Introdução a Deep Learning

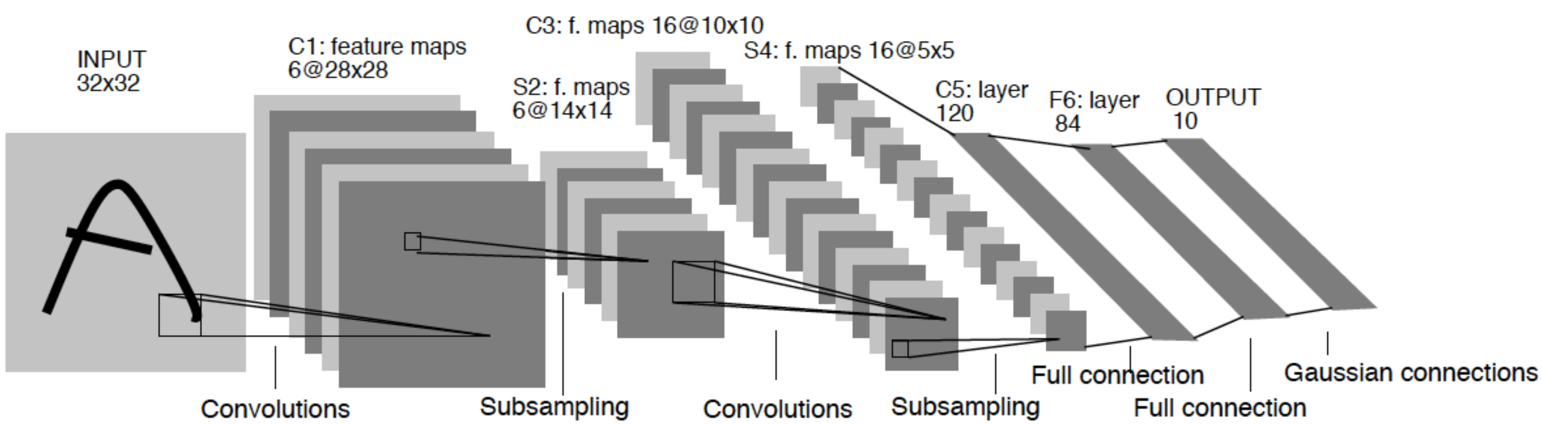
Luciano Barbosa

Centro de Informática UFPE

Universidade Federal de Pernambuco

What is Deep Learning?

- Sub-area of Machine Learning
- Not really something new
- Handwriting recognition paper (1998)



Img-Source: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

What is new?

- New training strategies: able to train deep networks
  - Hinton et al. (2006) presented a new, stacked training method which applies the concept of pre-training
- More data has been available
- Computer infrastructure (hardware and software) has improved- > bigger models

Supervised Learning

- Goal: to infer a function from examples to predict classes on new examples
  - Example of classification tasks: sentiment analysis, image detection, text categorization etc
- Two steps:
  - Training: learn the function
  - Test: apply the function on new examples

Supervised Learning

- Training set: instances and labels

Categorical

Training set

	viagra	learning	the	dating	nigeria	*spam*?
$\vec{x}_1 = ($	1	0	1	0	0)	$y_1 = 1$
$\vec{x}_2 = ($	0	1	1	0	0)	$y_2 = -1$
$\vec{x}_3 = ($	0	0	0	0	1)	$y_3 = 1$

- Instance represented by its features' vector: x
- Goal: to learn f(x)=y that better predicts y given x
- Label y
  - Categorical -> classification
  - Numerical -> regression

Features

- Very important for good classification results
- Good feature: high correlation with the classification outcome
  - Ex_1: weather prediction: temperature, humidity
  - Ex_2: sentiment analysis: words with polarity (positive/negative)
- Usually human-generated
- The training task becomes optimizing weights of the features for prediction

	viagra	learning	the	dating	nigeria	$spam?$
$\vec{x}_1 = ($	1	0	1	0	0)	$y_1 = 1$
$\vec{x}_2 = ($	0	1	1	0	0)	$y_2 = -1$
$\vec{x}_3 = ($	0	0	0	0	1)	$y_3 = 1$

Traditional ML vs. Deep Learning

Traditional Machine Learning

Feature Engineering: Describe your data with features a computer can understand	Machine Learning: Some hyper-parameter tuning

Deep Learning Approach

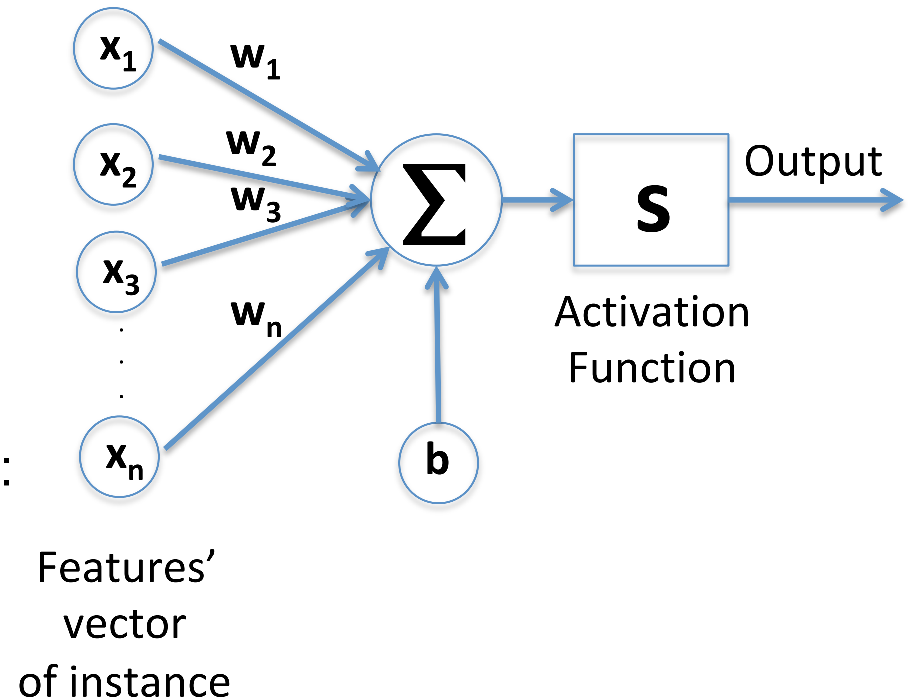
Getting Domain Expertise	Design / select a suitable network architecture	Optimize architecture & fine-tune parameters

Deep Learning

- Automatically learns good features or representations (instead of feature engineering)
- Multiple levels of representation from raw data
- Usually needs large amounts of data
  - Not necessarily labeled
- For small datasets: hand-designed features can still be included

Basics: Artificial Neuron

- Given **inputs**: $x_1, x_2, x_3, \ldots \in \mathbb{R}$
  and **weights**: $w_1, w_2, w_3, \ldots \in \mathbb{R}$
  and a **bias** value: $b \in \mathbb{R}$



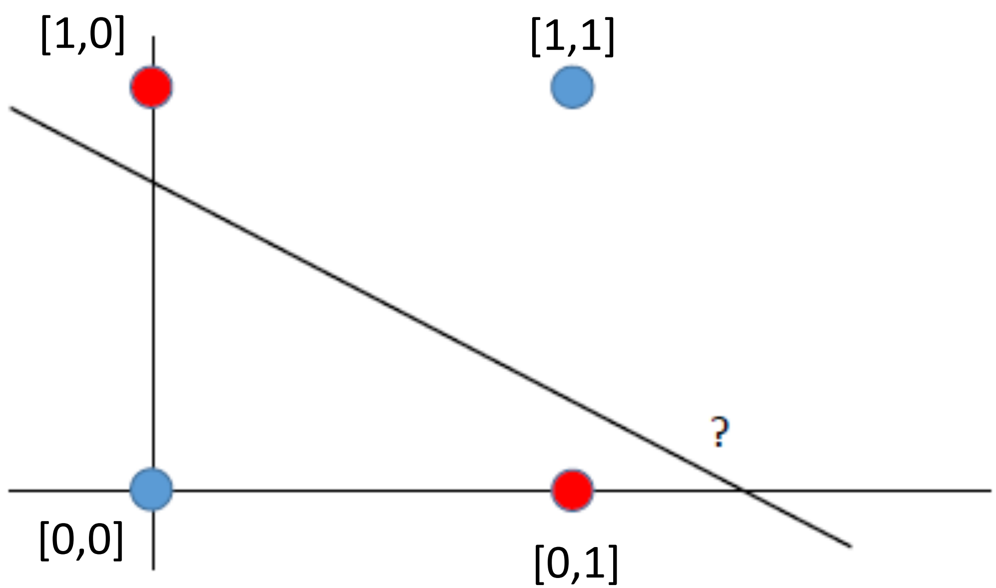
- A neuron produces a single output:

$$o_1 = s\left(\textstyle\sum_i w_i x_i + b\right)$$

- Function *s*: **activation function**
- The weights and bias values
  - Initialized randomly
  - Learned during training

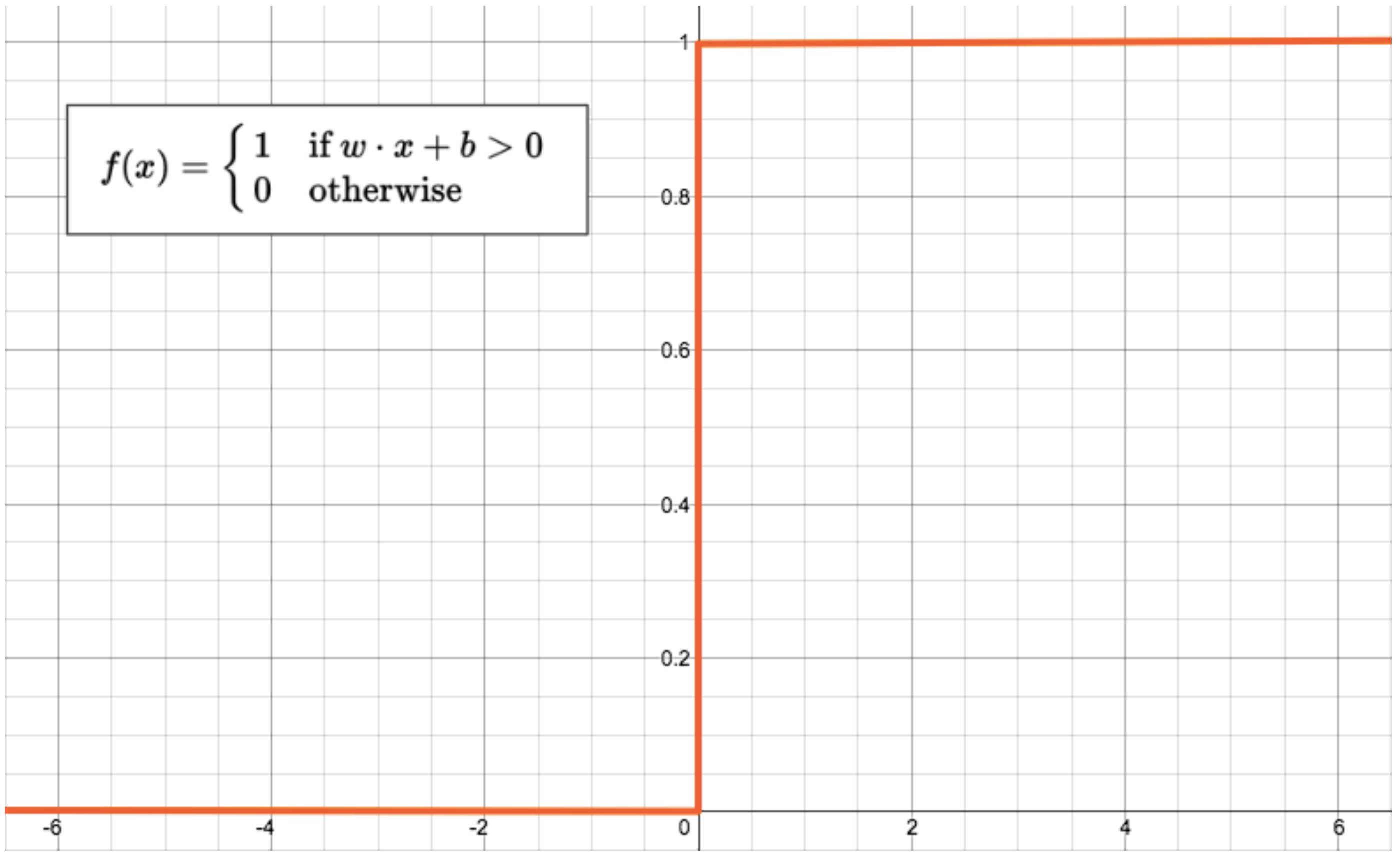
Activation Function & Non-linearity

- Motivation: linear classifier can not solve some problems
- Example: learning the XOR function
- Goal of activation function: to add non-linearity



Typical Activation Function

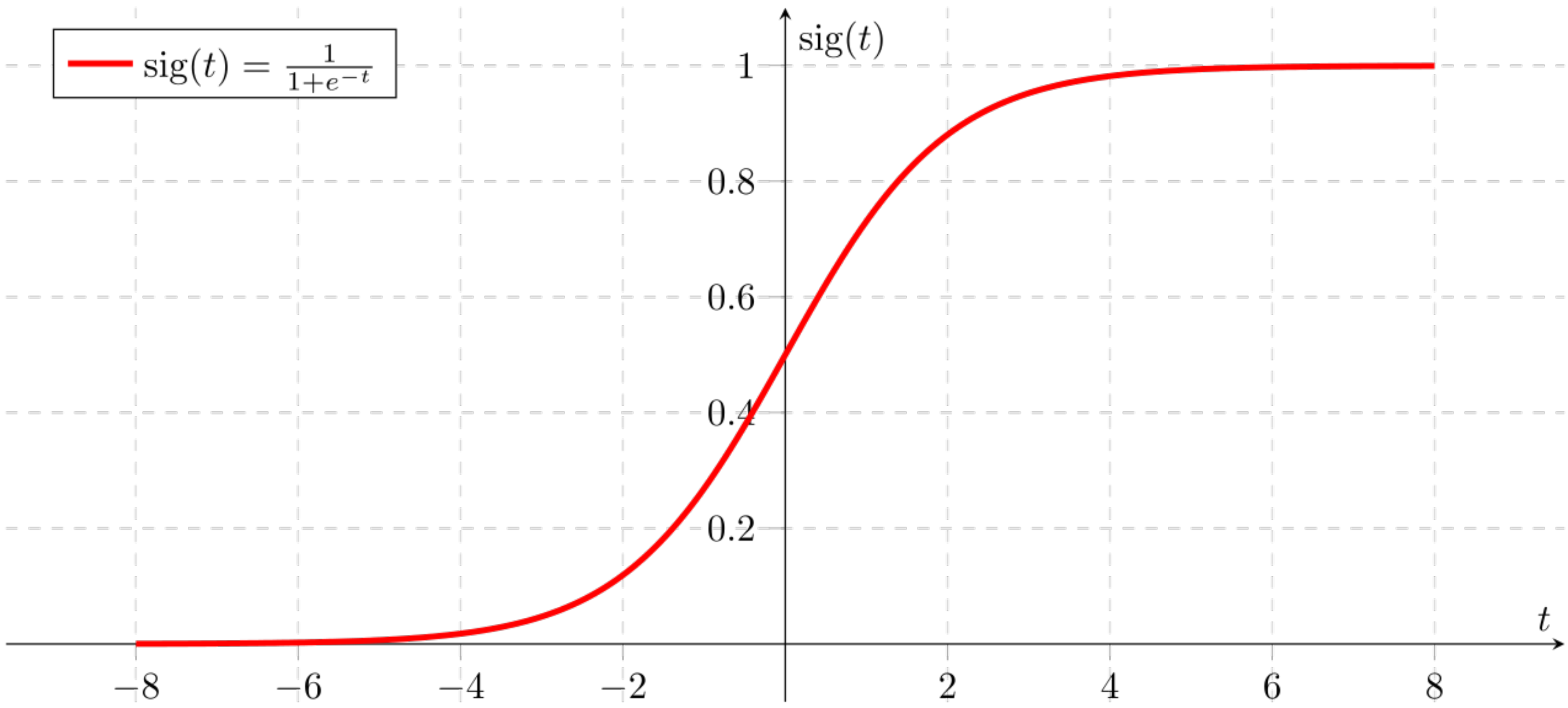
- Step function scales between 0 and 1



Img-Source: https://appliedgo.net/perceptron/

Typical Activation Function

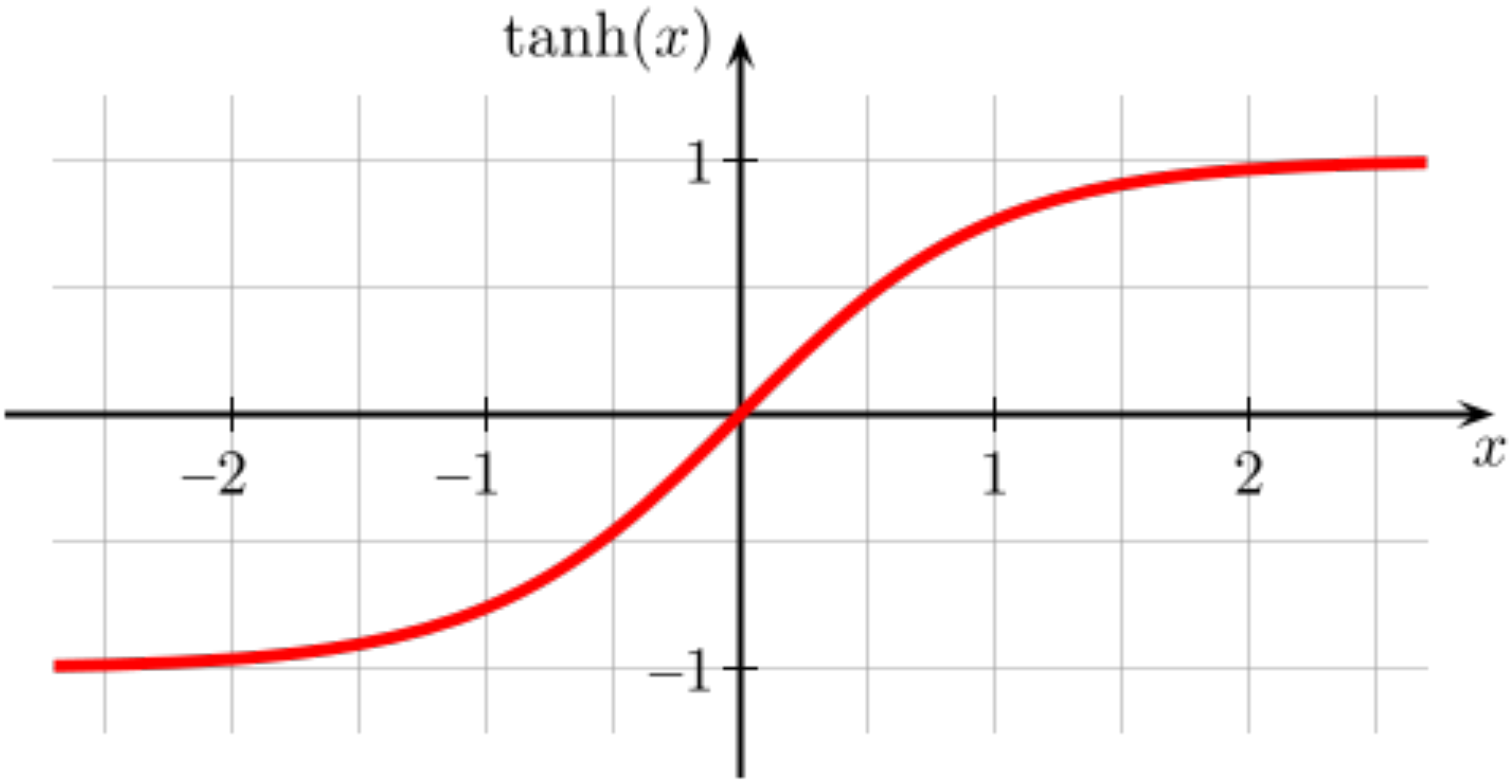
- Sigmoid Function scales between 0 and 1



Img-Source: Wikipedia

Typical Activation Function

- *Hyperbolic tangent* scales between -1 and 1



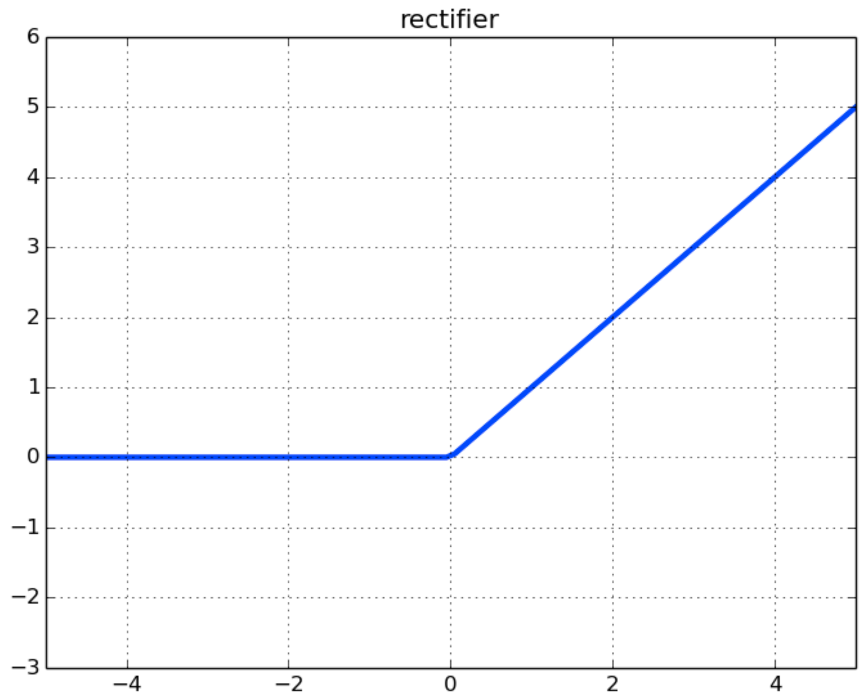
Img-Source: Wikipedia

Typical Activation Function

- Rectifier:

$$f(x) = \max(0, x)$$

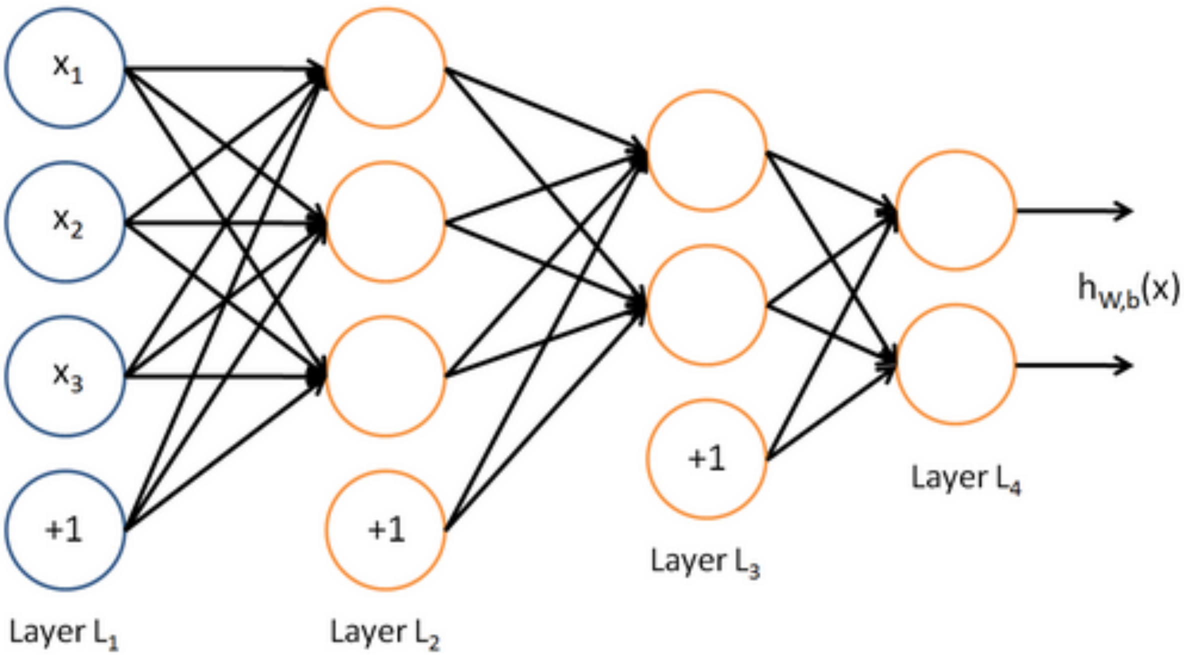
- A unit using the rectifier function is called rectified linear unit (ReLU)



Img-Source: http://datascience.stackexchange.com/questions/5706/what-is-the-dying-relu-problem-in-neural-networks

Feed Forward Neural Networks

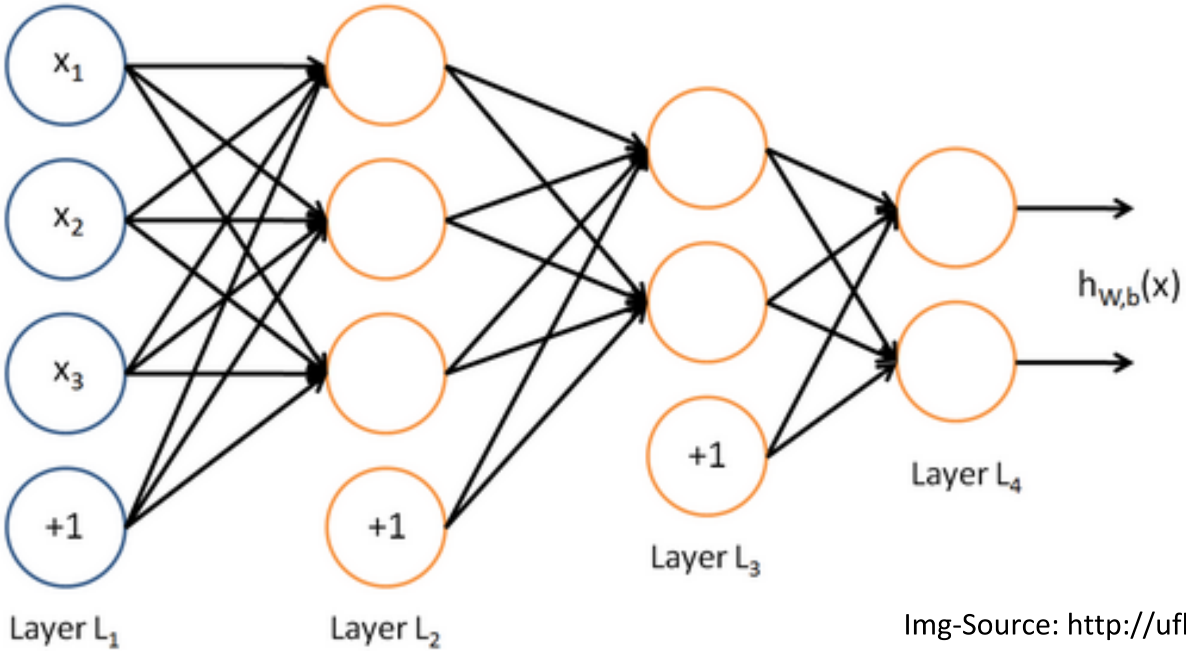
- Layers of neurons
- Information flows forward
- First layer: input data
- Last layer: output of the model



Img-Source: http://ufldl.stanford.edu/wiki/

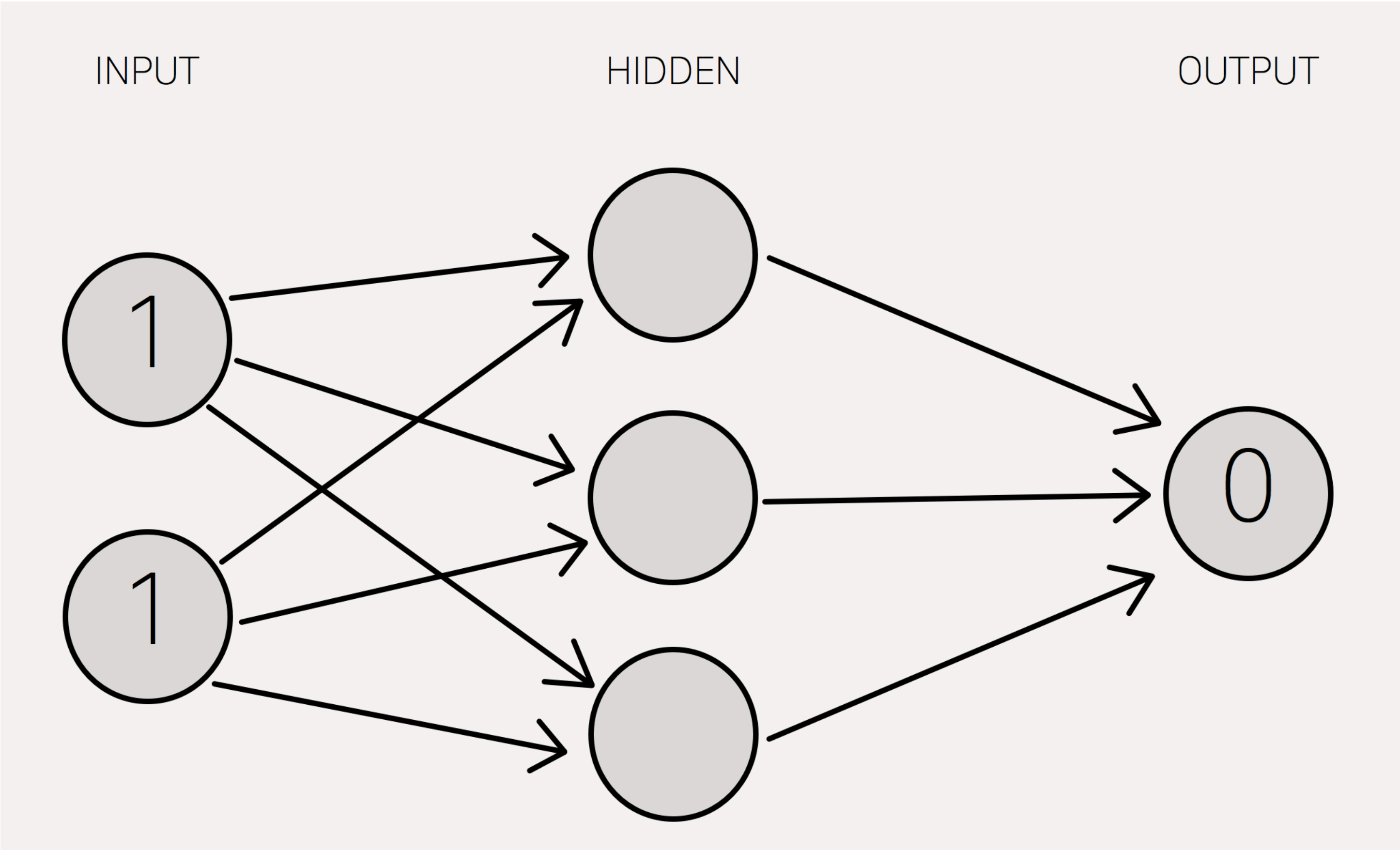
Feed Forward Neural Networks

- The hidden layers (L_2, L_3) represent learned non-linear combination of input data
  – Project the input to a (usually) low dimensional space



Img-Source: http://ufldl.stanford.edu/wiki/

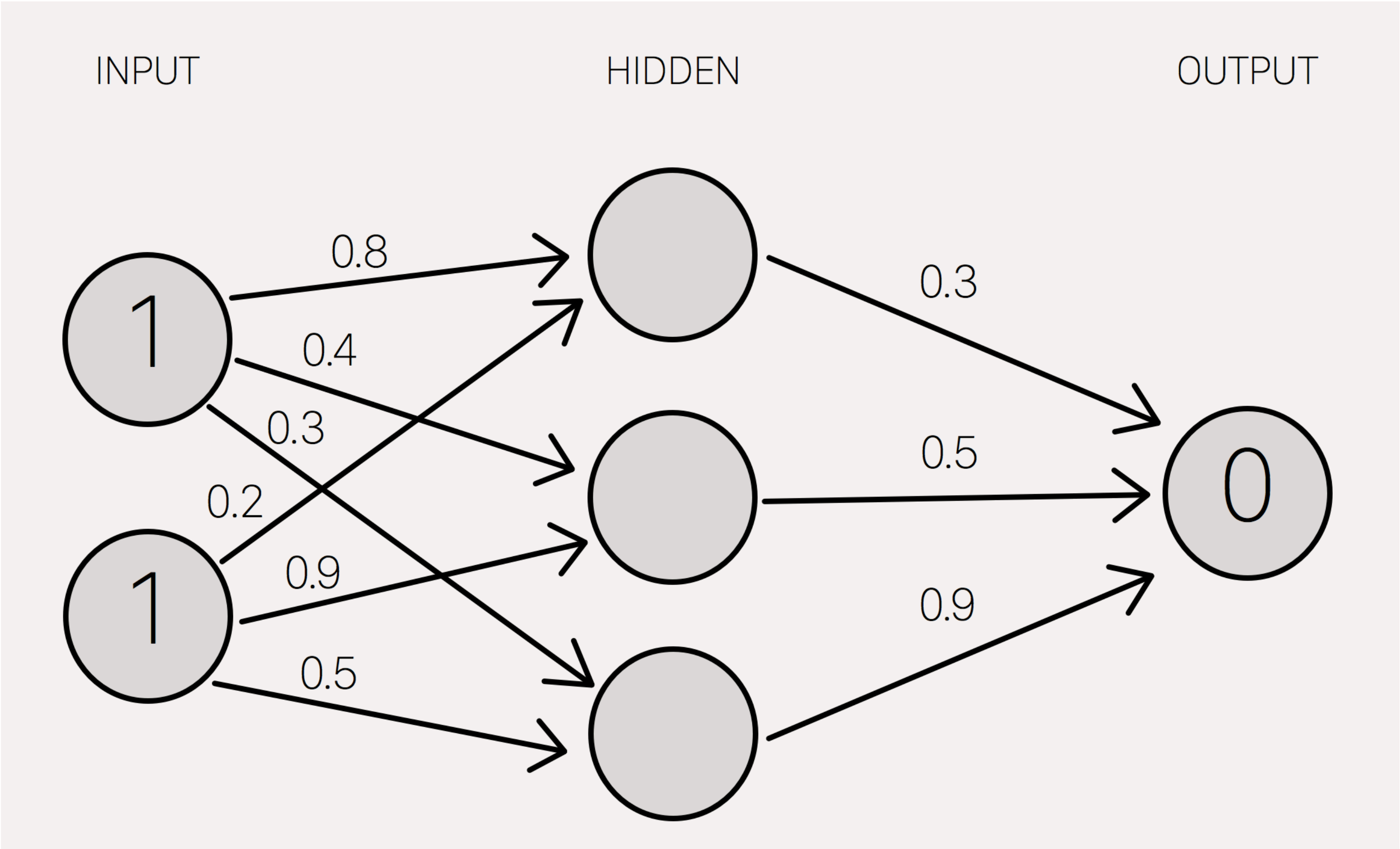
Computation: XOR Example



Img-Source: http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Computation: XOR Example

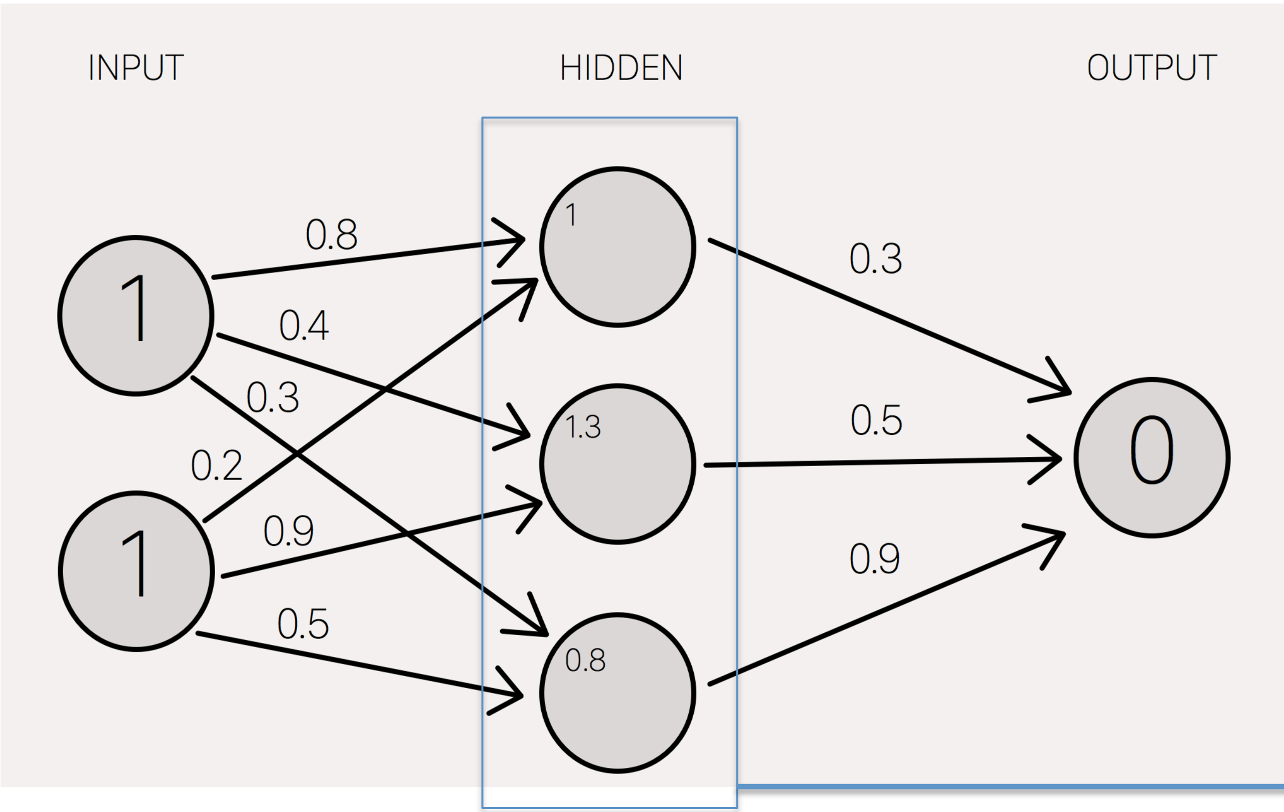
- Assigning initial weights



Img-Source: http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Computation: XOR Example

- Summing weights in the hidden layer



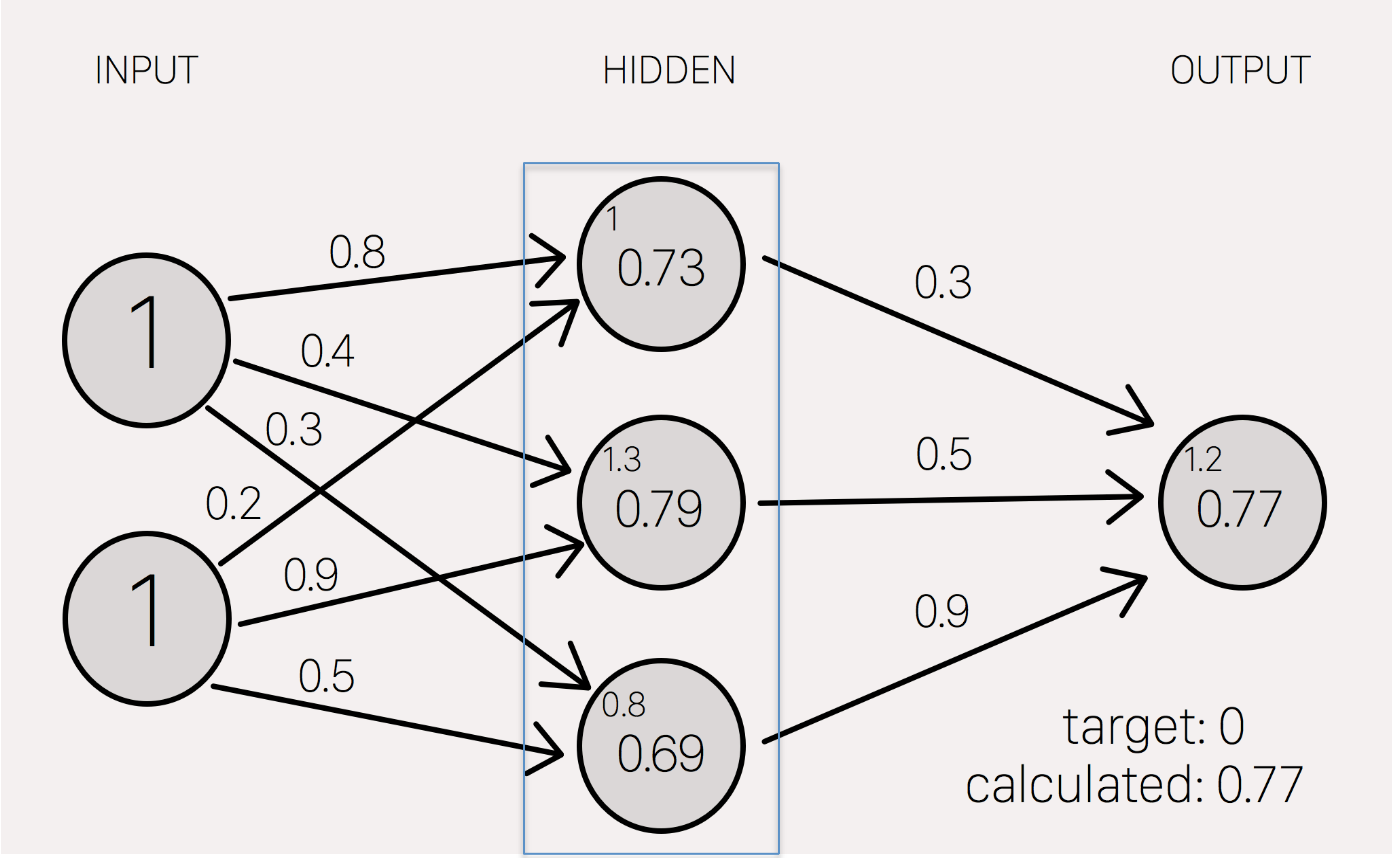
Matrix Operation

$$\underset{\text{Input}}{[1,1]}*\underset{W_H}{\begin{bmatrix}0.8 & 0.4 & 0.3\\ 0.2 & 0.9 & 0.5\end{bmatrix}}$$

Img-Source: http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Computation: XOR Example

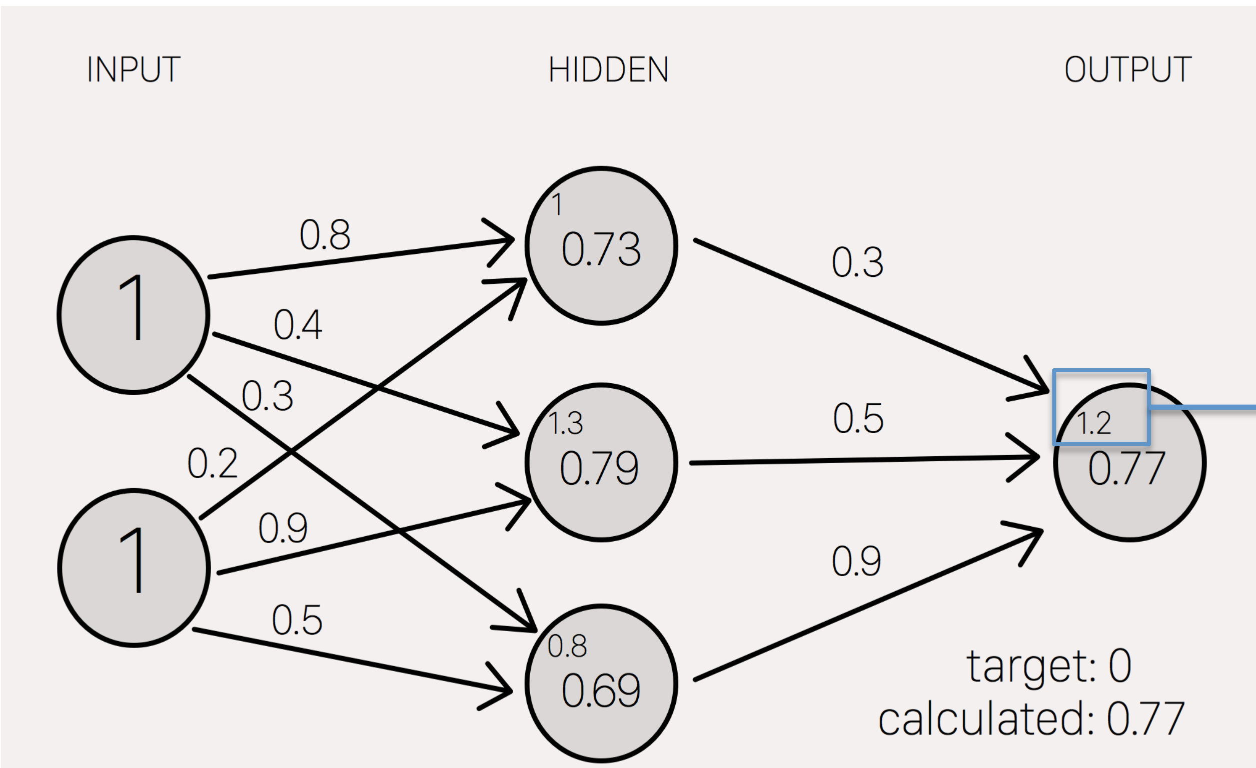
- Applying the activation function in the hidden layer



Img-Source: http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Computation: XOR Example

- Summing the weights and applying the activation function in the output layer



Matrix Operation

$$\underbrace{[0.73, 0.79, 0.69]}_{I_o} * \underbrace{\begin{bmatrix} 0.3 \\ 0.5 \\ 0.9 \end{bmatrix}}_{W_o}$$

Img-Source: http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Output Layer: Softmax-Classifier

- Multi-class classification
- Given *K* classes (*K* = number of output units), compute the activation *z* for the last layer:

$$z = W_3 l_3 + b_3 \in \mathbb{R}^K$$

Compute the final output *y:*

$$y_j = \mathrm{softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

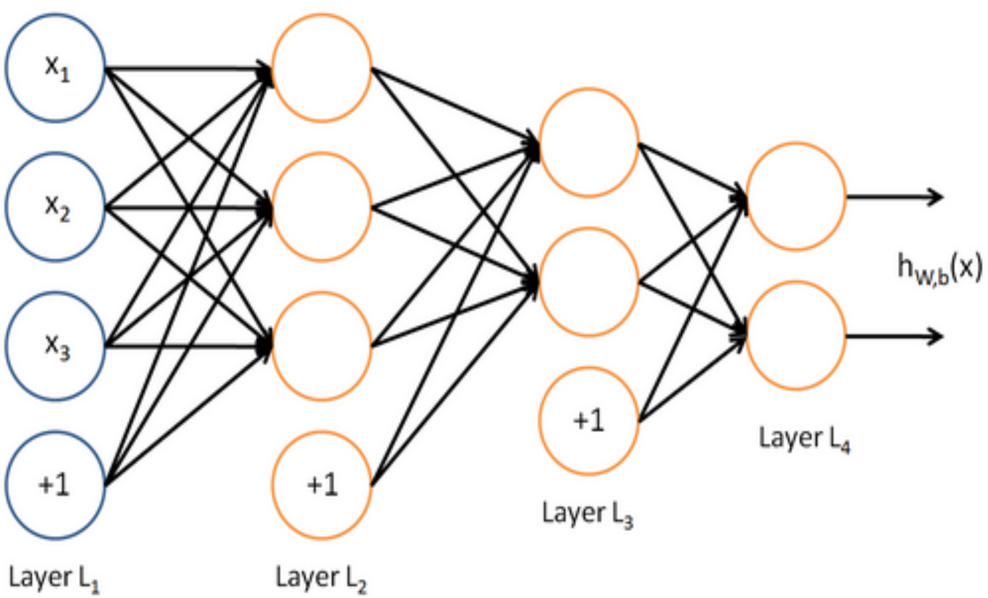
y_j can have values between 0 and 1

y_j sums up to 1 -> it can be interpreted as probability distribution

Neural Network as One Long Function

- Long function of vector and matrix operations

$$output = \text{softmax}(b_3 + W_3 \tanh(b_2 + W_2 \tanh(b_1 + W_1 x)))$$

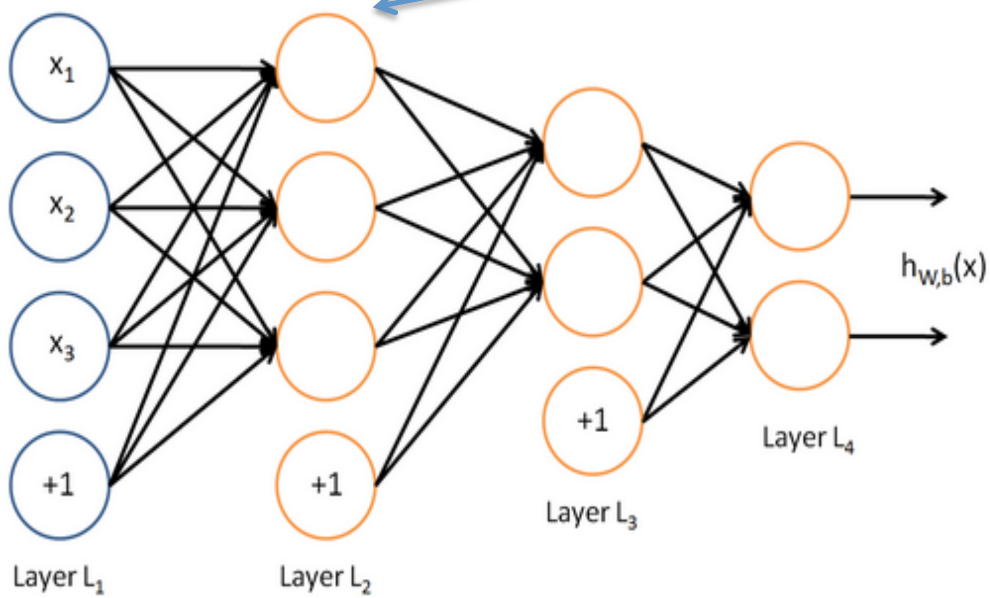


Img-Source: http://ufldl.stanford.edu/wiki/

Neural Network as One Long Function

- Long function of vector and matrix operations

$$output = \mathrm{softmax}(b_3 + W_3 \tanh(b_2 + W_2 \tanh(b_1 + W_1 x)))$$

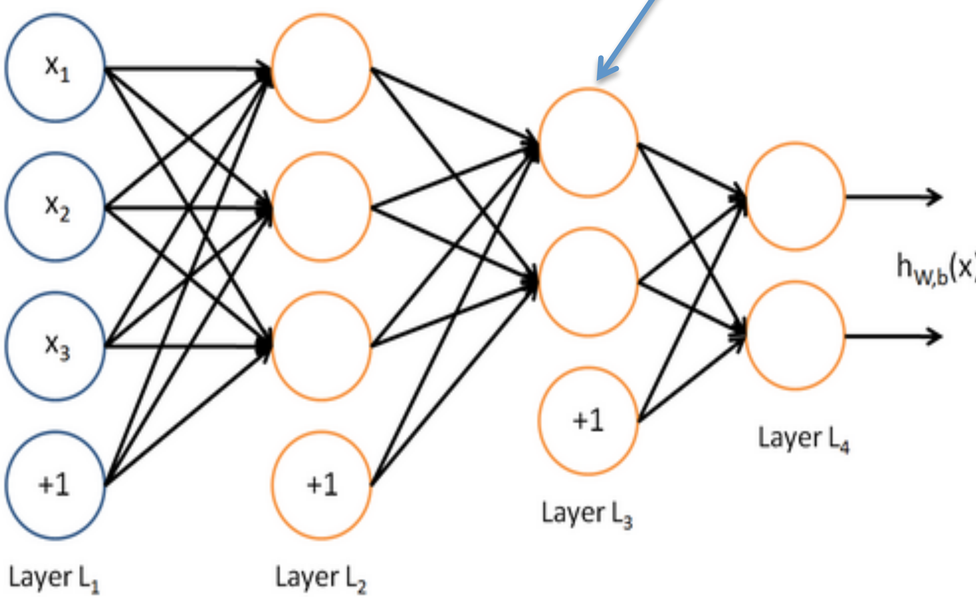


Img-Source: http://ufldl.stanford.edu/wiki/

Neural Network as One Long Function

- Long function of vector and matrix operations

$$output = \text{softmax}(b_3 + W_3 \underbrace{\tanh(b_2 + W_2 \tanh(b_1 + W_1 x))})$$

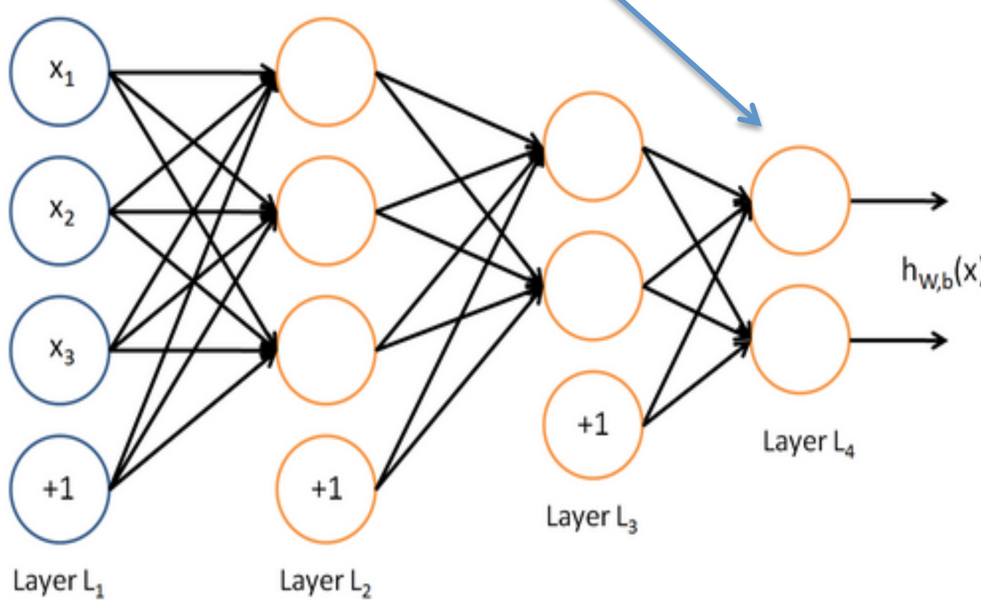


Img-Source: http://ufldl.stanford.edu/wiki/

Neural Network as One Long Function

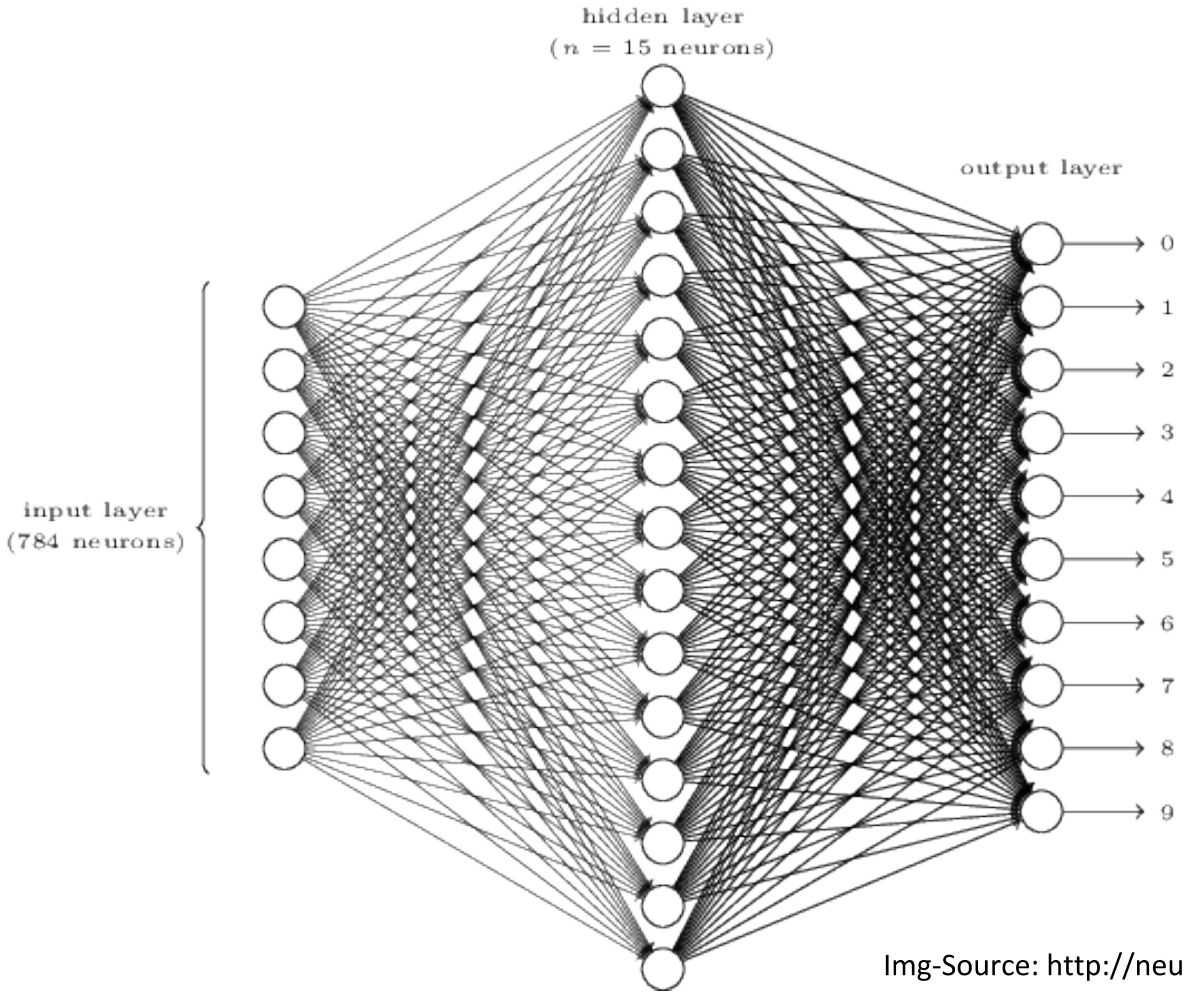
- Long function of vector and matrix operations
- Compose linear combinations and activation functions

$$output = \mathrm{softmax}(b_3 + W_3 \tanh(b_2 + W_2 \tanh(b_1 + W_1 x)))$$



Img-Source: http://ufldl.stanford.edu/wiki/

Feed-Forward Network for Handwritten Digit Recognition

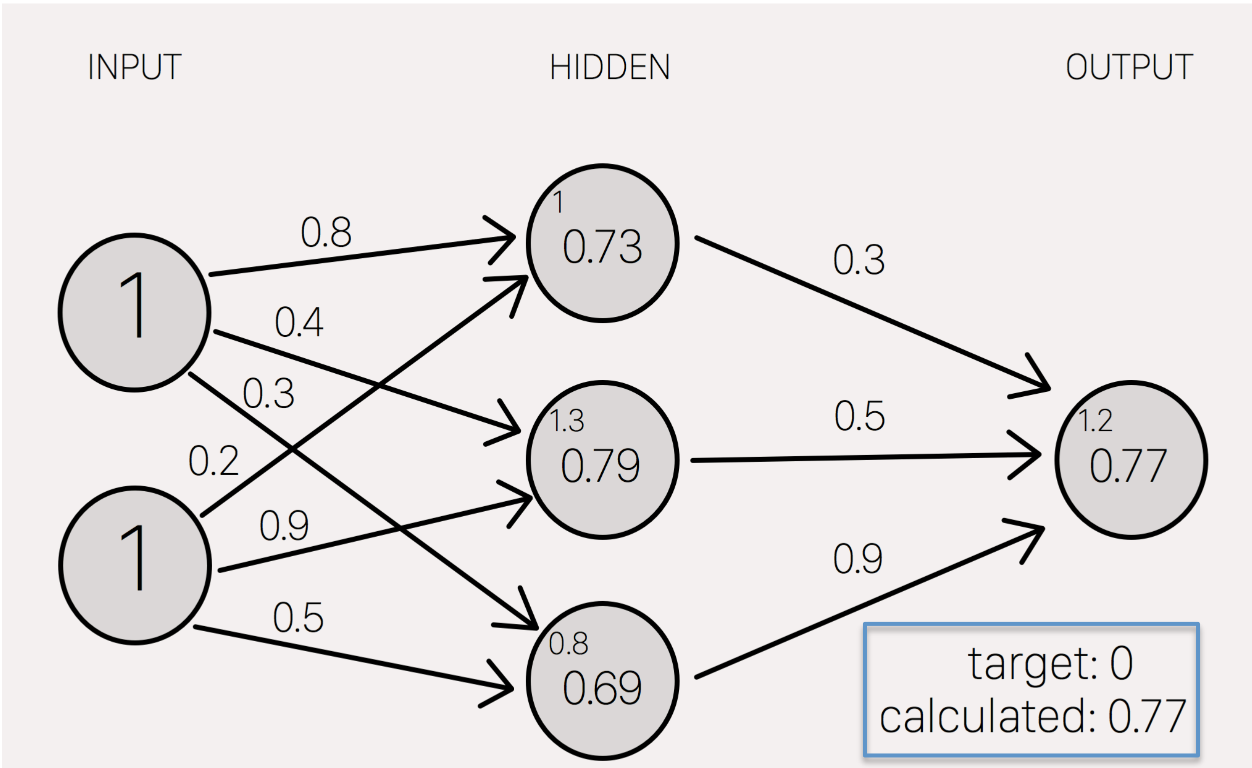


Img-Source: http://neuralnetworksanddeeplearning.com

Hidden Layers: Rules of Thumb

- The number and the size of the hidden layers can have a large impact on the performance
- More hidden layers $\Rightarrow$ more parameters to learn $\Rightarrow$ more data you need
- Start with a small number of hidden layers, i.e. with 1
- Increase number of hidden layers stepwise until you find an optimum

Learning the Weights



Img-Source: http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Training

- Goal: to minimize the error over the training data
- Error: difference between the output of the network and the expected output (true label)
- Mean-squared error:

$$\frac{1}{2}\sum(y_i - o_i)^2$$

where y_i : the expected value of instance *i* and *o* the network's output of *i*

Error Function: Negative log-likelihood

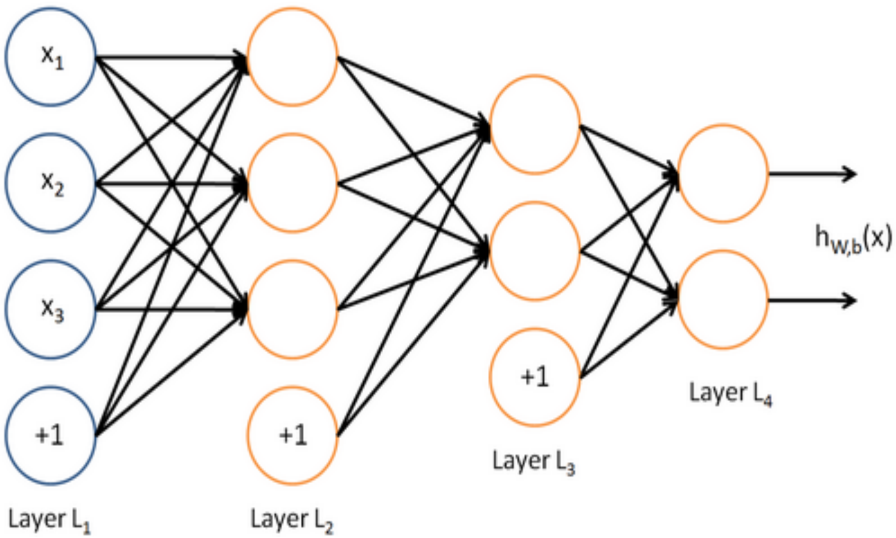
- Negative log-likelihood:

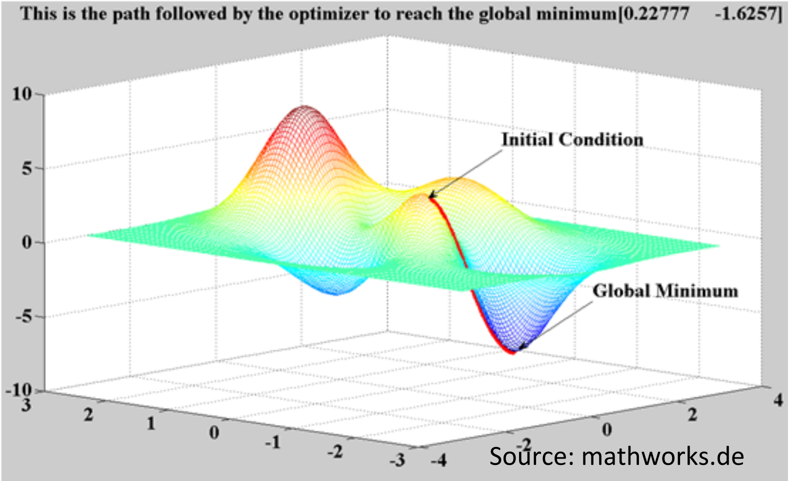
$$-\sum_{x \in X} log(P(Y = y^* | x))$$

- P(Y=y*|x): probability of x belonging to the true class y*
- Ex: given the true label of a sentence s is negative
  - The probability of s being negative returned by the network is 0.5
  - Error = - log(1/2) = 1
- Error = 0 if p = 1
- Error increases as p << 0

Example of Error Function

$$E(x, W, b) = -\log(\mathrm{softmax}(b_3 + W_3 \tanh(b_2 + W_2 \tanh(b_1 + W_1 x)))_y)$$



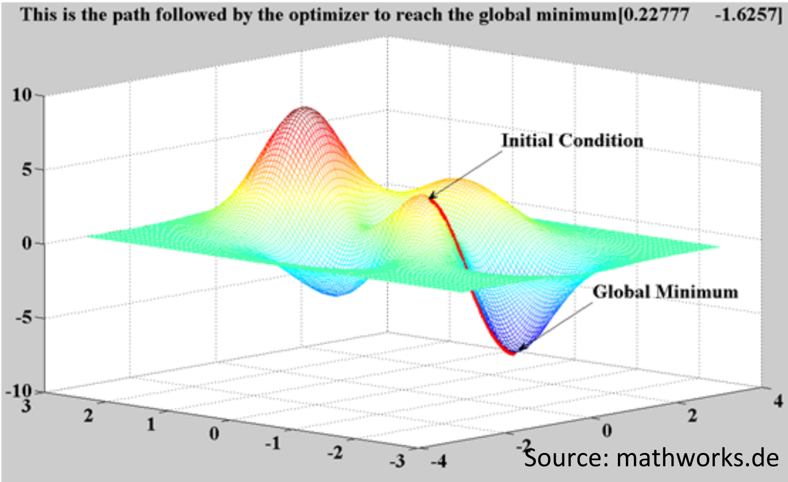


Error Curve

Img source: mathworks.de

Training: Back Propagation

- Tuning parameters (weights and biases) to minimize the error: optimization problem
- Analytical solution doesn't work: huge number of parameters



Img source: mathworks.de

- Gradient descent:
  – The gradient of the error curve points towards a local minima

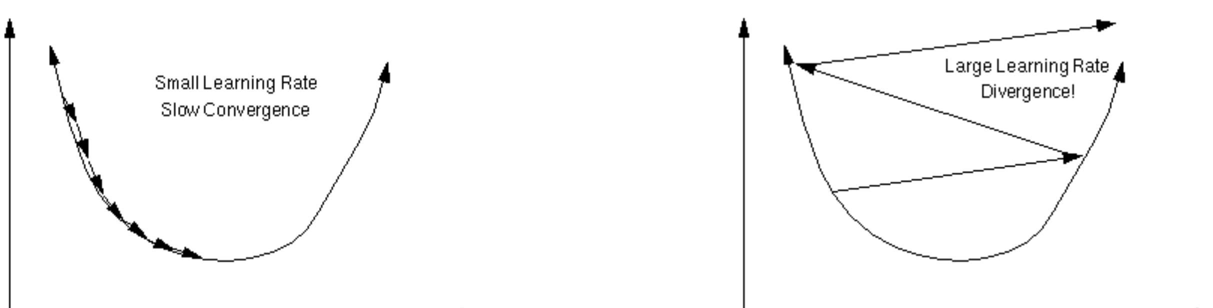
Training with Back Propagation

1. Initialize the weights and biases randomly
2. Given the input data, compute the values for the output neurons
3. Compare the output to the gold labels – compute error function
4. Compute the derivative for all tunable parameters (weights and parameters)
5. Update the parameters:

$$W^{(i)} := W^{(i)} - \lambda \frac{\partial}{\partial W^{(i)}} E(x, W, b)$$

$$b^{(i)} := b^{(i)} - \lambda \frac{\partial}{\partial b^{(i)}} E(x, W, b)$$

λ is denoting the learning rate



Img-Source: https://www.willamette.edu/~gorr/classes/cs449/linear2.html

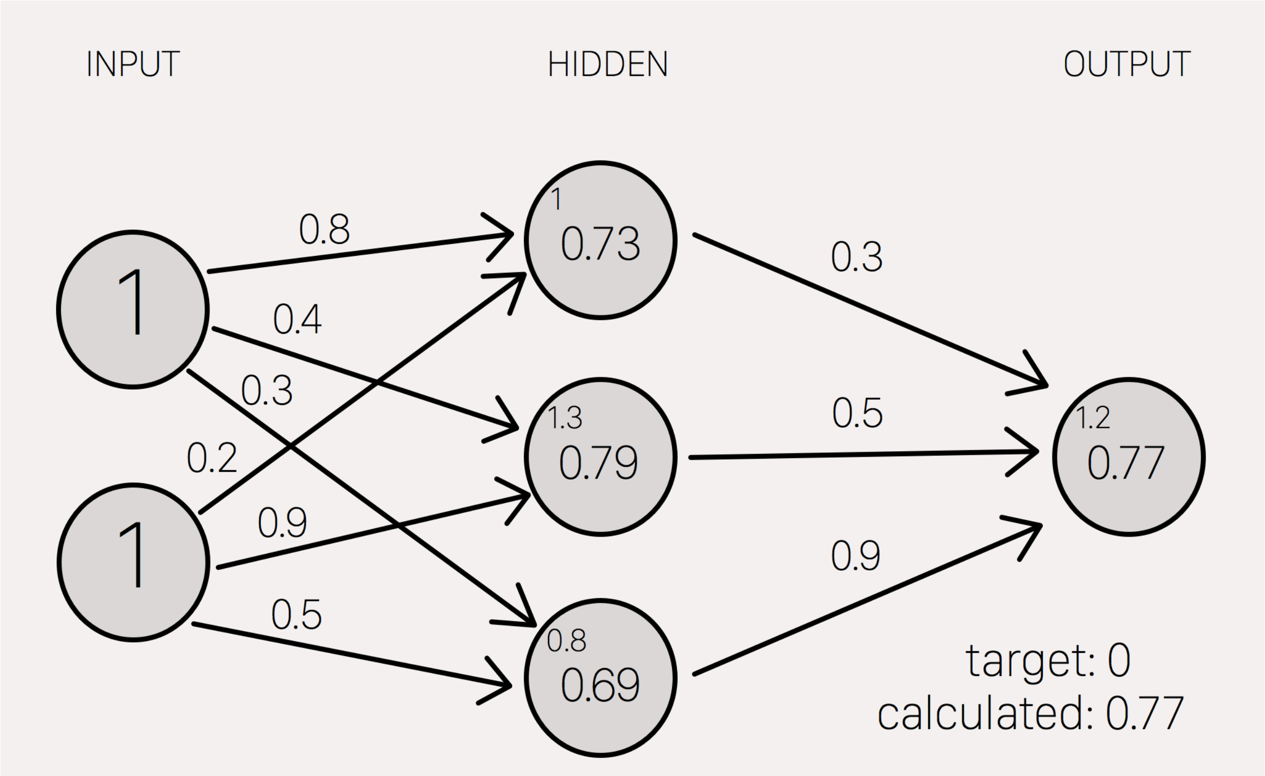
Training with Back Propagation

- Each iteration of backpropagation is called an epoch
- After each epoch, the error function decreases, converging to a local minima
- Mini-batches:
  1. Compute derivatives for few instances
  2. Accumulate them
  3. Update the parameters based on them

$$W^{(i)} := W^{(i)} - \lambda \frac{\partial}{\partial W^{(i)}} E(x, W, b)$$

$$b^{(i)} := b^{(i)} - \lambda \frac{\partial}{\partial b^{(i)}} E(x, W, b)$$

Back Propagation: Example

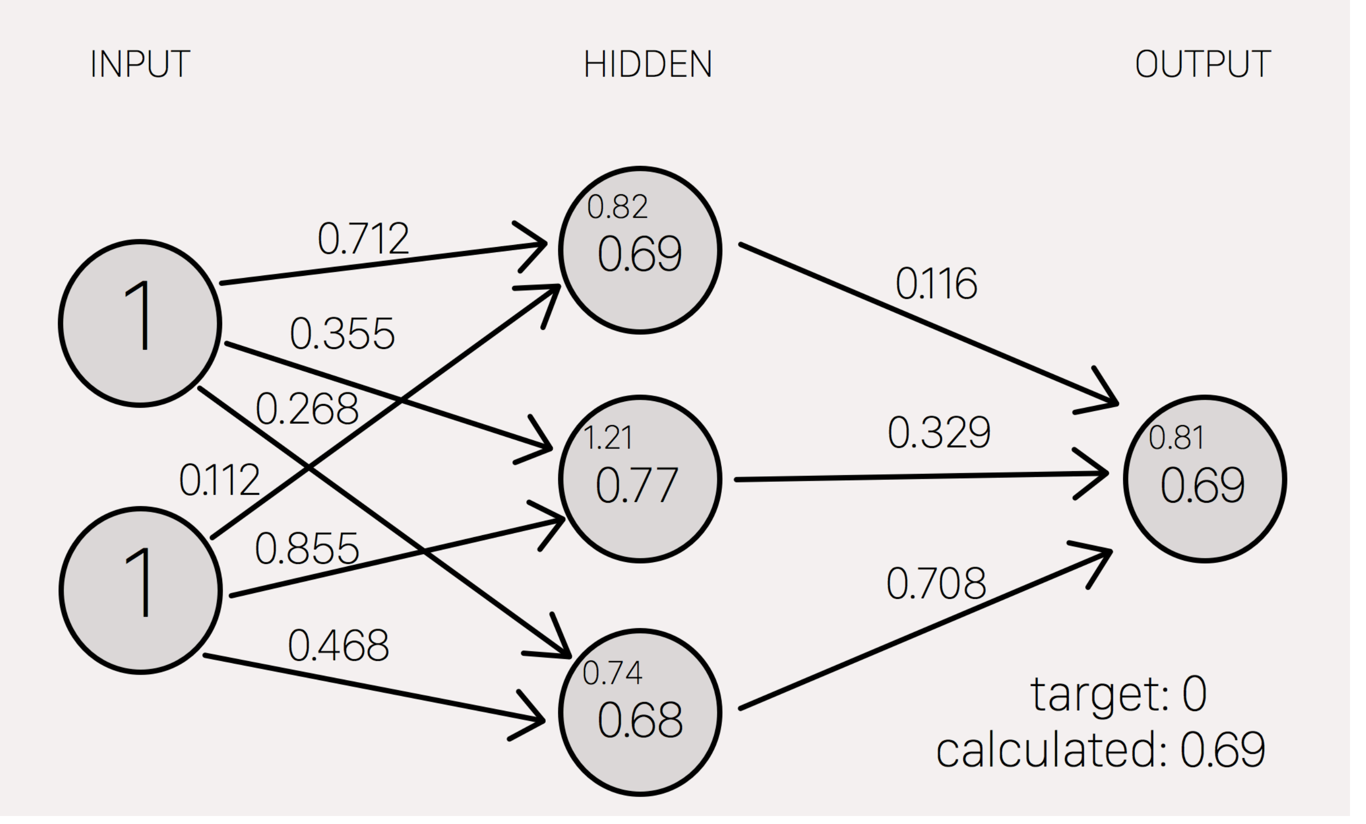


Img-Source: http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Back Propagation: Example

- After 1 epoch:

old	new
w1: 0.8	w1: 0.712
w2: 0.4	w2: 0.3548
w3: 0.3	w3: 0.2681
w4: 0.2	w4: 0.112
w5: 0.9	w5: 0.8548
w6: 0.5	w6: 0.4681
w7: 0.3	w7: 0.1162
w8: 0.5	w8: 0.329
w9: 0.9	w9: 0.708



Img-Source: http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/

Computation of Gradients

- Computation of the gradient (the derivative of a multi dimensional function) can be cumbersome
  - Billions of computations

- DL frameworks provide us automatic gradient computation
  - We don’t need to compute the derivative
  - And we don’t need to implement it into our program code

Motivation for Deep Learning

- In theory, feed forward networks with a single hidden layer can compute any function: no need for deep architectures
- However, learning shallow architectures is not always efficient
- Some problems require an exponential number of hidden units

What does a Deep Network can look like?

Convolution
Pooling
Softmax
Other

- Google’s Entry for the 2014 ImageNet Challenge
- 5 million parameters – 20MB model size

Requirements for Training

- Deal with billions of operations
  - Google had trained some models on up to 16000 cores
- Fast: performance in training time is **crucial**
- Nearly all operations are matrix operations (multiplications, additions)
  - Optimizing matrix multiplication for speed is hard
- Run on multiple CPUs and on GPUs