Studiere Stadt der Zukunft!

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

An Introduction to Neural Networks

-

Feedforward NN Backpropagation

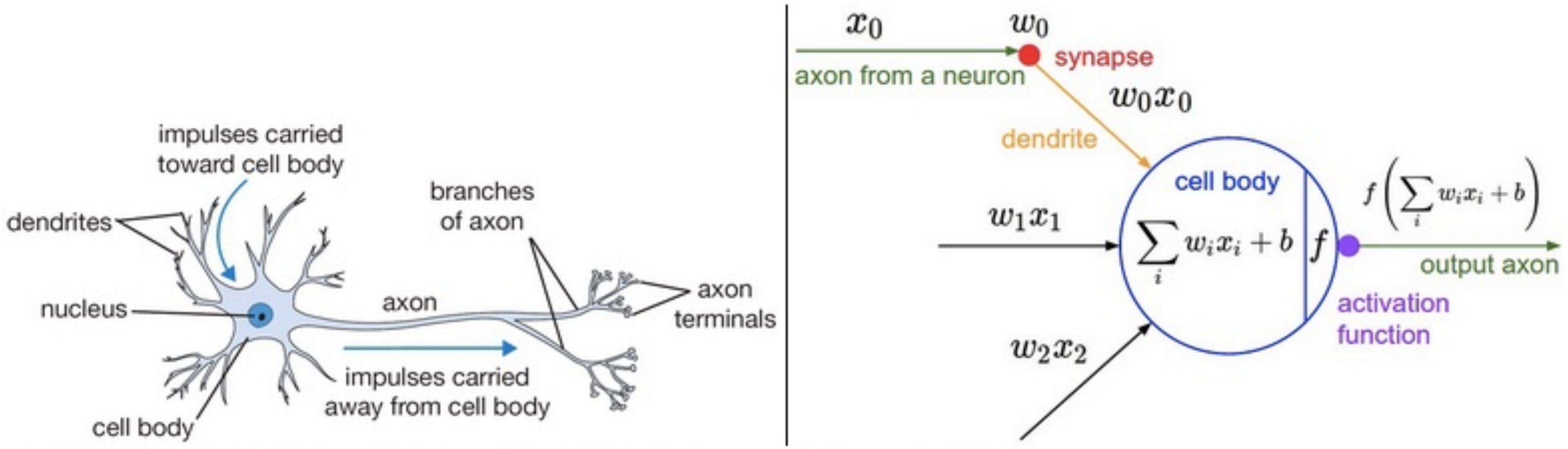
Agathe Merceron
Beuth University of Applied Sciences
Berlin, Germany

Agenda

- Artificial neuron
- Activation function
- Feedforward neural networks
- Forward calculation
- Loss function
- Backpropagation

Neuron

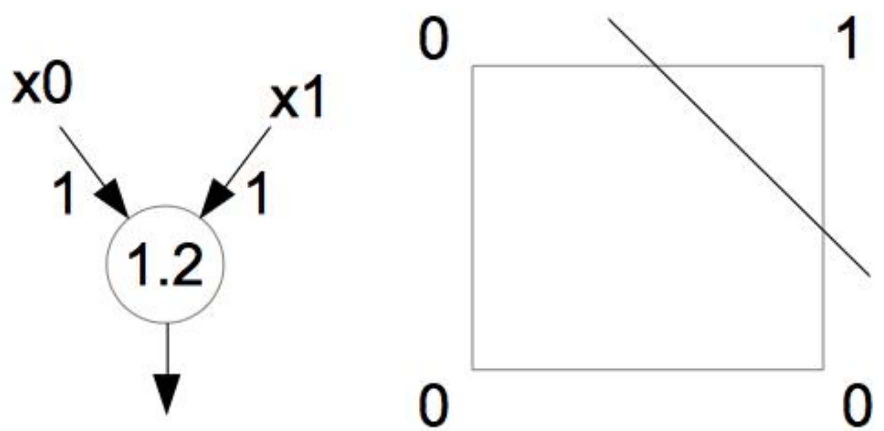
http://cs231n.github.io/neural-networks-1/



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

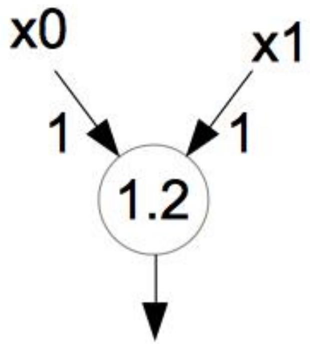
Neural networks and Boolean operators

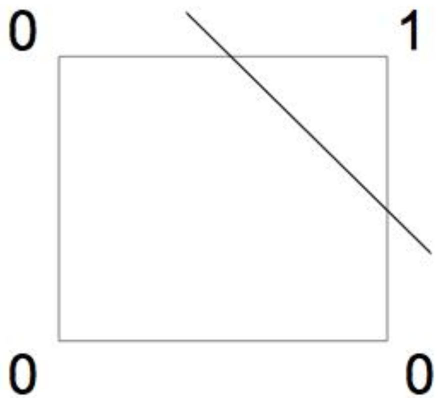
- The operator AND can be represented by a single neuron.
- Activation function: Heaviside function: 0 if the weighted sum is smaller then the number in the neuron, 1 otherwise.



Neural networks and Boolean operators

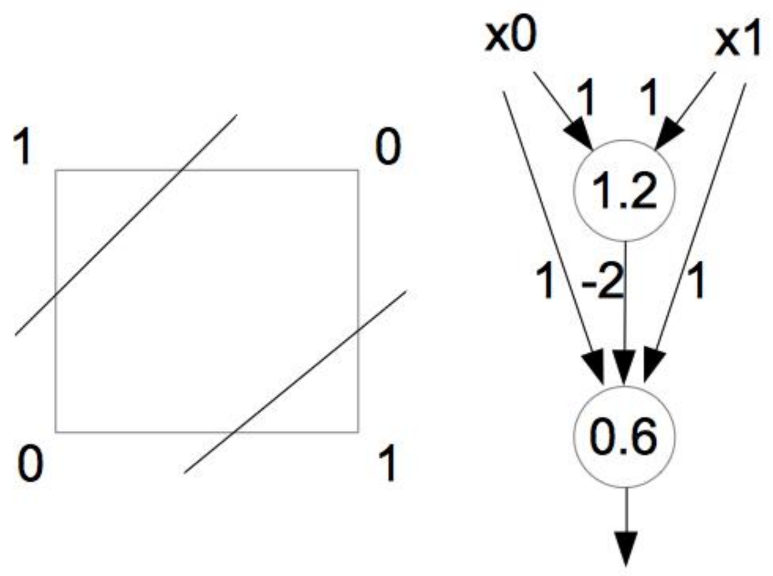
x0	x1	AND	Output
0	0	1*0+1*0 < 1.2	0
0	1	1*0+1*1 < 1.2	0
1	0	1*1+1*0 < 1.2	0
1	1	1*1+1*1 ≥ 1.2	1





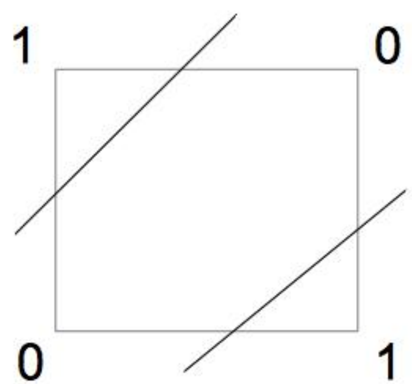
Neural networks and Boolean operators

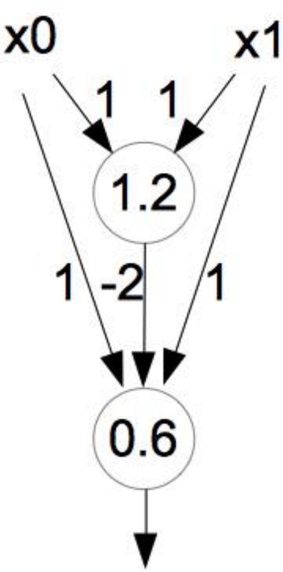
- The operator XOR cannot be represented by a single neuron. A second neuron is needed.
- Activation function: Heaviside function: 0 if the weighted sum is smaller as the number in the neuron, 1 otherwise.



Neural networks and Boolean operators

x0	x1	XOR			Output
0	0	1*0+1*0 < 1.2	0	1*0+1*0+ -2*0 < 0.6	0
0	1	1*0+1*1 < 1.2	0	1*0+1*1+ -2*0 ≥ 0.6	1
1	0	1*1+1*0 < 1.2	0	1*1+1*0+ -2*0 ≥ 0.6	1
1	1	1*1+1*1 ≥ 1.2	1	1*1+1*1+ -2*1 < 0.6	0



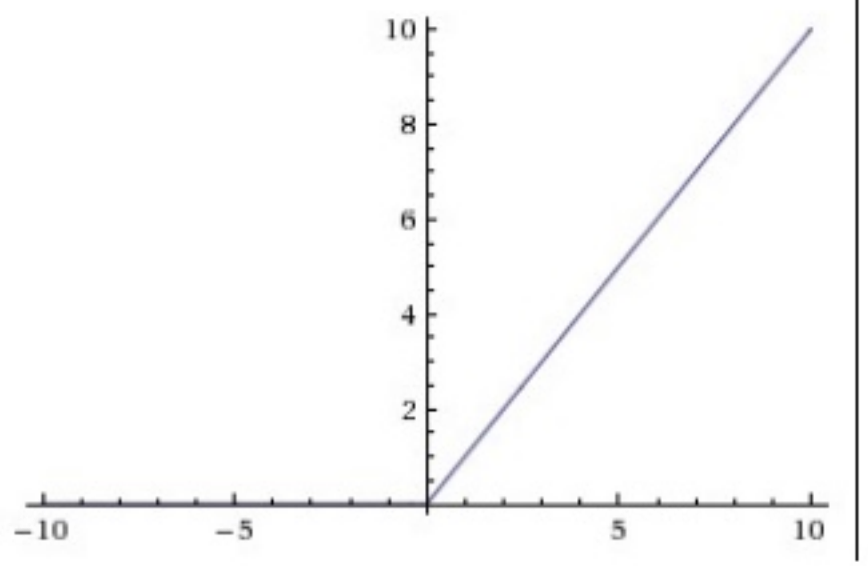


Activation functions

- Heaviside function: 1 if weighted sum of the inputs bigger than the threshold in the neuron, 0 otherwise.
- Rectified Linear Units (ReLU): f(x) = $\begin{Bmatrix} 0, if\ x \leq 0 \\ x, if\ x > 0 \end{Bmatrix}$
- Logistic sigmoid function: f(x) = $\frac{1}{1+e^{-x}}$
- tanh: f(x) = $\frac{e^{x} - e^{-x}}{e^{x}+e^{-x}}$

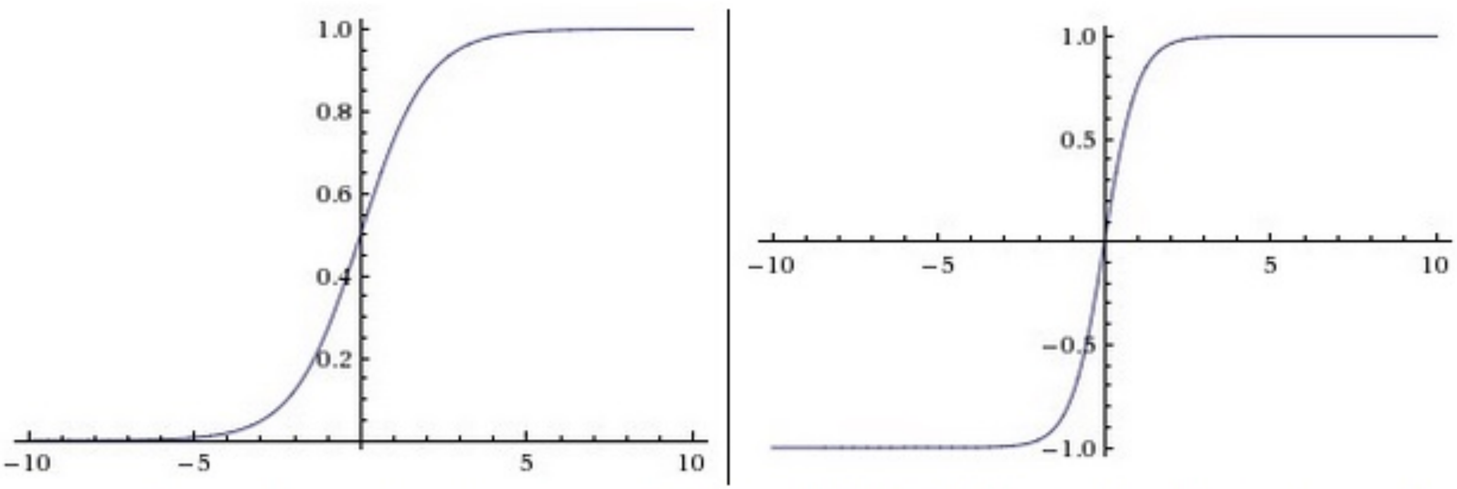
Activation functions

- Rectified Linear Units (ReLu):



https://cs231n.github.io/neural-networks-1/#classifier

Activation functions: squashing functions

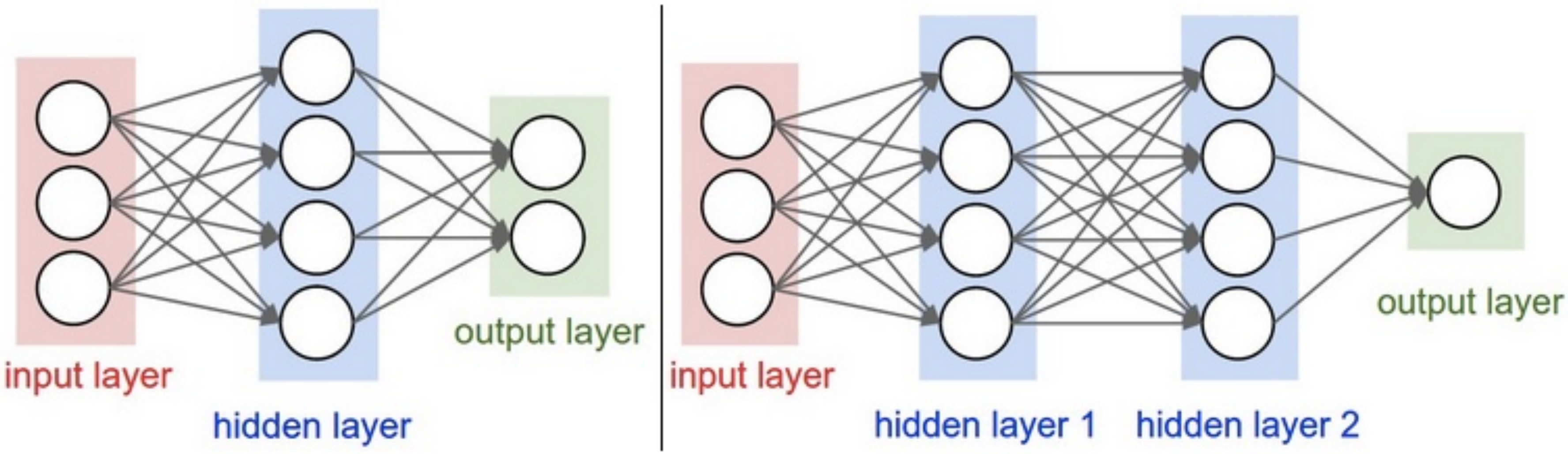


Left: Sigmoid non-linearity squashes real numbers to range between [0,1] **Right:** The tanh non-linearity squashes real numbers to range between [-1,1].

https://cs231n.github.io/neural-networks-1/#classifier

Feedforward neural networks

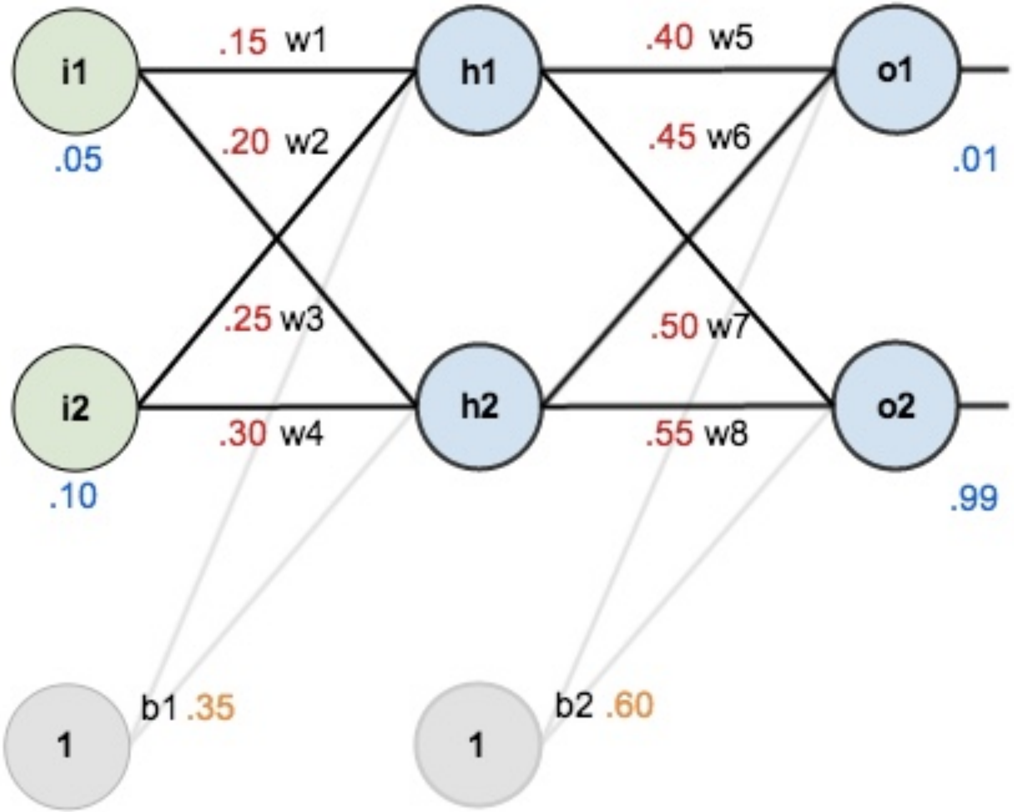
http://cs231n.github.io/neural-networks-1/



Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. **Right:** A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

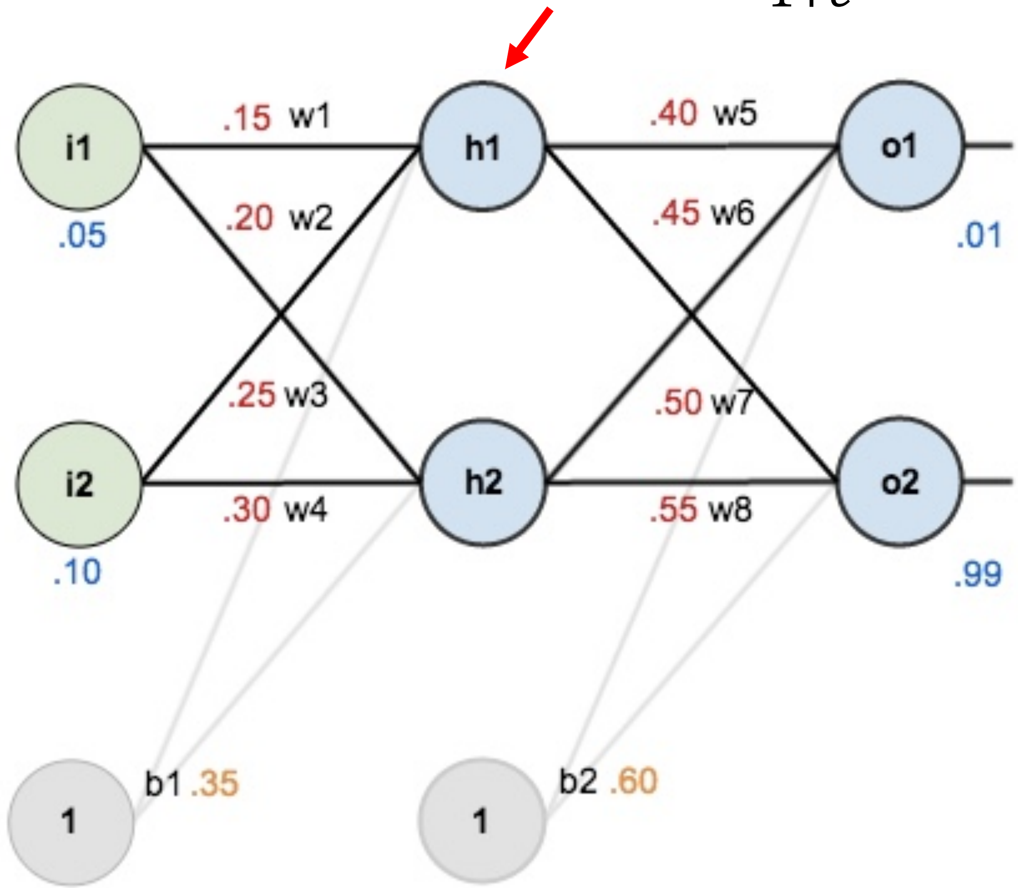
Hands-On: Forward Calculation

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



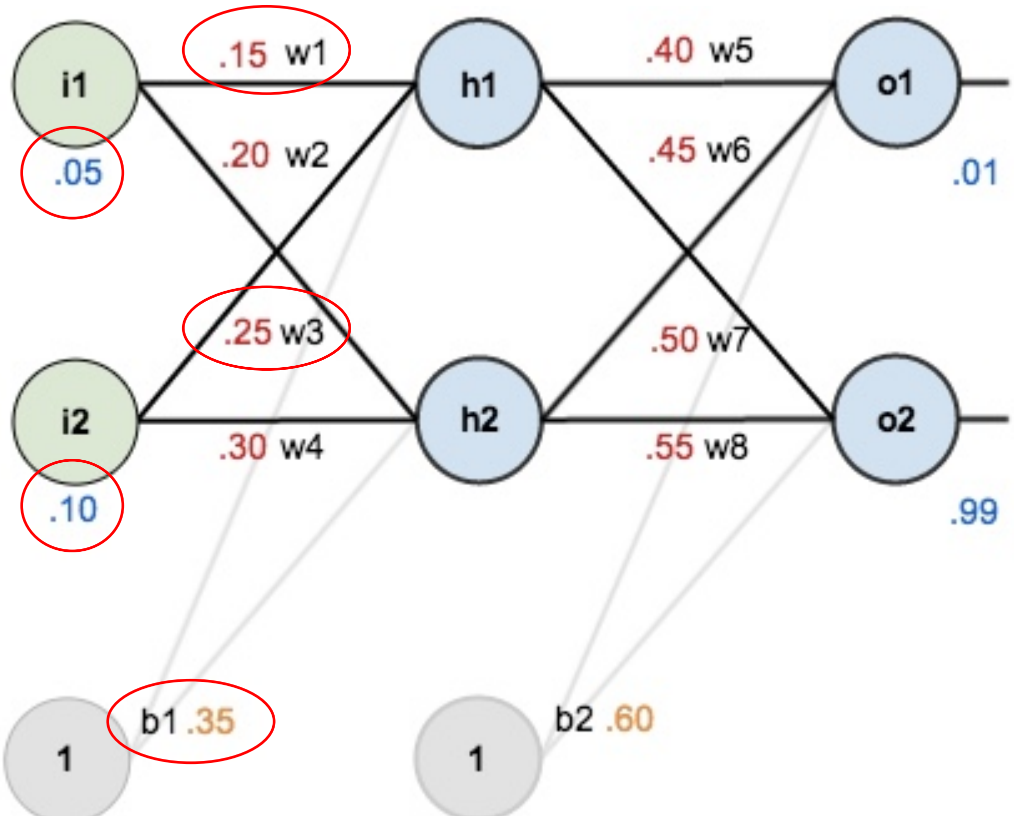
Hands-On: Forward Calculation 1

- Calculate the output of neuron h1 for the inputs (0.05, 0.1) and the sigmoid function f(x) = $\frac{1}{1+e^{-x}}$



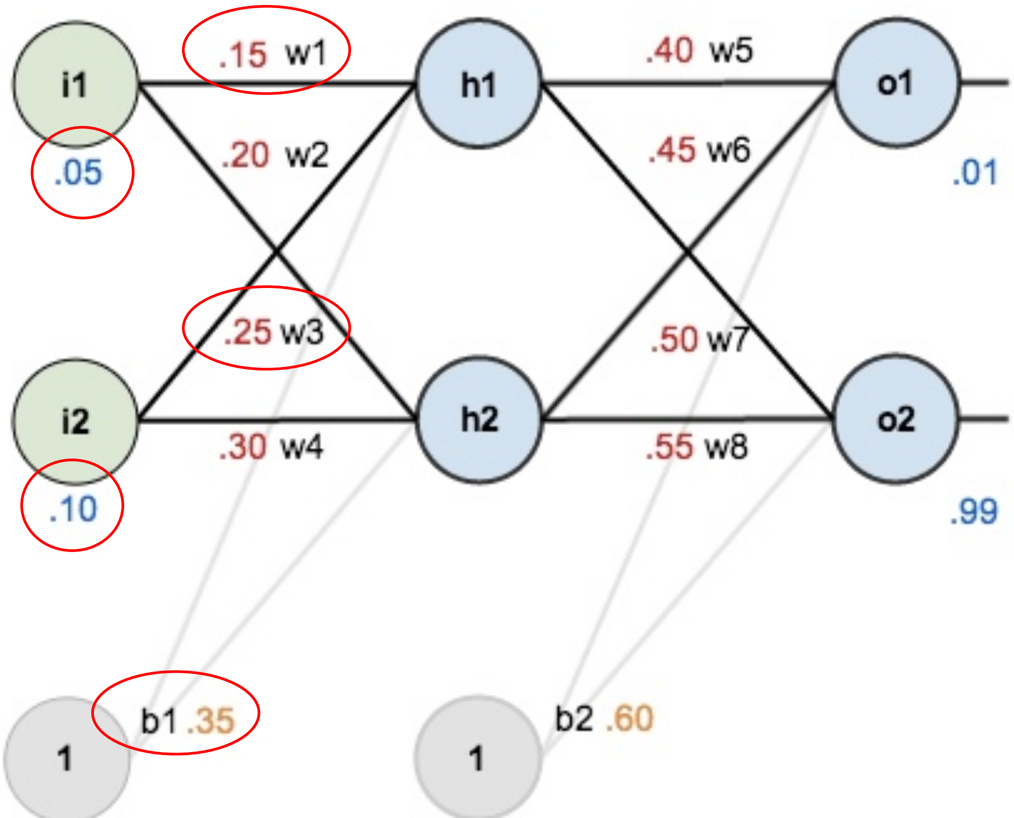
Hands-On: Forward Calculation 1

- Calculate the output of neuron h1 for the inputs (0.05, 0.1) and the sigmoid function f(x) = $\frac{1}{1+e^{-x}}$



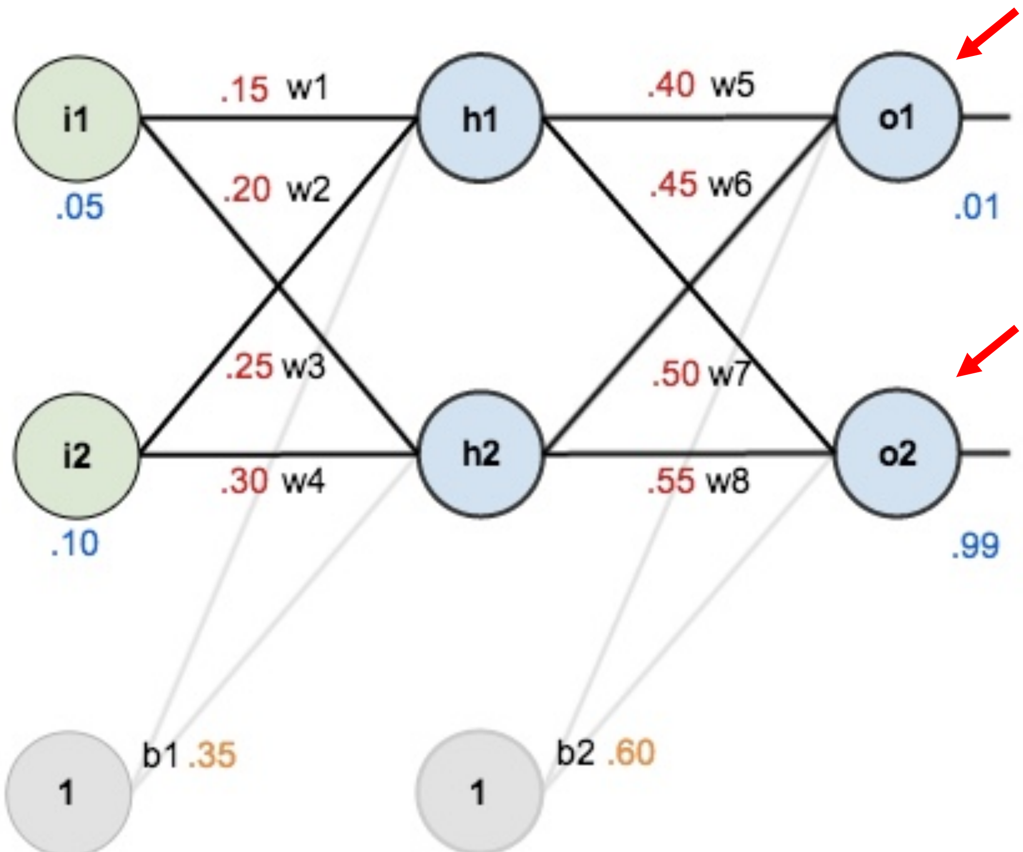
Hands-On: Forward Calculation 1

- Input h1 = 0.05*0.15 + 0.10*0.25 + 0.35 = 0.3775
- $f(x) = \frac{1}{1+e^{-0.3775}} = 0.5932$



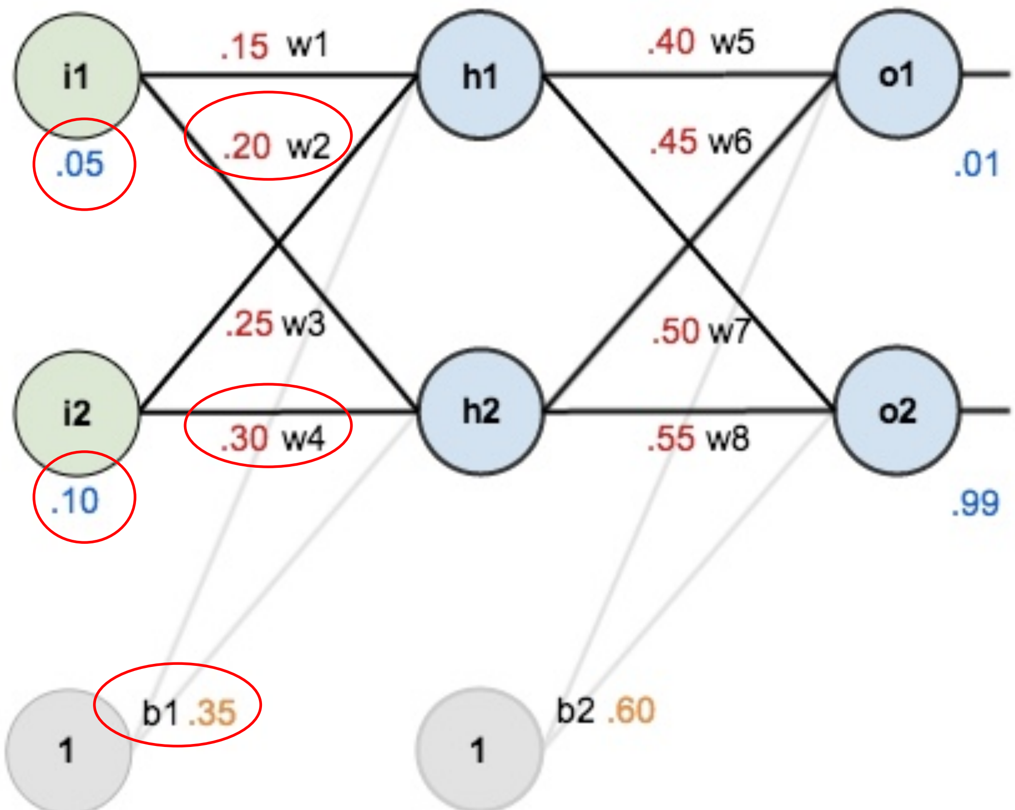
Hands-On: Forward Calculation 2

- Calculate the output of neurons o1 and o2 for the inputs (0.05, 0.1) and the sigmoid function f(x) = $\frac{1}{1+e^{-x}}$



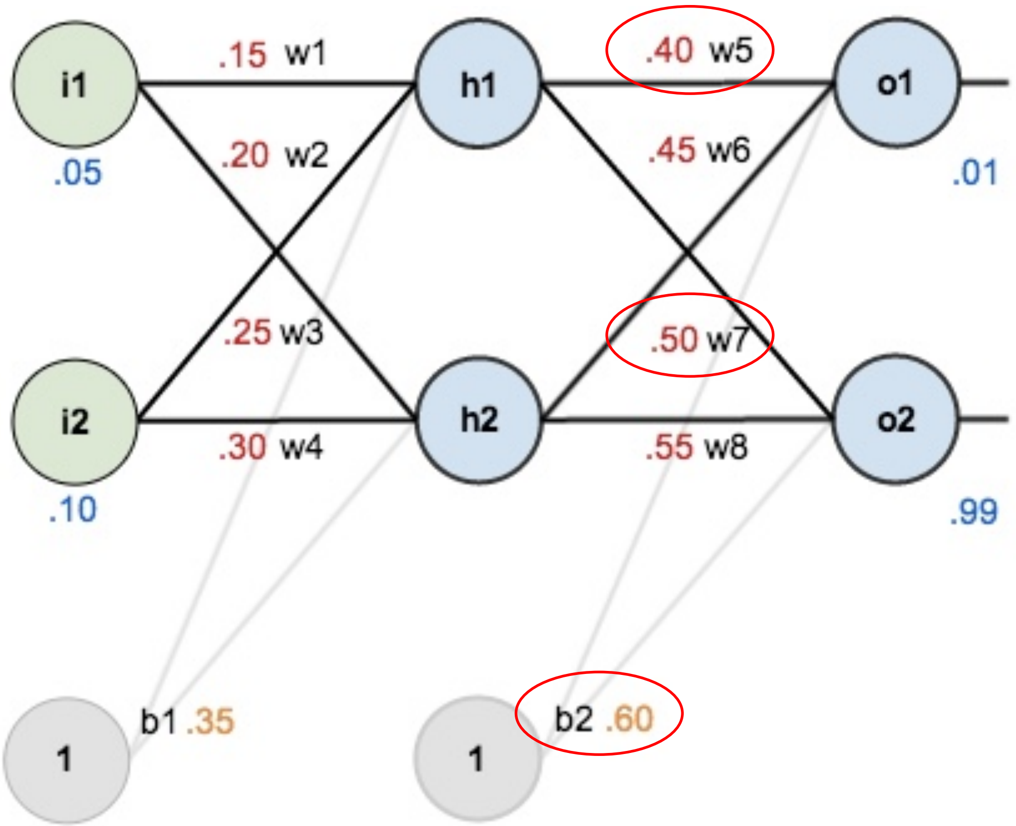
Hands-On: Forward Calculation 2

- Input h2 = 0.05*0.20 + 0.10*0.30 + 0.35 = 0.3925
- f(x) = $\frac{1}{1+e^{-0.3925}}$ = 0.5968



Hands-On: Forward Calculation 2

- Input o1 = 0.5932*0.40 + 0.5968*0.50 + 0.60 = 1.1059
- Out o1 = $\frac{1}{1+e^{-1.1059}}$ = 0.7514, Out o2 = $\frac{1}{1+e^{-1.2249}}$ = 0.7729



Universal approximation theorem

“a feedforward network with a linear output layer and at least one hidden layer with any “squashing” activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.... A neural network may also approximate any function mapping from any finite dimensional discrete space to another.“

Deep Learning; Ian Goodfellow, Yoshua Bengio, Aaaron Courville; MIT Press; 2016. P. 198

Feedforward neural networks

- Structure must be chosen:
  - Number of inputs, of hidden layers, of neurons per hidden layers, activation function, output function, loss function etc. : the hyperparameters;
  - Training costly (also in energy)

- In the training, the weights are learned (stochastic gradient descent, backpropagation algorithm)

Feedforward neural networks

- Can be fooled!
  - Experiment with 10 000 parabola and random points (5000 each):

    Class x y

    Parabola, 37.66, 1418.25

    Random, 84.65, 222.071
  - 1 hidden layer with 3 units and a bias neuron.
  - If shuffled, accuracy 95%.
  - If not shuffled and all random points first: accuracy 75%.
  - If not shuffled and all parabola points first: accuracy 50%.

Training loop [Cholet p.49]

- Draw a batch of training samples x with class T
- Run the network on x to obtain output O
- Compute the loss of the network, i.e. mismatch between O and T
- Compute the gradient of the loss
- Update the weights

- Repeat till termination condition: the errors do not change or the loss is small enough

Hands-On – Compute the loss (Mean Squared Error)

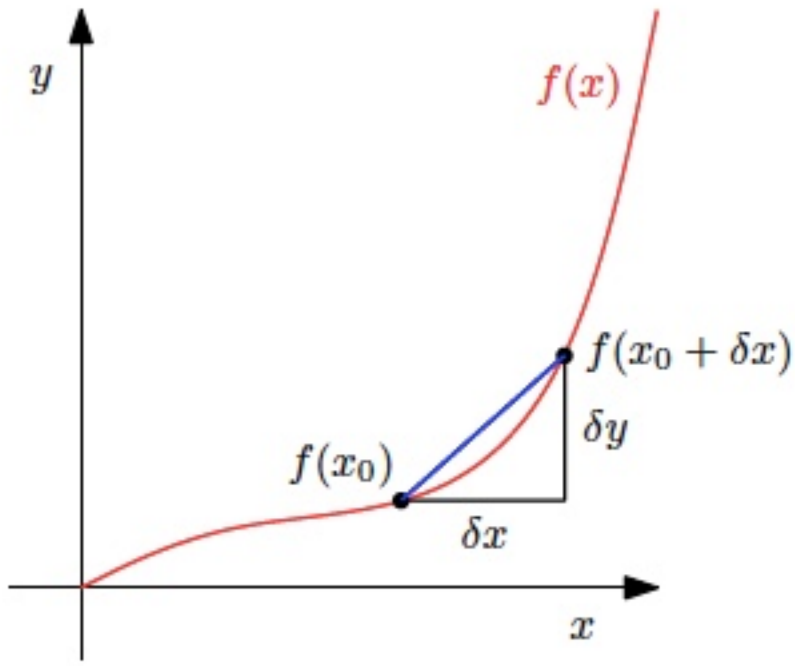
- Loss = $\sum_{i=1}^{2} \frac{1}{2} (T_i - O_i)^2$
- $T_1 = 0.01$, $T_2 = 0.99$, $O_1 = 0.7514$, $O_2 = 0.7729$
- Loss = $\frac{1}{2}(0.01 - 0.7514)^2 + \frac{1}{2}(0.99 - 0.7729)^2$
  $= 0.2984$

Gradient of the loss: Why?

- If the loss is not 0, how do we know whether we should increase a weight or decrease it?
- We need to know whether our overall function is ascending (weight should be decreased) or descending (weight should be increased).
- For a simple function f: $R \rightarrow R$, the derivative gives this information.
- For a complex function f: $R^n \rightarrow R^m$, the gradient gives this information,

Gradient of the loss: Why?

- The derivative below shows that f increases at x_0: $\frac{df}{dx} = \lim_{\partial x \to 0} \frac{f(x_0 + \partial x)}{\partial x}$



Mathematics of Machine Learning p. 141

Gradient of the loss: Why?

- For a complex function f: $R^n \rightarrow R^m$, the gradient of the loss $\nabla\, loss$ gives this information.

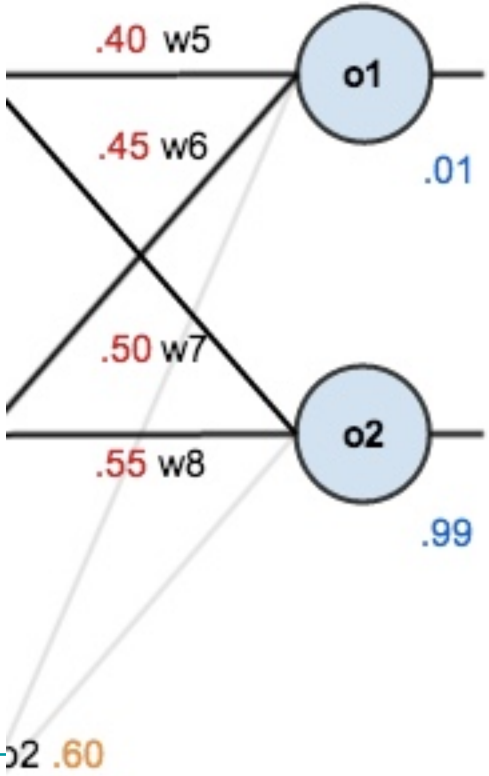
- New weights: $w' = w \; - l\, \nabla Loss$, where l is the learning rate.

Backpropagation

- Uses partial derivatives and the chain rule to calculate the change for each weight efficiently.
- Starts with the derivative of the loss function and propagates the calculations backwards.

Hands-On – Backpropagation

- How much a change in w_5 affects the loss? To know it, calculate the partial derivative of the loss with respect to w5. There is a chain of three functions:
- Loss = $\sum_{i=1}^{2} \frac{1}{2} (T_i - O_i)^2$
- $O_i = \frac{1}{1+e^{-Input\_i}}$
- $\mathrm{Input\_i} = \sum_{k=1}^{2} wk * ik + \mathrm{b2}$



Hands-On: Backpropagation

- Partial derivatives with respect to $w5$:
- Loss = $\frac{1}{2}\,(T1 - O1)^2 + \frac{1}{2}\,(T2 - O2)^2$
- $O1 = \frac{1}{1+e^{-Input\_1}}$
- $Input\_1 = w5 * out\ h1 + w6 * out\ h2 + b2$

- $\frac{\partial Loss}{\partial w5} = \frac{\partial Loss}{\partial O1} * \frac{\partial O1}{\partial Input\_1} * \frac{\partial Input\_1}{\partial w5}$

Hands-On: Backpropagation

- Loss $= \frac{1}{2}(T1 - O1)^2 + \frac{1}{2}(T2 - O2)^2$
- $\frac{\partial Loss}{\partial O1} = \frac{1}{2} * 2(T1 - O1) *$ -1 = -(T1 $- O1$)= 0.7414
- T1 : 0.01 and $O1$: 0.7514

Hands-On: Backpropagation

- $O1 = \frac{1}{1+e^{-Input\_1}}$
- $\frac{\partial O1}{\partial Input\_1} = O1(1 - O1) = 0.7514\ (1 - 0.7514) = 0.1868$

Hands-On: Backpropagation

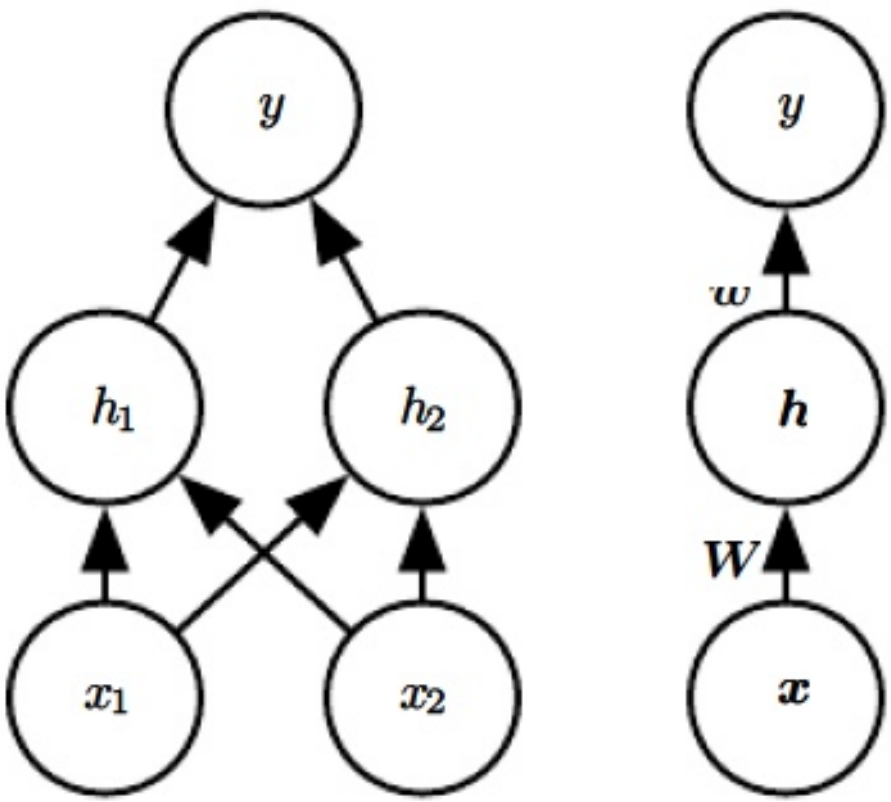
- $Input\_1 = w5 \ * out\ h1 + w6 \ * out\ h2 + b2$
- $\frac{\partial Input\_1}{\partial w5} = out\ h1 = 0.5932$

Hands-On: Backpropagation

- $\frac{\partial Loss}{\partial w5} = \frac{\partial Loss}{\partial O1} * \frac{\partial O1}{\partial Input_1} * \frac{\partial Input\_1}{\partial w5}$
- $\frac{\partial Loss}{\partial w5} = 0.7414 * 0.1816 * 0.5932 = 0.0821$
- $w5' = w5 - l * 0.0821 = 0.4 - 0.5 * 0.0821 = 0.3589$
- With 0.5 as learing rate.

Feedforward neural networks

- Compact graphical representation: W is the weights-matrix. Deep Learning; Ian Goodfellow, Yoshua Bengio, Aaaron Courville; MIT Press; 2016. P. 174



Feedforward neural networks

- Compact graphical representation: W is the weights-matrix.
- *h = g(Wx) h: neurons in the hidden layer, x : input, g: activation function.*
- *Our example W x*

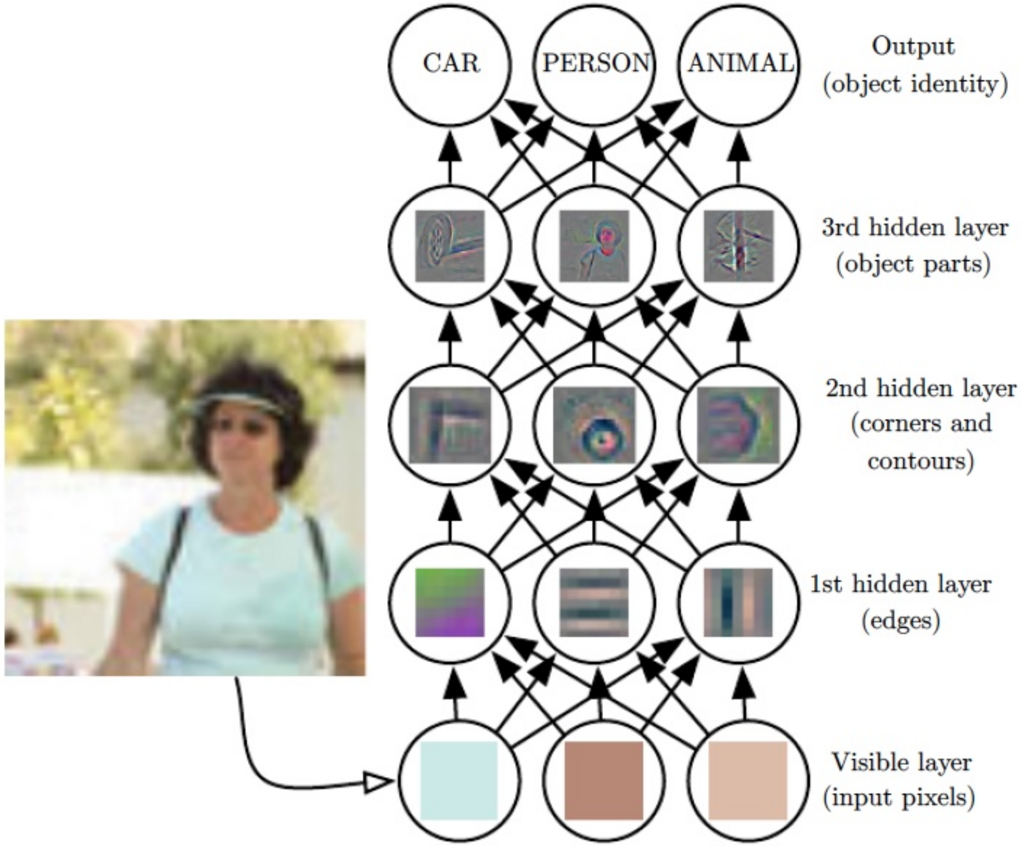
$$\begin{matrix} 0.15 & 0.25 & 0.35 \\ 0.2 & 0.3 & 0.35 \end{matrix} \; . \; \begin{matrix} 0.05 & 0.1 & 1 \end{matrix}$$

Neural networks and deep learning

- Well-known types of NN:
  - Convolutional Neural Networks (CNN) – reduce fully connectedness through the use of a convolutional operator.
  - Long Short Term Memory (LSTM) neural networks – topology is recurrent.
- Hidden layers extract increasingly abstract features from the data

Neural networks and deep learning

- Hidden layers extract increasingly abstract features from the data – Deep Learning p. 6



References

- François Chollet. Deep Learning with Python. Manning 2018.
- Marc Peter Deisenroth, A. Aldo Faisal, Cheng Soon Ong. The Mathematics of Machine Learning. https://mml-book.github.io/
- Ian Goodfellow, Yoshua Bengio, Aaaron Courville. Deep Learning. MIT Press; 2016.

Questions?

- Thank you for your attention!