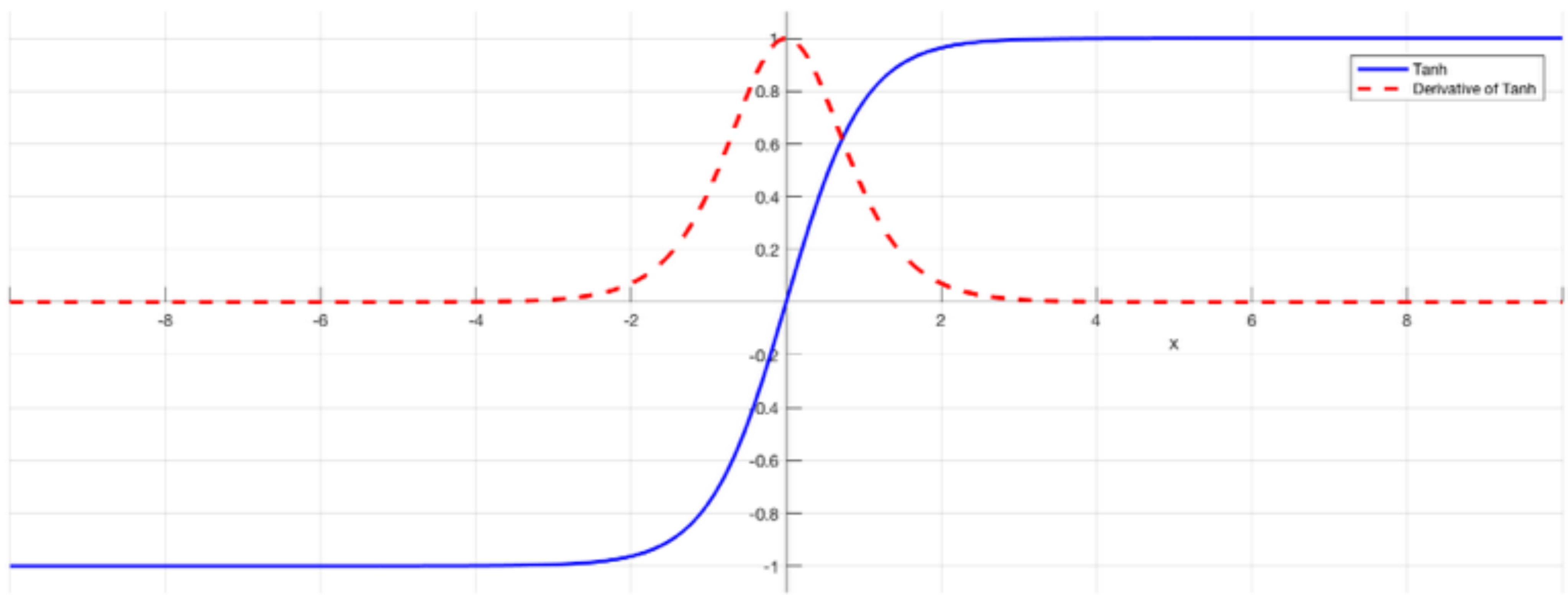


Reference: https://isaacchanghau.github.io/2017/05/22/Activation-Functions-in-Artificial-Neural-Networks/

The Hyperbolic Tangent Activation Function

- The tanh function is also "s"-shaped like the sigmoidal function, but the output range is (-1, 1)

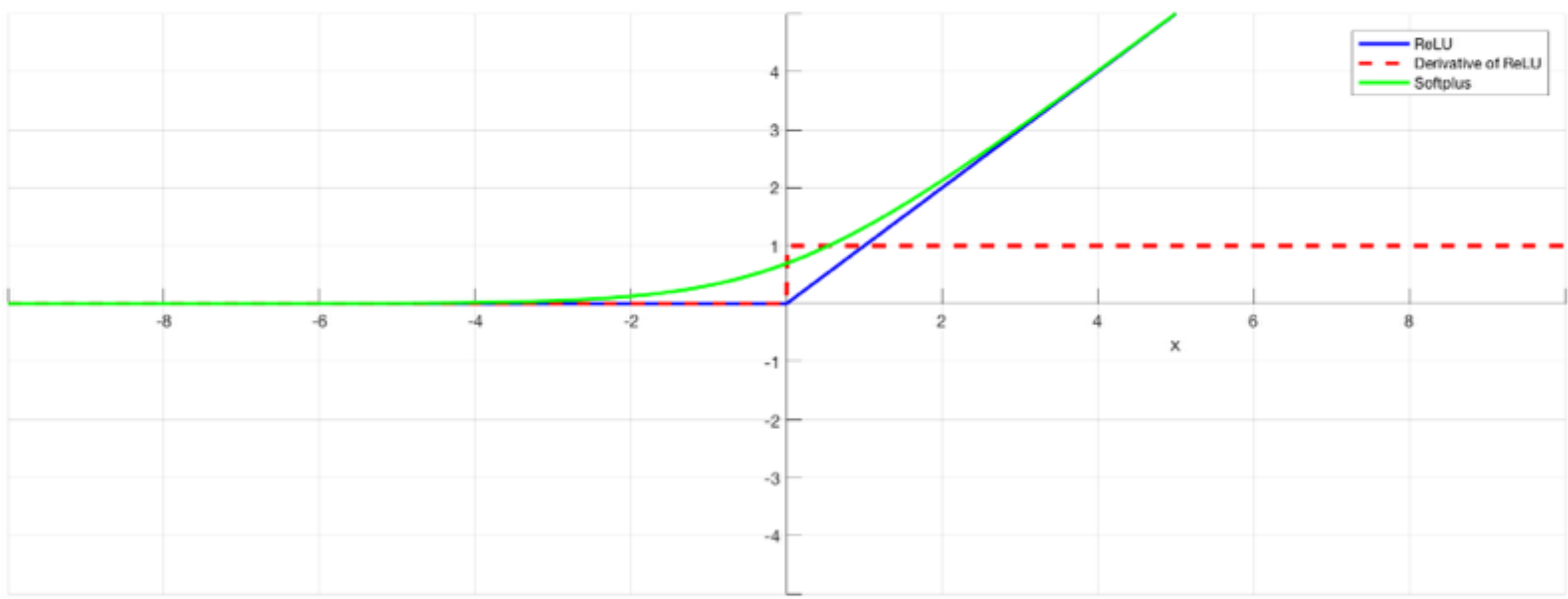
$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \qquad tanh'(x) = 1 - tanh^2(x)$$



Reference: https://isaacchanghau.github.io/2017/05/22/Activation-Functions-in-Artificial-Neural-Networks/

Rectified Linear Units (ReLU)

- The ReLU (used for the hidden layer neurons) is defined as $$f(x) = max(0, x)$$
- The range of the ReLU is between 0 to ∞



Multi-class Problems

- Binary classification - either 1 or 2 neurons in the output layer depending on what activation function you choose (1 neuron - tanh or sigmoid, 2 neurons - softmax)
- Multiclass classification - Number of neurons in the output layer, K, corresponds to the number of classes in your dataset - K neurons with softmax activation function

Softmax Function

- Used in the output layer of a neural network-based classifier

- Generalization of the logistic function that "squashes" a K-dimensional vector **z** of arbitrary real values to a K-dimensional vector σ(z) of real values in the range [0,1] that add up to 1

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_j}} \quad \text{for j = 1, …., K}$$

Backpropagation Algorithm

- Error $$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2$$
  $$= \frac{1}{2}||\mathbf{t} - \mathbf{z}||^2$$
- Change in weight for gradient descent $$\Delta\mathbf{w} = -\eta\frac{\partial J}{\partial \mathbf{w}}$$
- Gradient descent update $$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m)$$

Error Backpropagation Algorithm

- Hidden to Output Layer Weights

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}}$$
$$= -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

where, change in error with respect to the activation of the unit,

$$\delta_k = -\frac{\partial J}{\partial net_k}$$

Easy form for δ_k,

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k)f^{'}(net_k)$$

Error at output x slope of nonlinearity

Also with respect to weights net is differentiated easily $\frac{\partial net_k}{\partial w_{kj}} = y_j$

Error Backpropagation (cont'd)

- Units internal to the network have no explicit error signal
- Chain rule of differentiation helps!

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

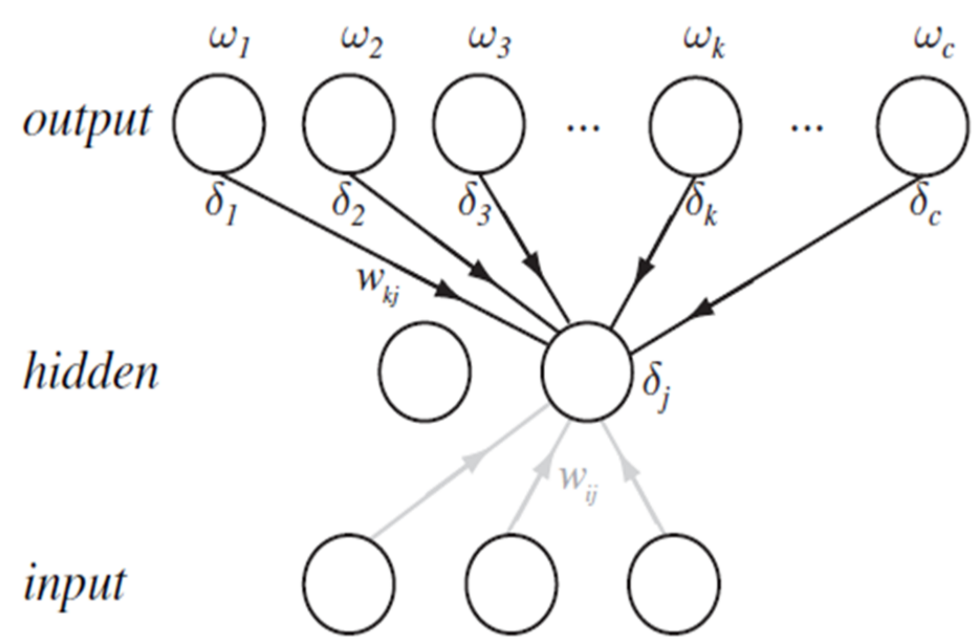
$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2] \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)f^{'}(net_k)w_{kj}
\end{aligned}$$

Error propagation (cont'd)

$$\delta_j \;=\; f'(\mathrm{net}_j)\sum_{k=1}^{c} w_{kj}\,\delta_k$$

and the update rule

$$\delta w_{ji} = \eta\, x_i\, \delta_j \;=\; \eta\left[\sum_{k=1}^{c} w_{kj}\delta_k\right] f'(\mathrm{net}_j)\, x_i$$



- Signal y_j propagates forward
- δ_j's propagate backwards