

Artificial Neural Networks II



COMPCSI 762

Instructor: Thomas Lacombe

Adapted from: Meng-Fen Chiang

WEEK 11

OUTLINE

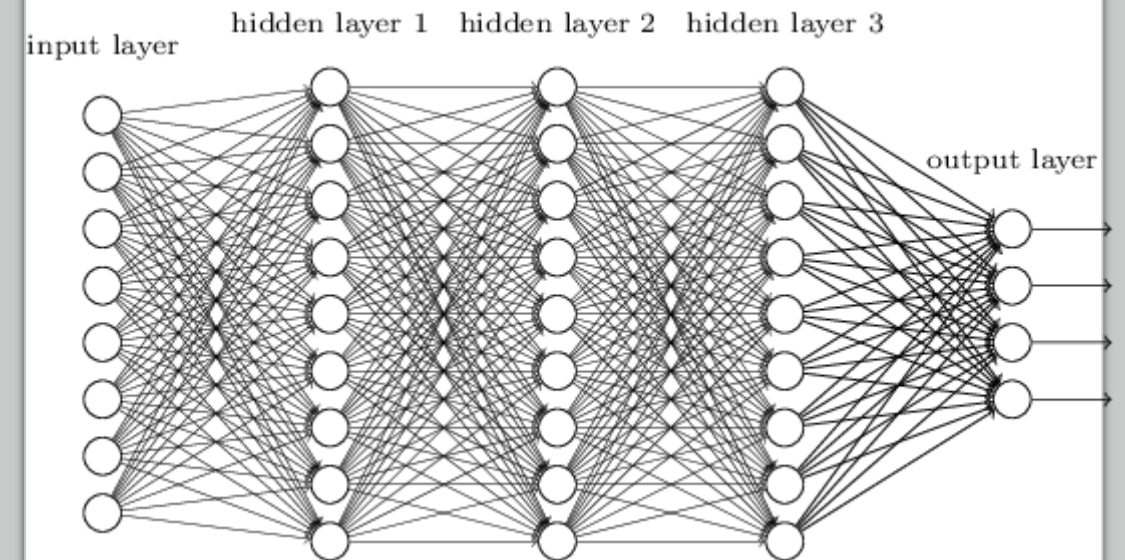
Introduction

Artificial Neural Networks (ANN)

- Single Unit: Architecture of Perceptron (NN1)
- Connection to Shallow Machine Learning (NN1)
- Multi-Layer Feed-Forward Neural Network (NN2)

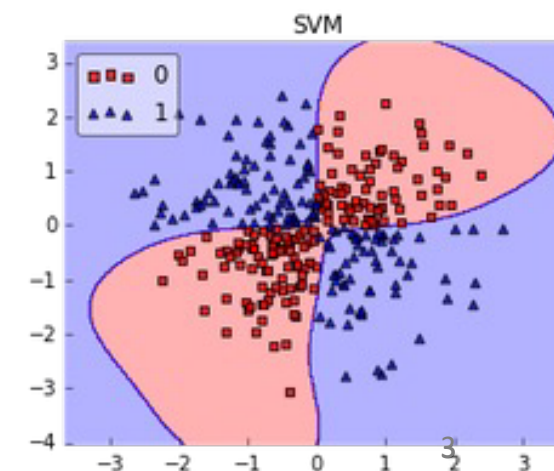
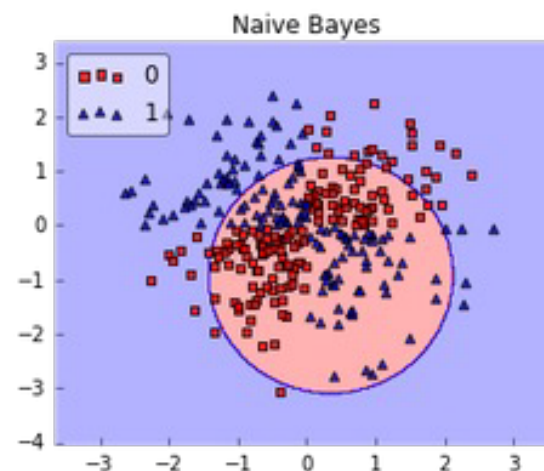
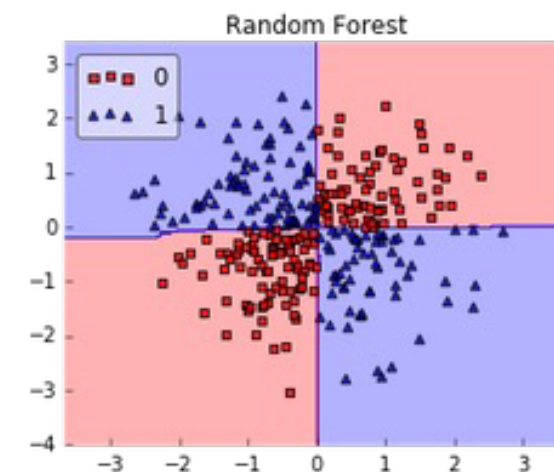
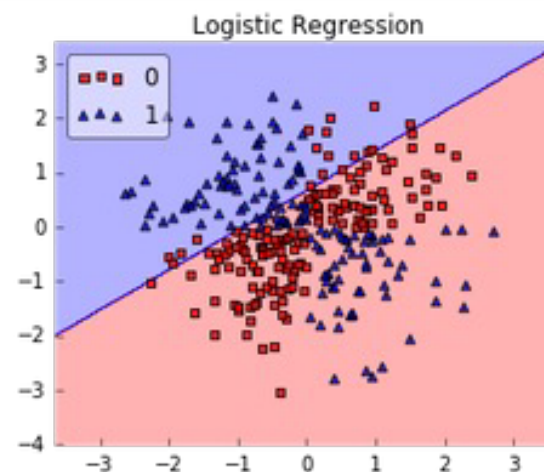
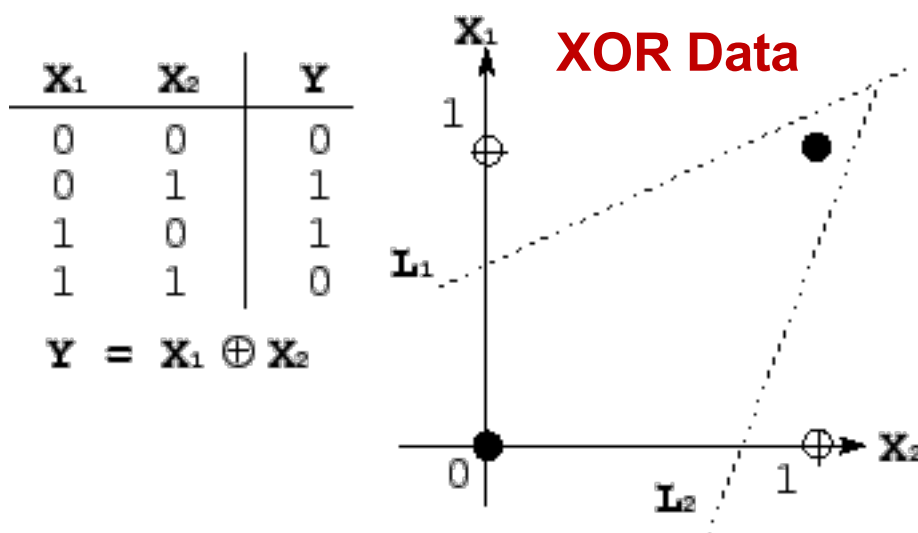
Design Issues (NN3)

Deep Learning / Large Language Models (NN4)



Non-Linearly Separable Data (XOR Data)

- Perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly



Multi-Layer Feed-Forward Neural Network (FFN)

Architecture: A **two-layer** network

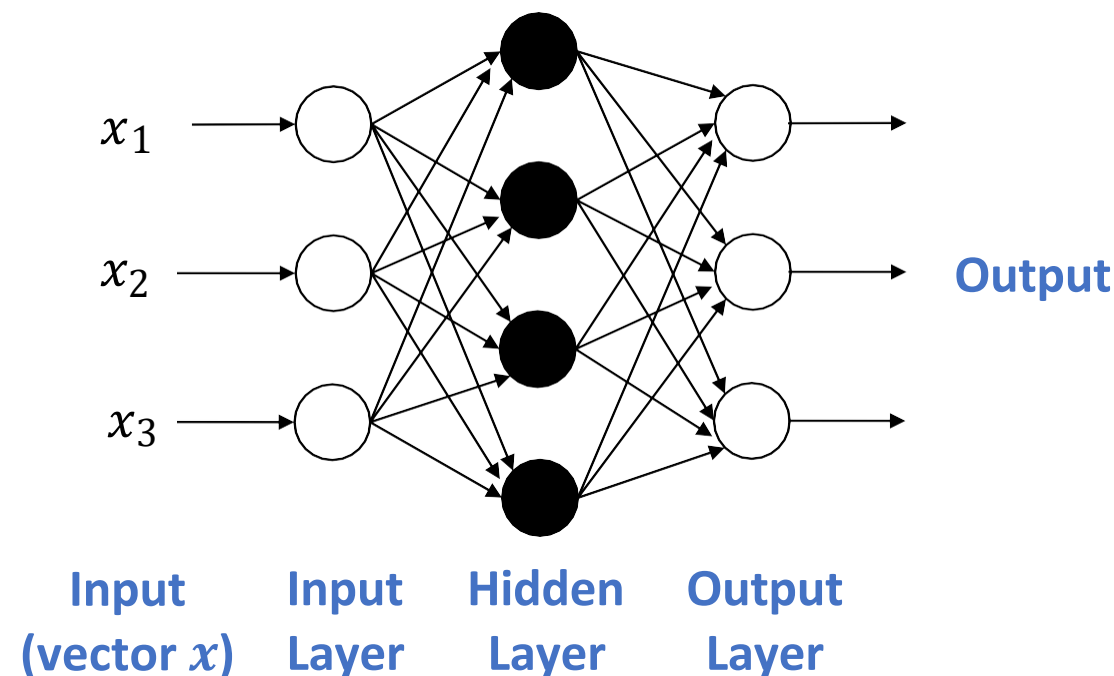
Activation Function:

- g : Nonlinear transformation
- e.g. sigmoid transformation

Hidden Layer: $\mathbf{h} = \mathbf{g}(W^{(1)} \mathbf{x} + b^{(1)})$

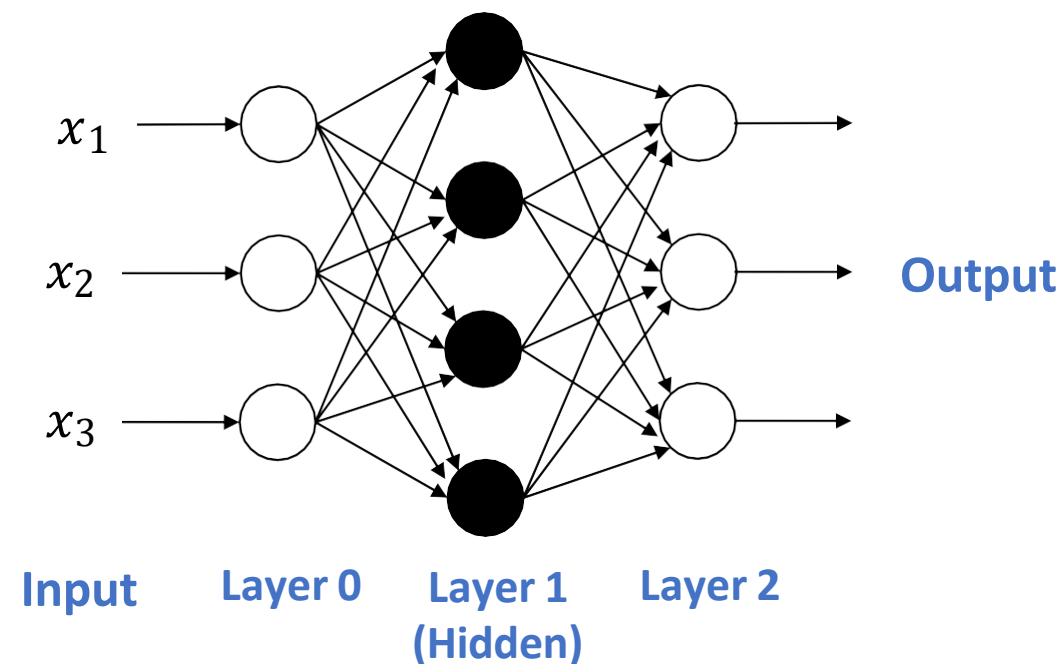
Output Layer: $\mathbf{o} = \mathbf{g}(W^{(2)} \mathbf{h} + b^{(2)})$

 
 Weight Matrix Bias Term



Multi-Layer (FF) Neural Network

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are then weighted and fed simultaneously to a hidden layer
- The number of hidden layers is arbitrary
- The network is **feed-forward**: None of the weight cycles back to an input unit or to an output unit of a previous layer



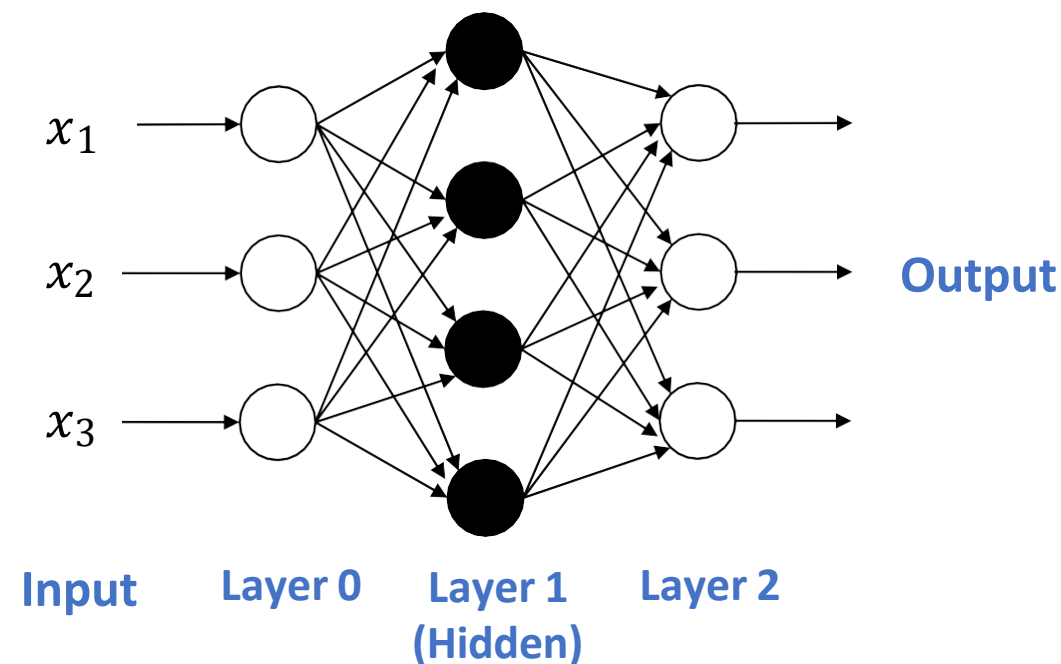
Multi-Layer Neural Network

- Every node in a **hidden layer** operates on activations from preceding layer and transmits activations forward to nodes of next layer

$$h = g(\underbrace{W^{(1)} x + b^{(1)}}_{\text{Linear Predictor}})$$

Activation Value \swarrow \downarrow Activation Function

- Networks perform **non-linear regression**: Given enough hidden units and enough training samples, they can closely approximate any continuous function

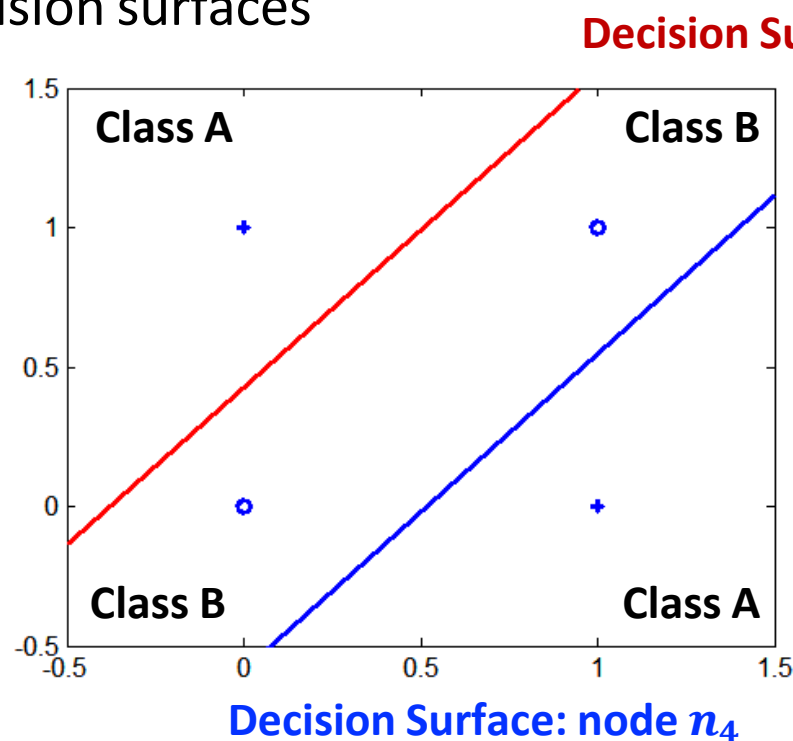
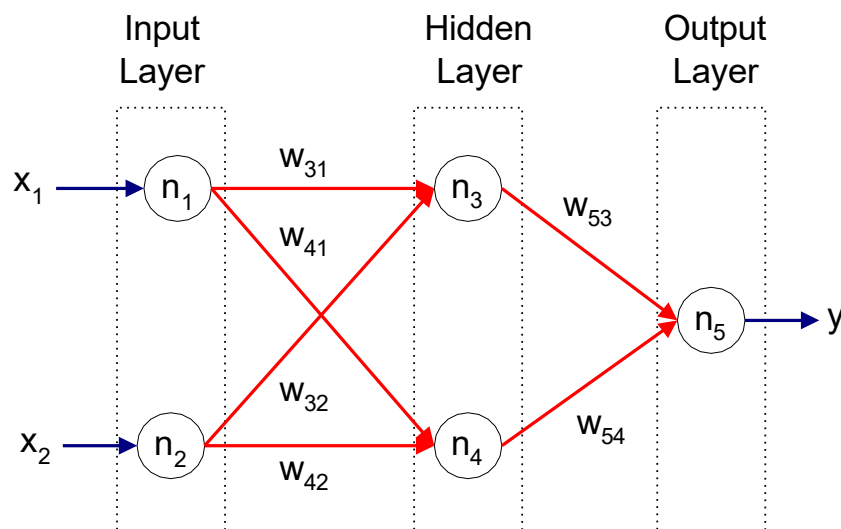


Multiple Layer Neural Network

- Given a set of training data $S = ((x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}))$
- $y^{(i)}$ is categorical: classification task (multi-class or binary)
- $y^{(i)}$ is continuous: regression task
- Goal: Find \mathbf{w} , such that minimize the empirical risk is minimized
- Solution: Stochastic gradient descent (SGD) + chain rule = **Backpropagation**

Example: XOR Problem

- With at least one hidden layer + non-linear activation function, multi-layer FFN can solve classification task involving nonlinear decision surfaces



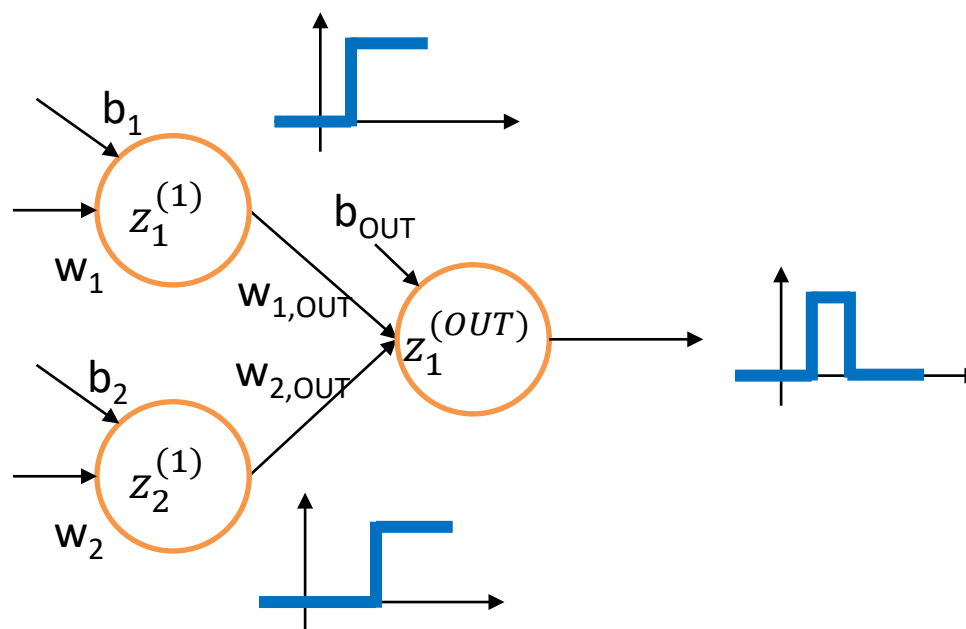
Universal Function Approximation Theorem

Hornik theorem 1: Whenever the activation function is *bounded and nonconstant*, then, for any finite measure μ , standard multilayer feedforward networks can approximate any function in $L^p(\mu)$ (the space of all functions on R^k such that $\int_{R^k} |f(x)|^p d\mu(x) < \infty$) arbitrarily well, provided that sufficiently many hidden units are available.

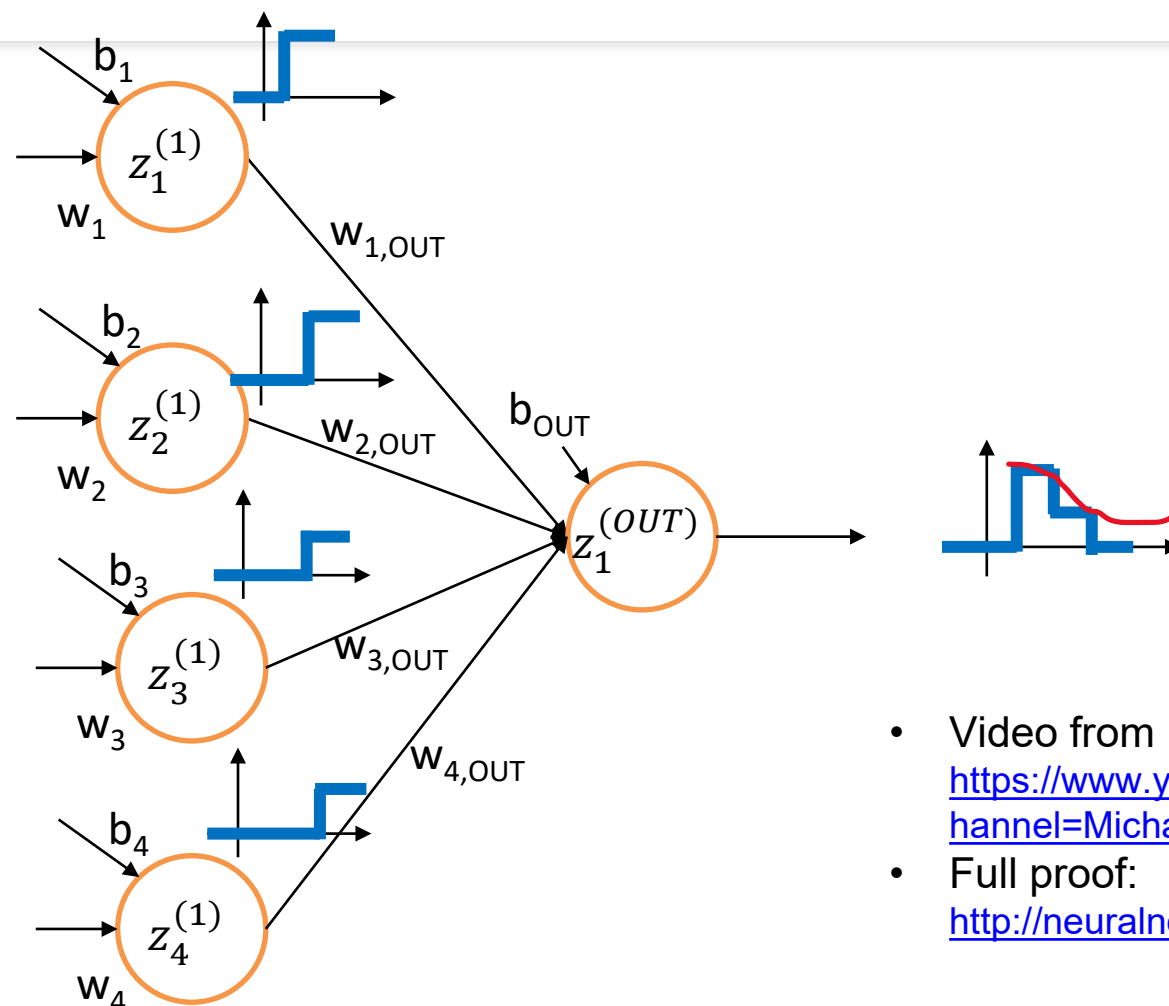
Hornik theorem 2: Whenever the activation function is *continuous, bounded and non-constant*, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on X arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

In summary: Given any continuous function $f(x)$, if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate $f(x)$.

Intuition for the theorem



Intuition for the theorem



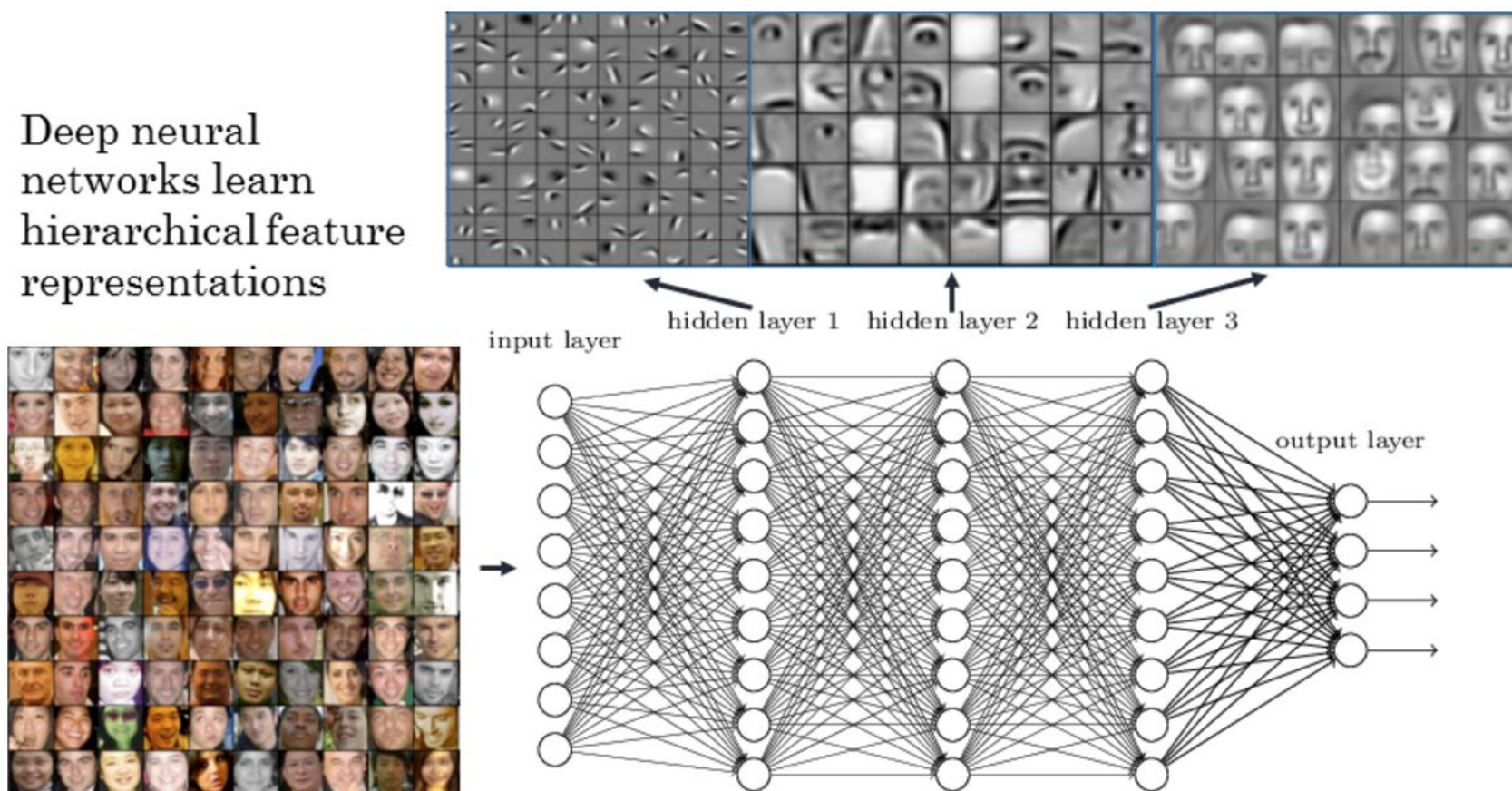
- Video from Michael Nielsen:
https://www.youtube.com/watch?v=ljqkc7OLenI&ab_c hannel=MichaelNielsen
- Full proof:
<http://neuralnetworksanddeeplearning.com/chap4.html>

Multiple Hidden Layers

- Activations at hidden layers can be viewed as features extracted as functions of inputs
- Every **hidden** layer represents a level of abstraction
 - Complex features are compositions of simpler features
- Number of layers is the depth of ANN → Deeper networks express complex hierarchy of features

Multiple Hidden Layers

Deep neural networks learn hierarchical feature representations



Le Cun et al. (2015)
 Raphael et al. (2019)

Capability of Neural Network

- M_1 : 0 hidden layer + linear activation function → linear surface
- M_2 : 0 hidden layer + non-linear activation function → linear surface (LR)
- M_3 : 1 hidden layer + linear activation function → combination of linear surface
- M_4 : 1 hidden layer + non-linear activation function → non-linear surface (MLP)

Training: Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- **Loss Function**. For each training tuple, the **weights** are modified to **minimize the loss** between the network's prediction and the actual target value, say mean squared error
- Stochastic gradient descent + chain rule = **Backpropagation**
 - Modifications are made in the “backwards” direction
 - From the **output layer**, through each **hidden layer** down to the **first hidden layer**, hence “backpropagation”

RECAP: (1) Chain Rule (2) Gradient Descent

- **The Chain Rule:** if f and g are both differentiable and $F(x) = f \circ g$ is the composite function defined by $F(x) = f(g(x))$, then F is differentiable and F' is given by the product:

$$F'(x) = f'(g(x)) g'(x)$$

In Leibniz notation, if $y = f(u)$ and $u = g(x)$ are both differentiable functions, then

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

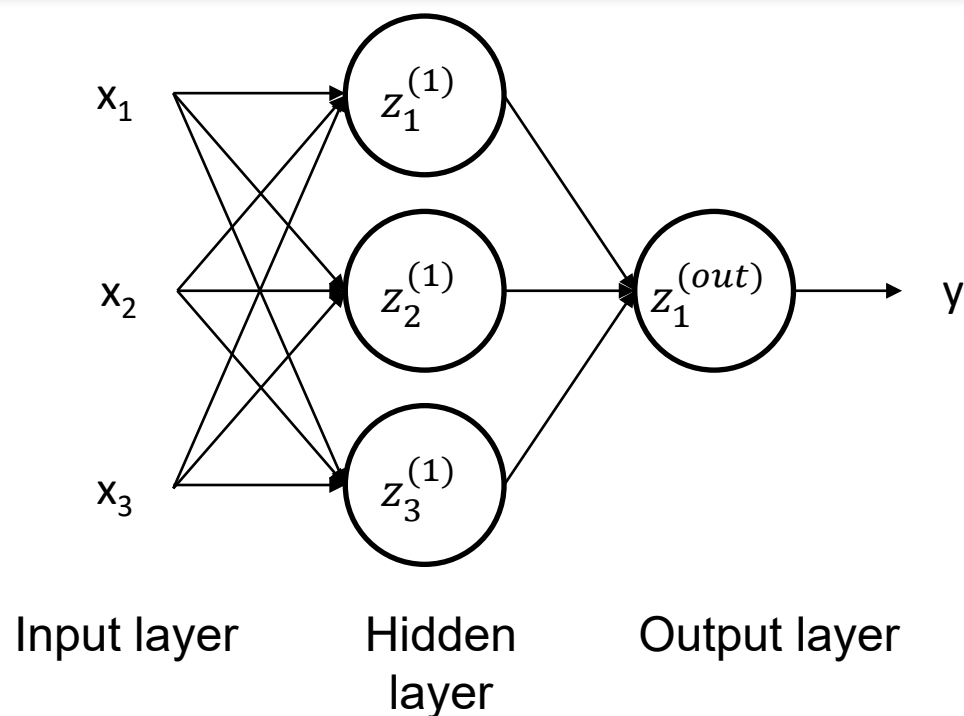
- **Gradient descent:** Update parameters in the direction of “maximum descent” in the loss function across all points
- **Stochastic gradient descent (SGD):** update the weight for every instance
- **Mini-batch SGD:** update over min-batches of instances

Training: Backpropagation

- For each training instance:
 1. **Make a prediction (forward pass/propagation)**
 2. Measure the error/loss
 3. **Go through each layer in reverse to measure the error contribution from each connection (backward pass/propagation)**
 4. Slightly tweak the connection weights to reduce the error (SGD step)

Until stopping criterion is reached

Backpropagation example step by step



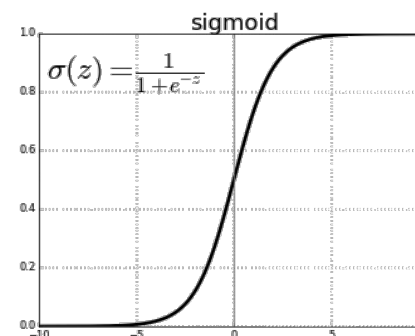
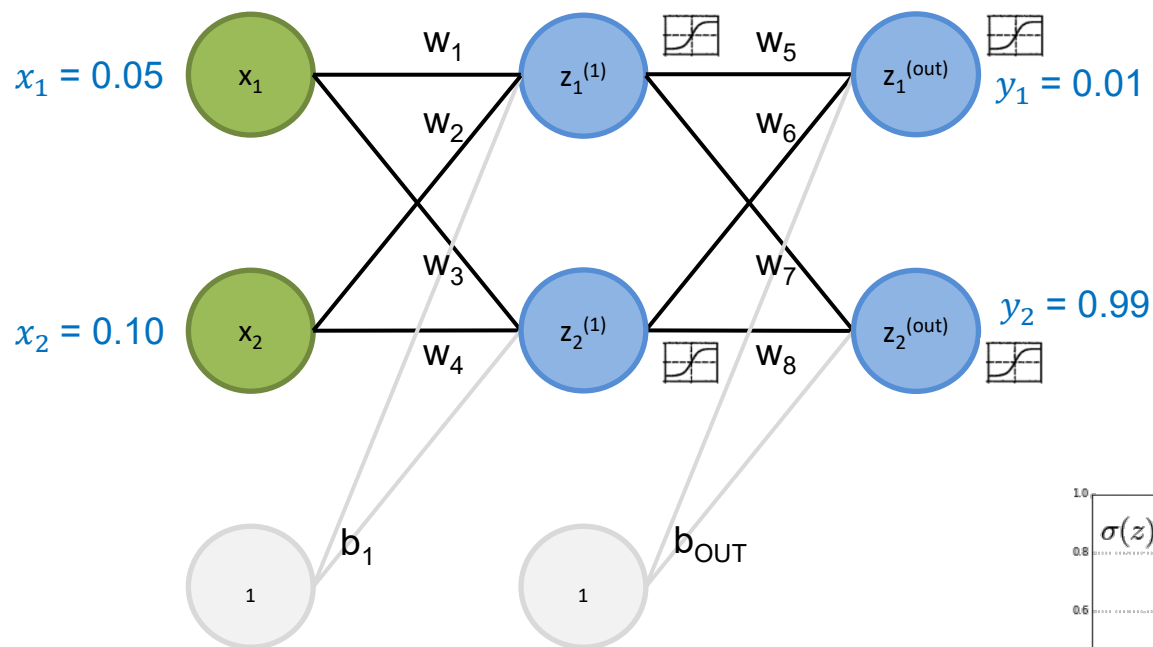
$$out_{z_i}^{(k)} = g\left(in_{z_i}^{(k)}\right)$$

$$in_{z_i}^{(k)} = \sum_j w_{i,j}^{(k-1,k)} out_{z_j}^{(k-1)}$$

$in_{z_i}^{(k)}$ is the input of neuron i in layer k (after input function)

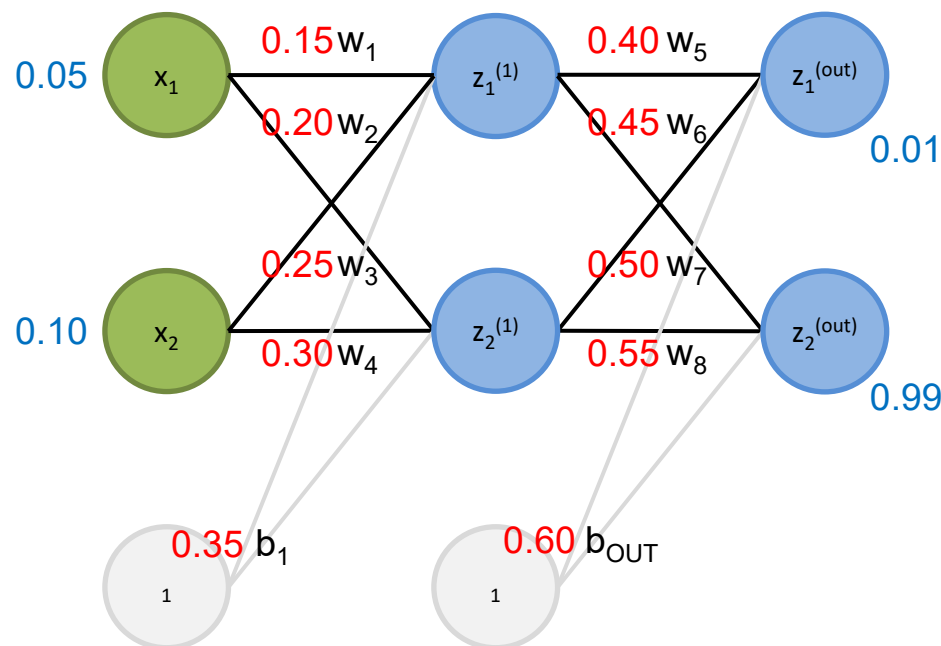
$out_{z_i}^{(k)}$ is the output of neuron i in layer k (after activation)

Backpropagation example step by step



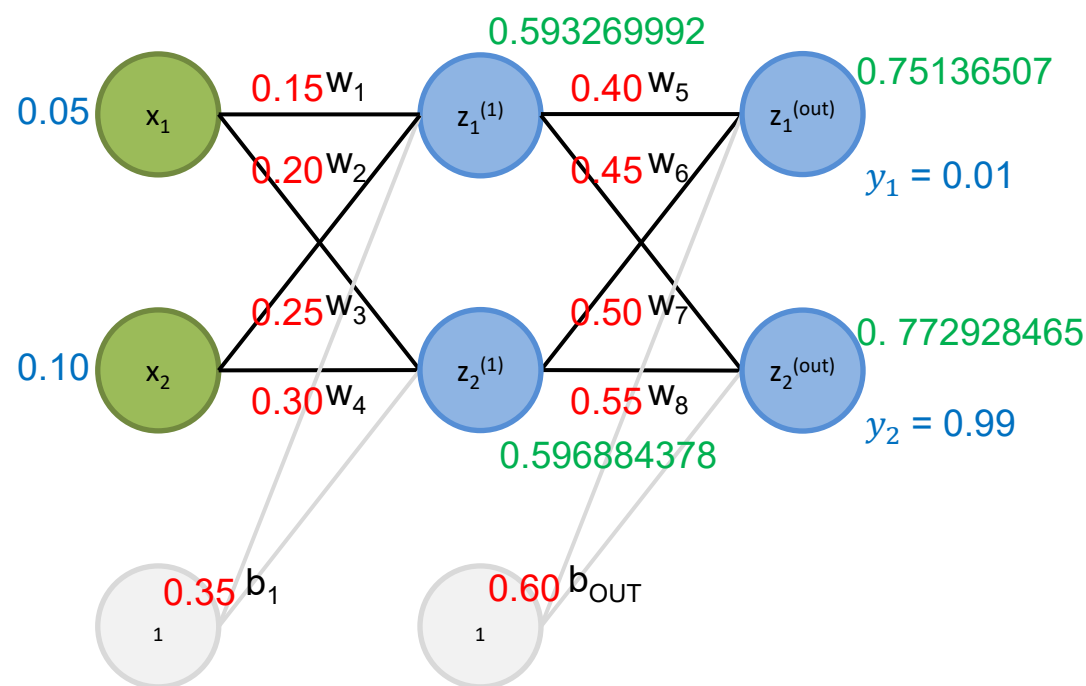
Backpropagation example step by step

- Initialisation of the weights



Backpropagation example step by step

- Forward propagation



$$\begin{aligned} \text{out_}z_1^{(1)} &= S(\overbrace{w_1 * x_1 + w_2 * x_2 + b_1 * 1}^{\text{in_}z_1^{(1)}}) \\ &= S(0.15 * 0.05 + 0.20 * 0.10 + 0.35 * 1) \\ &= S(0.3775) \\ &= 0.593269992 \end{aligned}$$

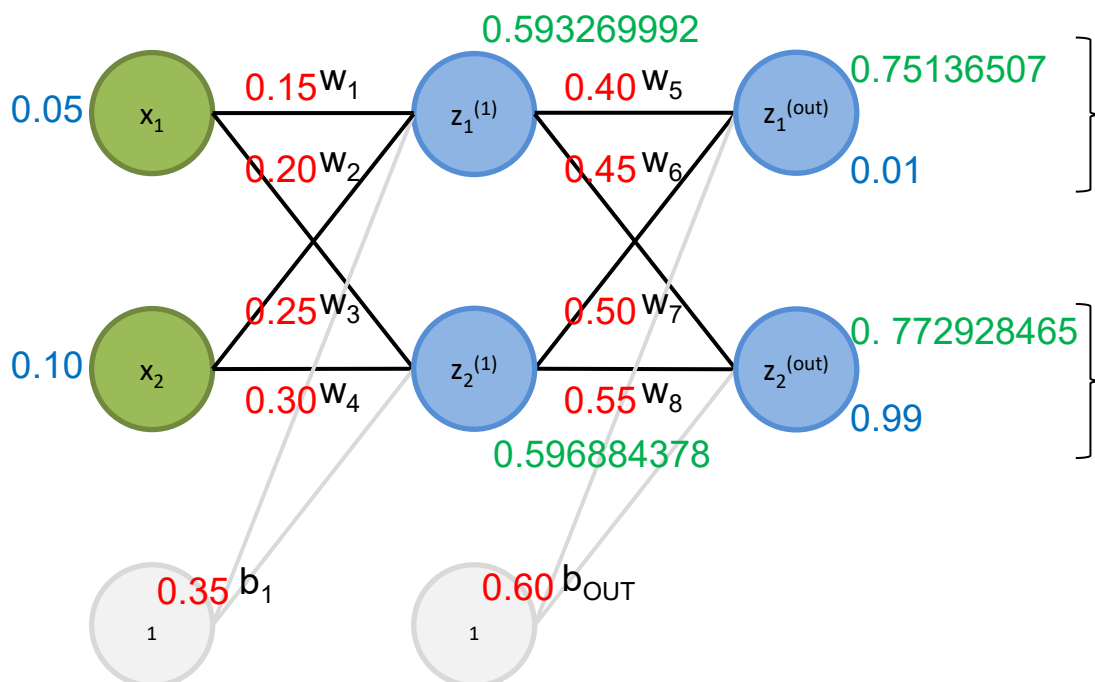
$$\begin{aligned} \text{out_}z_2^{(1)} &= S(\overbrace{w_3 * x_1 + w_4 * x_2 + b_1 * 1}^{\text{in_}z_2^{(2)}}) \\ &= S(0.25 * 0.05 + 0.30 * 0.10 + 0.35 * 1) \\ &= 0.596884378 \end{aligned}$$

$$\begin{aligned} \text{out_}z_1^{(out)} &= S(\overbrace{w_5 * \text{out_}z_1^{(1)} + w_6 * \text{out_}z_2^{(1)} + b_{OUT} * 1}^{\text{in_}z_1^{(out)}}) \\ &= 0.75136507 \end{aligned}$$

$$\begin{aligned} \text{out_}z_2^{(out)} &= S(\overbrace{w_7 * \text{out_}z_1^{(1)} + w_8 * \text{out_}z_2^{(1)} + b_{OUT} * 1}^{\text{in_}z_2^{(out)}}) \\ &= 0.772928465 \end{aligned}$$

Backpropagation example step by step

- Calculating the total error



$$E(y, \hat{y}) = \frac{1}{2} ||y - \hat{y}||^2$$

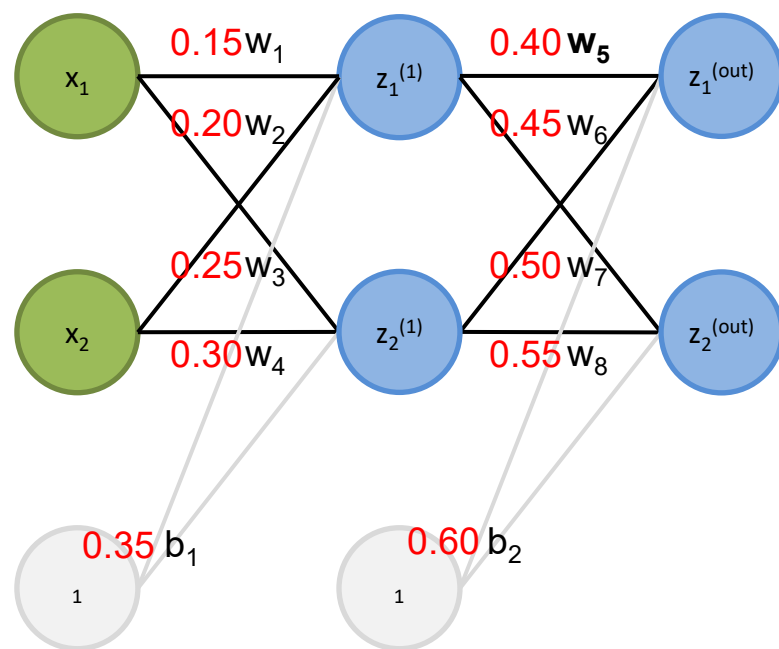
$$E_{z_1^{(out)}} = \frac{1}{2} (0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{z_2^{(out)}} = \frac{1}{2} (0.99 - 0.772928465)^2 = 0.023560026$$

$$E_{tot}^{(out)} = E_{z_1^{(out)}} + E_{z_2^{(out)}} = 0.298371109$$

Backpropagation example step by step

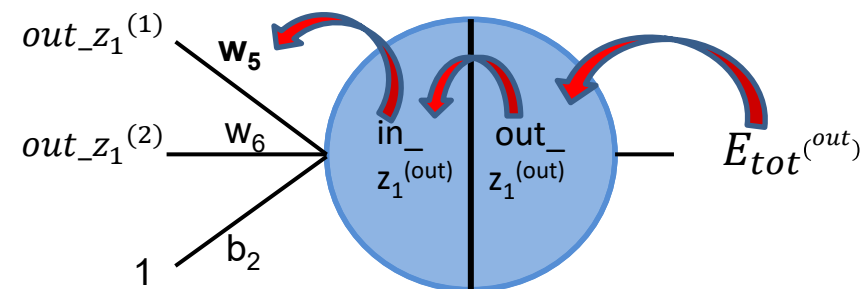
- Back propagation (output layer)



w_5 influence on $E_{tot}^{(out)}$? $\frac{\partial E_{tot}^{(out)}}{\partial w_5}$

Chain rule:

$$\frac{\partial E_{tot}^{(out)}}{\partial w_5} = \frac{\partial E_{tot}^{(out)}}{\partial out_{z_1}^{(out)}} * \frac{\partial out_{z_1}^{(out)}}{\partial in_{z_1}^{(out)}} * \frac{\partial in_{z_1}^{(out)}}{\partial w_5}$$



Backpropagation example step by step

- Back propagation (output layer)

$$\frac{\partial E_{tot}^{(out)}}{\partial W_5} = \frac{\partial E_{tot}^{(out)}}{\partial out_{z_1}^{(out)}} * \frac{\partial out_{z_1}^{(out)}}{\partial in_{z_1}^{(out)}} * \frac{\partial in_{z_1}^{(out)}}{\partial W_5}$$

$$\begin{aligned} E_{tot}^{(out)} &= E_{z_1}^{(out)} + E_{z_2}^{(out)} \\ &= \frac{1}{2} (y_1 - out_{z_1}^{(out)})^2 + \frac{1}{2} (y_2 - out_{z_2}^{(out)})^2 \end{aligned}$$

$$\begin{aligned} \frac{\partial E_{tot}^{(out)}}{\partial out_{z_1}^{(out)}} &= 2 * \frac{1}{2} (y_1 - out_{z_1}^{(out)})^{2-1} * -1 + 0 \\ &= out_{z_1}^{(out)} - target_{z_1}^{(out)} \\ &= 0.75136507 - 0.01 \\ &= 0.74136507 \end{aligned}$$

Backpropagation example step by step

- Back propagation (output layer)

$$\frac{\partial E_{tot}^{(out)}}{\partial W_5} = \frac{\partial E_{tot}^{(out)}}{\partial out_{z_1}^{(out)}} * \frac{\partial out_{z_1}^{(out)}}{\partial in_{z_1}^{(out)}} * \frac{\partial in_{z_1}^{(out)}}{\partial W_5}$$

$$out_{z_1}^{(out)} = \frac{1}{1 + e^{-in_{z_1}^{(out)}}} \text{ (activation = sigmoid function)}$$

$$\text{Sigmoid} \rightarrow g'(x) = g(x)(1-g(x))$$

$$\begin{aligned} \frac{\partial out_{z_1}^{(out)}}{\partial in_{z_1}^{(out)}} &= out_{z_1}^{(out)} * (1 - out_{z_1}^{(out)}) \\ &= 0.75136507 * (1 - 0.75136507) \\ &= 0.186815602 \end{aligned}$$

Backpropagation example step by step

- Back propagation (output layer)

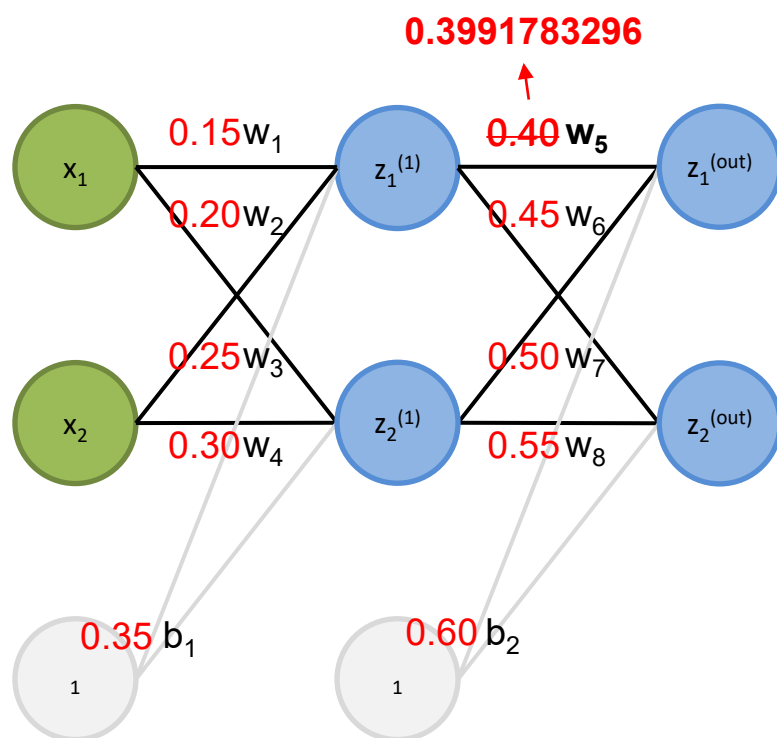
$$\frac{\partial E_{tot}^{(out)}}{\partial W_5} = \frac{\partial E_{tot}^{(out)}}{\partial out_{z_1}^{(out)}} * \frac{\partial out_{z_1}^{(out)}}{\partial in_{z_1}^{(out)}} * \frac{\partial in_{z_1}^{(out)}}{\partial W_5}$$

$$in_{z_1}^{(out)} = w_5 * out_{z_1}^{(1)} + w_6 * out_{z_2}^{(1)} + b_2 * 1$$

$$\begin{aligned} \frac{\partial in_{z_1}^{(out)}}{\partial W_5} &= out_{z_1}^{(1)} + 0 + 0 \\ &= out_{z_1}^{(1)} \\ &= 0.593269992 \end{aligned}$$

Backpropagation example step by step

- Back propagation (output layer)



$$\begin{aligned}\frac{\partial E_{tot}^{(out)}}{\partial w_5} &= \frac{\partial E_{tot}^{(out)}}{\partial out_{z_1^{(out)}}} * \frac{\partial out_{z_1^{(out)}}}{\partial in_{z_1^{(out)}}} * \frac{\partial in_{z_1^{(out)}}}{\partial w_5} \\ &= 0.74136507 * 0.186815602 * 0.593269992 \\ &= 0.082167041\end{aligned}$$

Gradient descent to decrease the error
(with learning rate $\alpha = 0.01$) :

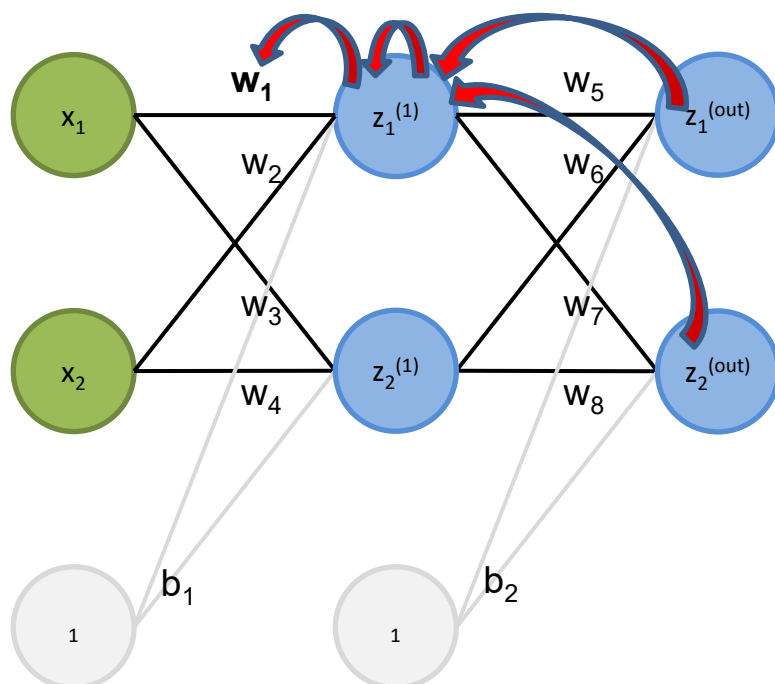
$$w_5 \leftarrow w_5 - \alpha * \frac{\partial E_{tot}^{(out)}}{\partial w_5}$$

$$w_5 \leftarrow 0.40 - 0.01 * 0.082167041$$

$$w_5 \leftarrow 0.3991783296$$

Backpropagation example step by step

► Backpropagation (hidden layer)



$$\frac{\partial E_{tot}^{(out)}}{\partial \mathbf{W}_1} = \underbrace{\frac{\partial E_{tot}^{(out)}}{\partial out_{z_1}^{(1)}} * \frac{\partial out_{z_1}^{(1)}}{\partial in_{z_1}^{(1)}}}_{\text{backward pass}} * \frac{\partial in_{z_1}^{(1)}}{\partial \mathbf{W}_1}$$

$$\frac{\partial E_{tot}^{(out)}}{\partial out_{z_1}^{(1)}} = \frac{\partial E_{z_1}^{(out)}}{\partial out_{z_1}^{(1)}} + \frac{\partial E_{z_2}^{(out)}}{\partial out_{z_1}^{(1)}}$$

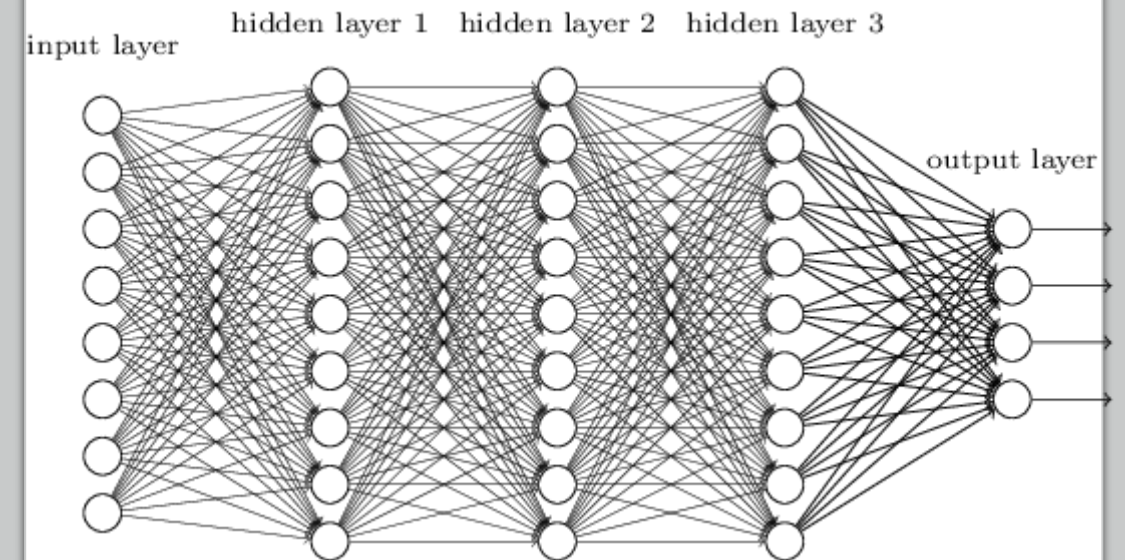


Jupyter Notebook

Multi-Layer Perceptron Coding Example
(Classifying Handwritten Digits)

SUMMARY

- Multi-Layer Perceptron Architecture
 - Multiple Hidden Layers
 - Nonlinear Activation Functions
- Training: Backpropagation algorithm
 - Example step by step



Resources

- Coding Libraries
 - ConvnetJS: a toy 2D classification with 2-layer neural network. [[link](#)]
 - Python Machine Learning (3rd Edition) by Sebastian Raschka at <https://github.com/rasbt/python-machine-learning-book-3rd-edition>
- Book Chapters
 - Chapter 6.7, 6.8 Introduction to Data Mining by Kumar et al.