Applied Artificial Intelligence/ Artificial Intelligence

Neural Networks

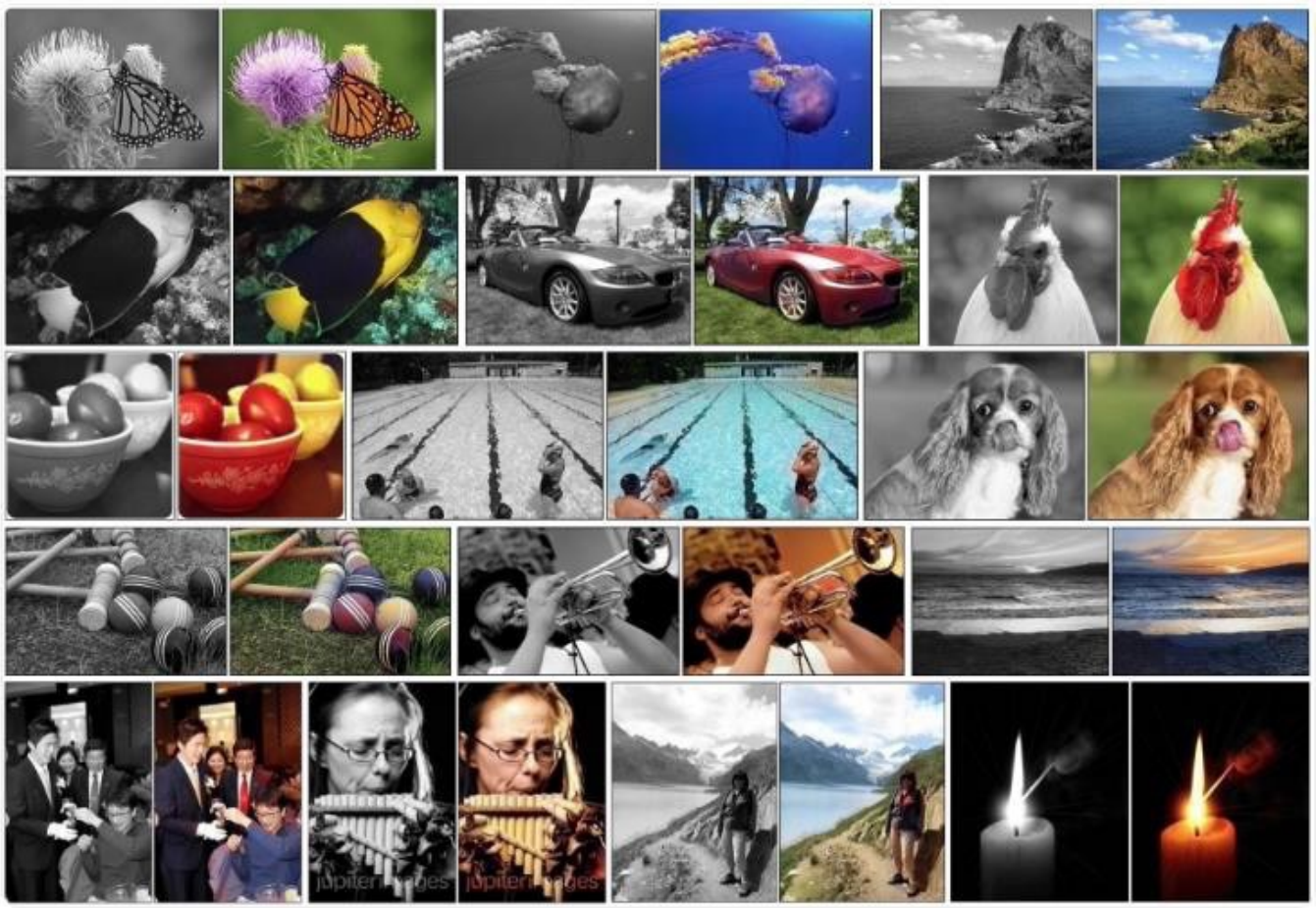
Neural Networks

- Biologically inspired family of algorithms that is modelled on the human brain

- Neural Networks can be used for classification, clustering and numeric prediction tasks.

Applications

- Automated colorization of black & white Pictures

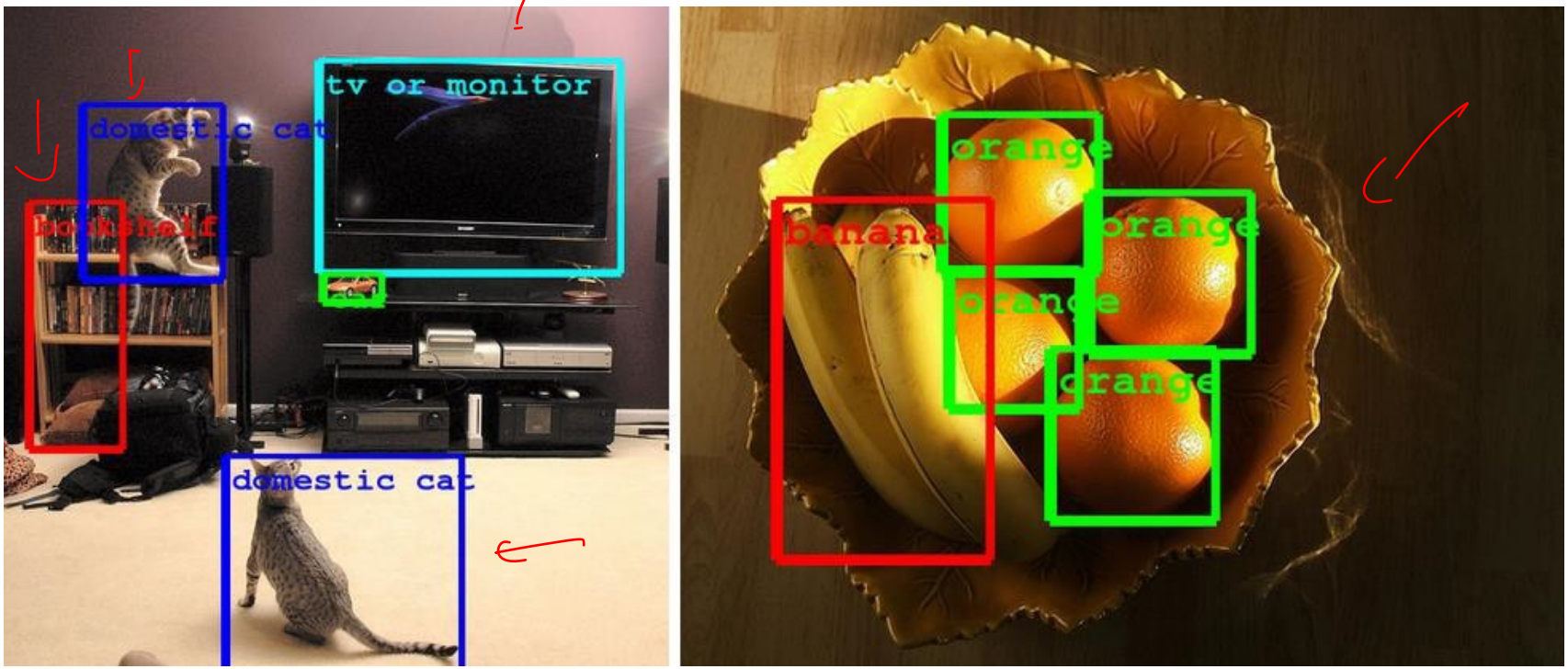


Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful image colorization." *European Conference on Computer Vision*. Springer International Publishing, 2016.

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Applications

- Automated object classification and detection in pictures



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Applications

- Adding sound to silent movies
- Automatic machine translation
- Turning sketches into paintings
- Image caption generation
- Computer vision
- etc.

"little girl is eating piece of cake."

"a young boy is holding a baseball bat."

Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

More Applications

- In general it can be used for classification as well as for numeric prediction
- For classification has been used for recognizing both printed and handwritten digits
- For numeric prediction has been used for forecasting time series such as weather data (temperature, pressure, wind speed, etc), stock market prices, etc.
- Neural Network handwritten recognition:

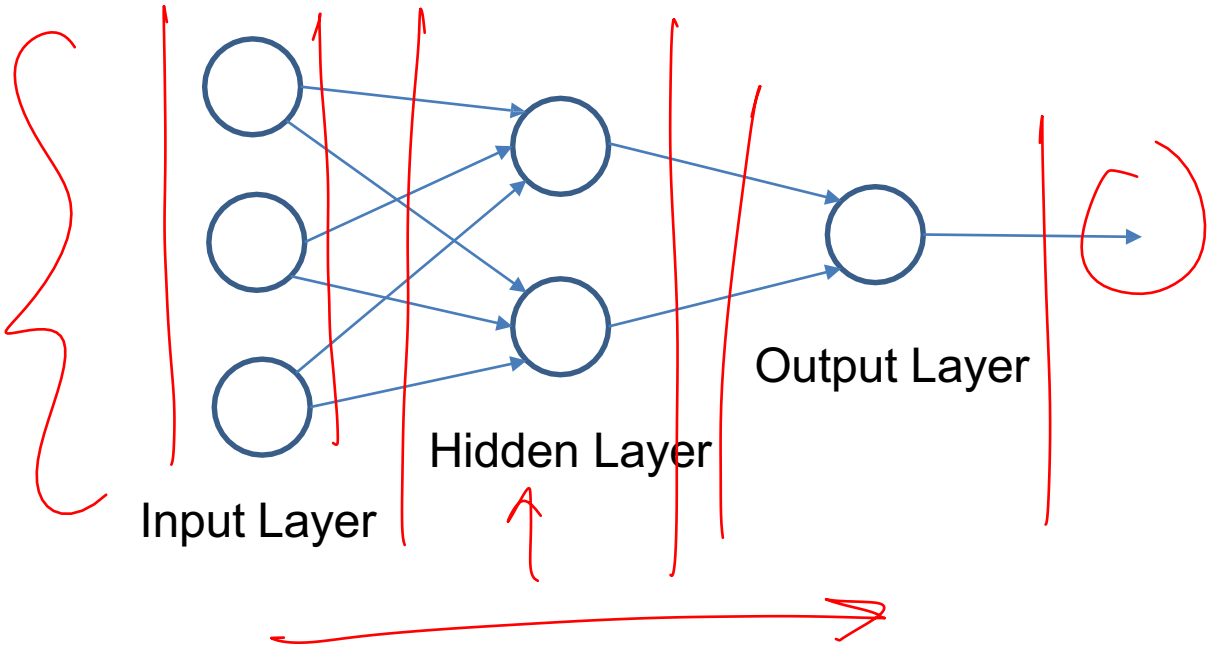
http://myselph.de/neuralNet.html
http://www.sund.de/netze/applets/BPN/bpn2/ochre.html

- Neural Nets play Pong:

http://www.youtube.com/watch?v=LD6OgKEj5JE

Artificial Neural Networks

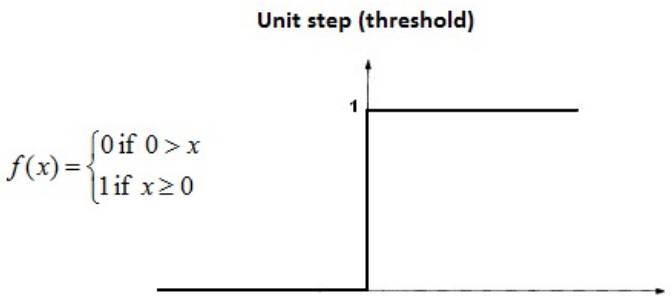
- **Artificial neural networks** (ANN) model the brain's biological structure.
- Typically organized in layers
  - – A layer comprises a number of interconnected "**neurons**", which deploy an **activation function**



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Activation Function

- **Activation functions** compute a mathematical operation on the signal to be output
  - Imposes a threshold upon which an output is sent
  - The choice of the function depends on the problem to be solved by the network
- Common activation functions include
  - Threshold function (unit step)
  - Linear function
  - Sigmoidal function
  - etc.



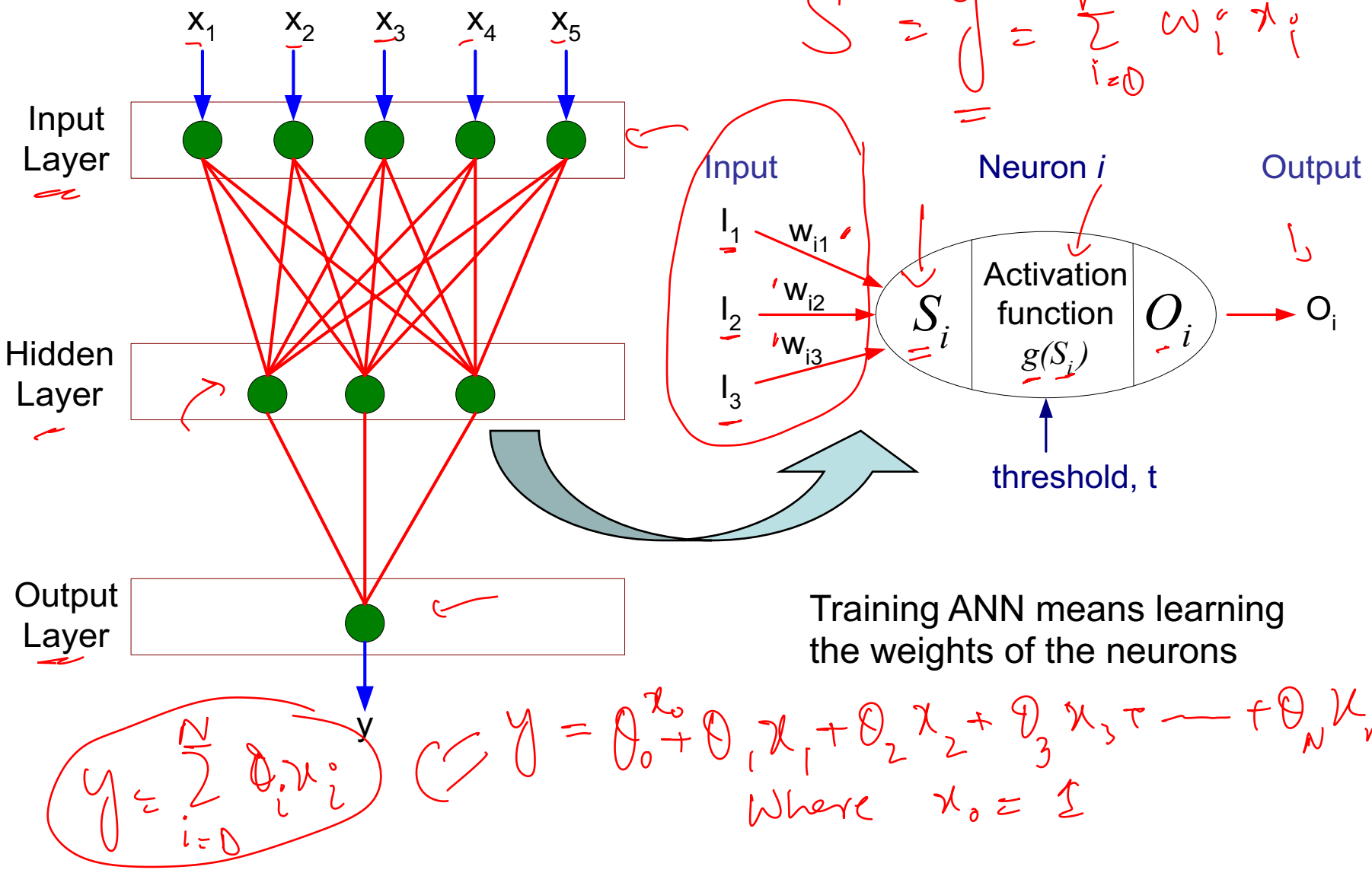
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

ANN Layers

- Input Layer
  - Patterns are input to the network through the input layer, which are in turn communicated to **one or more** hidden layers
- Hidden Layer
  - Using a system of weighted connections, it processes its input and communicates it to the output layer upon surpassing a given **threshold**
- Output Layer
  - Presents a **meaningful** transformation of the inputs

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

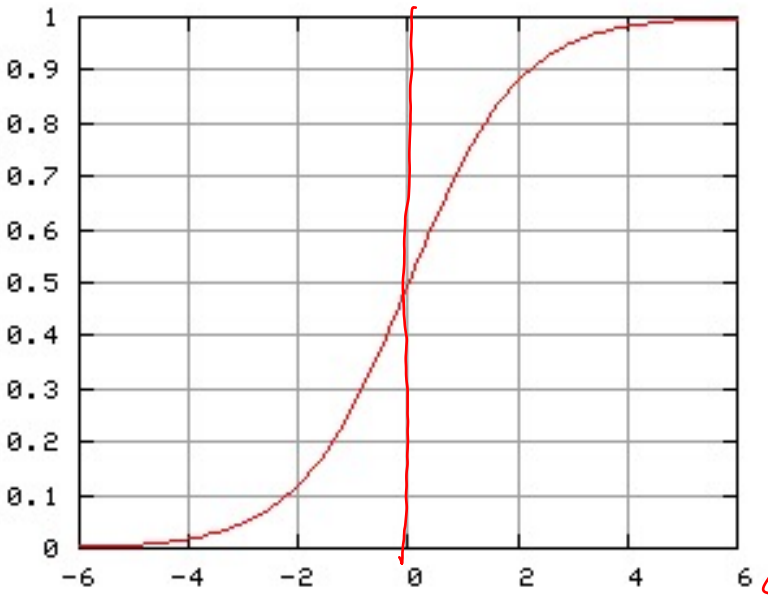
General Structure of ANN



Training ANN means learning the weights of the neurons

Activation Functions

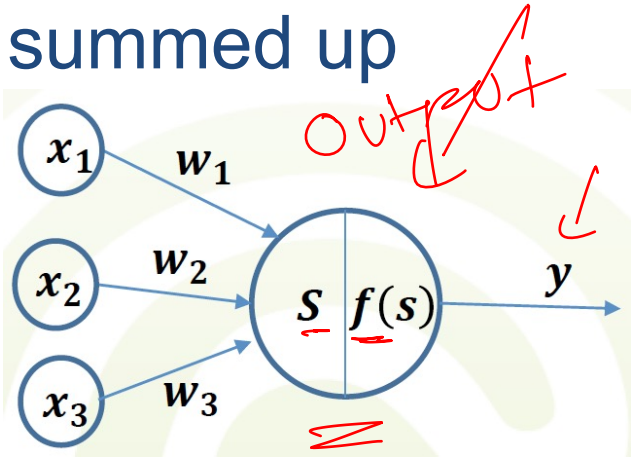
- Just as with human neurons, the neurons in the hidden layer are activated only when the input they receive is above a certain threshold value
- To model this situation the sigmoid function is commonly used as an activation function



Perceptron

- A modeled neuron is called a **Perceptron**
  - It receives n inputs
  - x_i signals are transmitted over w_i weighted links/synapses
  - Weighted input signals are summed up

$$S = \sum_{i=1}^{n} w_i x_i$$



  - Activation Function $f(s)$ is computed
  - If value exceeds some threshold the neuron fires a signal y, otherwise it remains silent

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Perceptron

- The ANN's **weights** can be either
  - Be set using a-priori domain knowledge
  - Be deduced through continuous weight re-adjustments upon training the ANN.
    - **Supervised learning:** ANN is trained with input and
    - corresponding output patterns (e.g., perceptron algorithm)
    - **Unsupervised learning:** ANN is trained to observe clusters of patterns in the given input

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

ANN Architectures

- There are mainly two network structures
  - **Feed-forward** neural networks
    - the input passes in one direction only
  - **Recurrent** neural networks
    - allows feedback and temporarily remembers previous input events

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

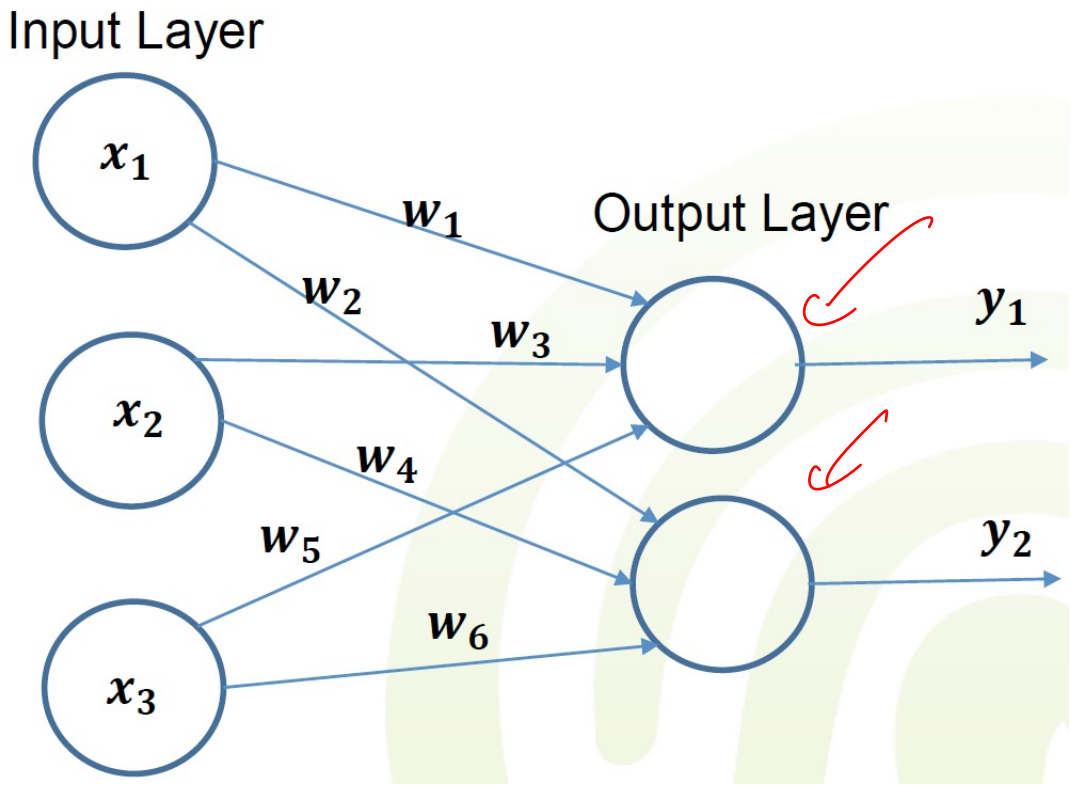
Feed-forward Networks

- **Feed-forward networks** are a special type of ANNs
  - may have zero to multiple hidden layers
  - No **feedback connections** are present i.e. connections from the output units to other units
- Common types of feed-forward networks
  - Single-layer perceptrons
  - Multi-layer perceptrons (MLP)

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Single-layer perceptron

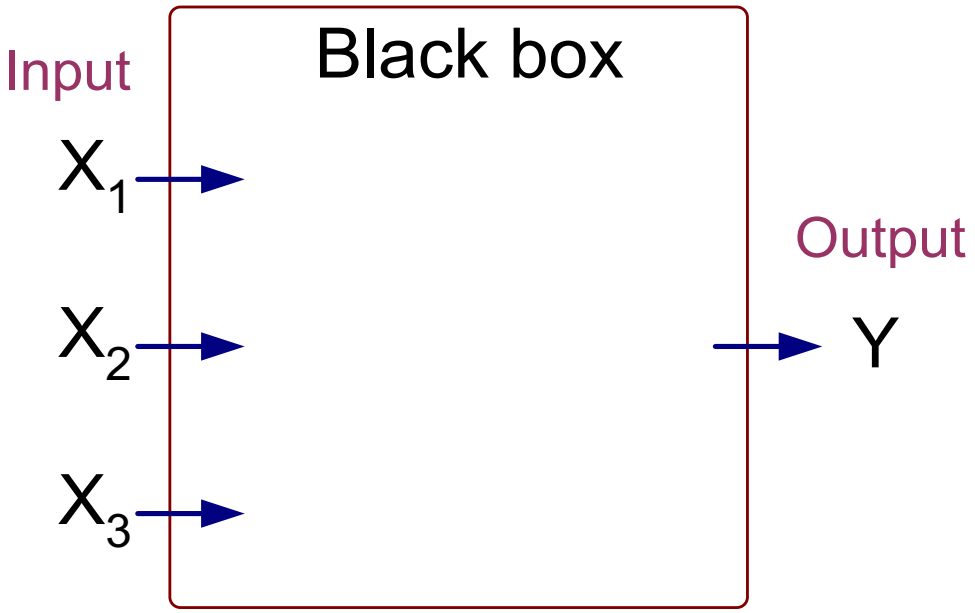
- The input layer of n neurons directly communicates with an output layer of m neurons



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Single-layer perceptron -Examples

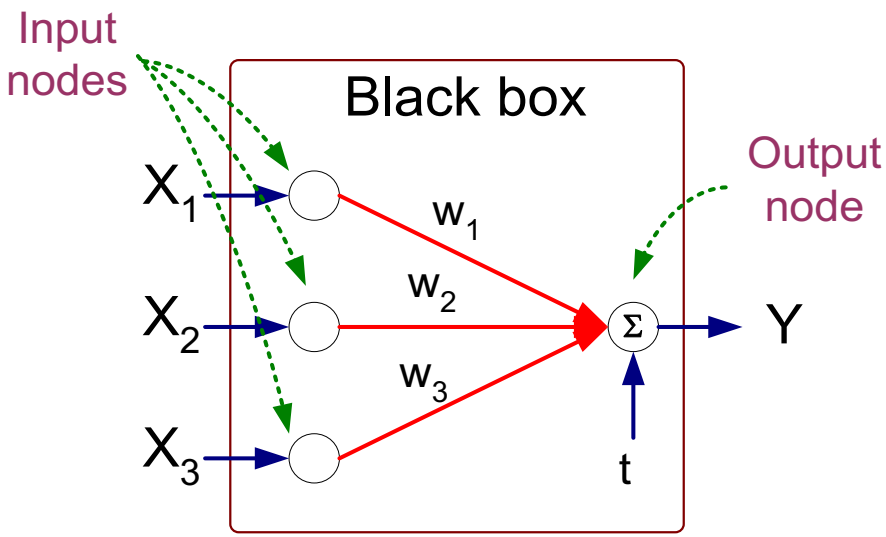
X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



Output Y is 1 if at least two of the three inputs are equal to 1.

Single-layer perceptron -Examples

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t



Perceptron Model

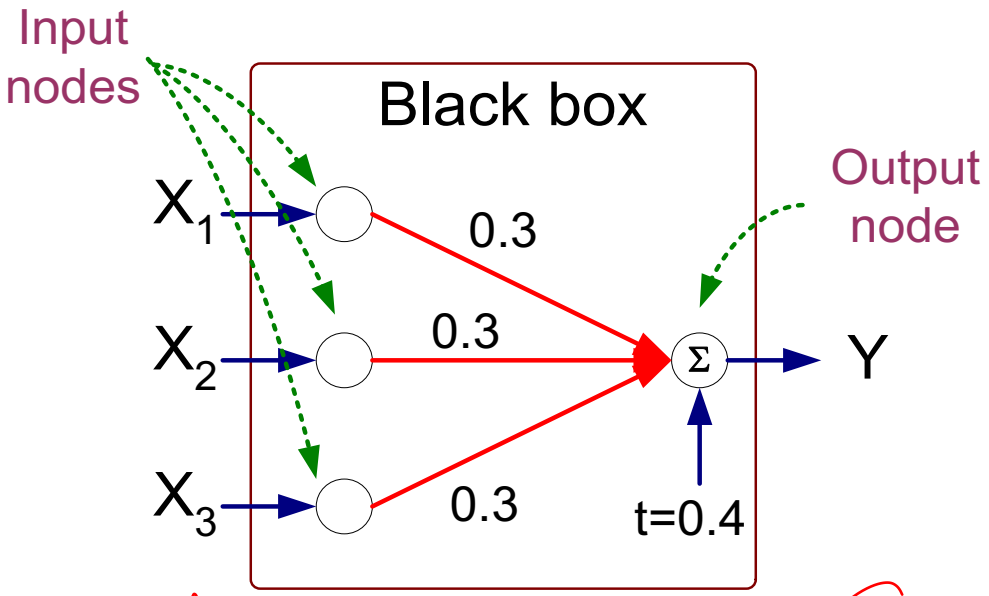
$$Y = I(\sum_i w_i X_i - t)$$

or

$$Y = sign(\sum_i w_i X_i - t)$$

Single-layer perceptron -Examples

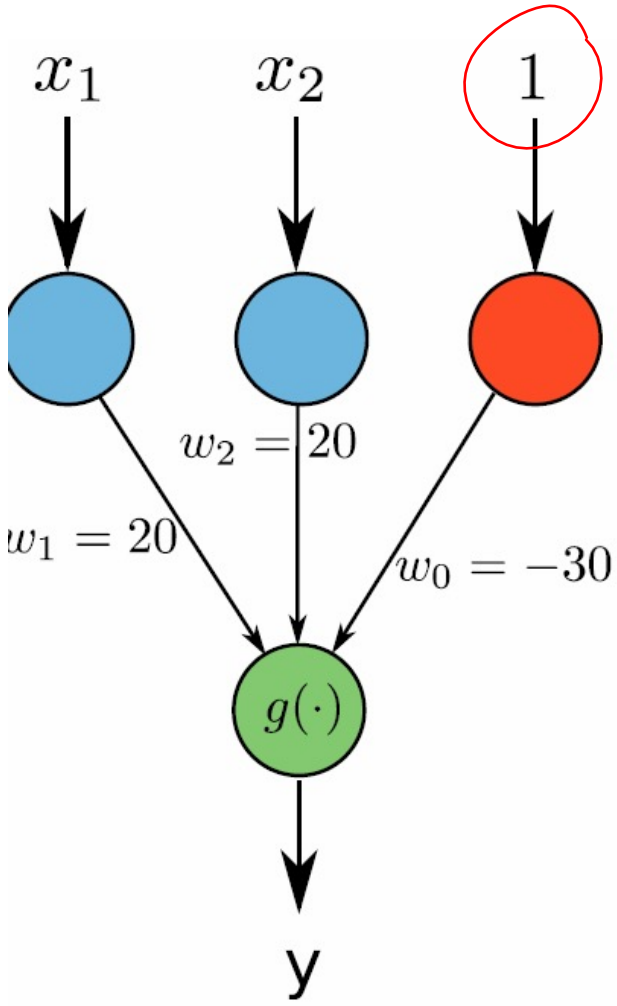
X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

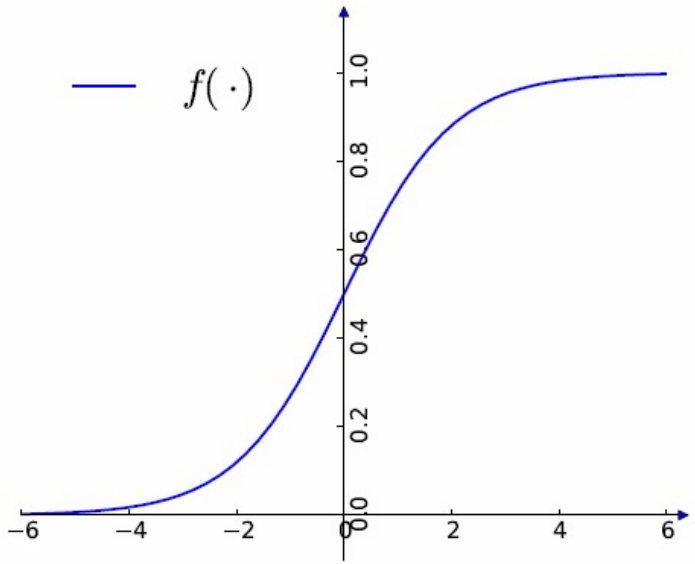


$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

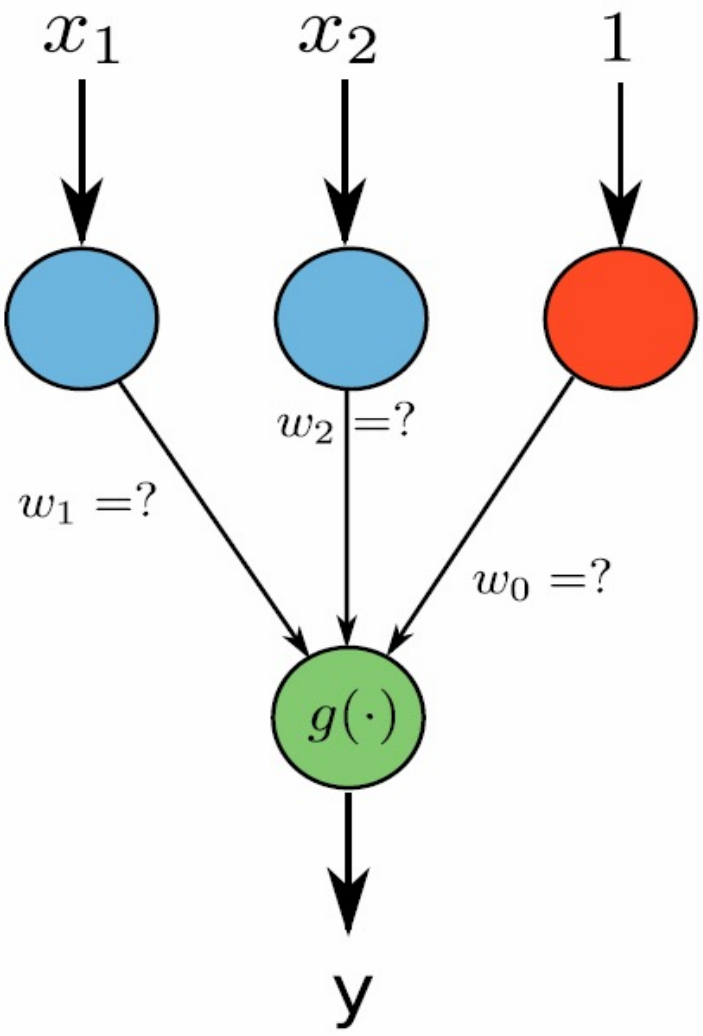
Solving the Logical AND Problem

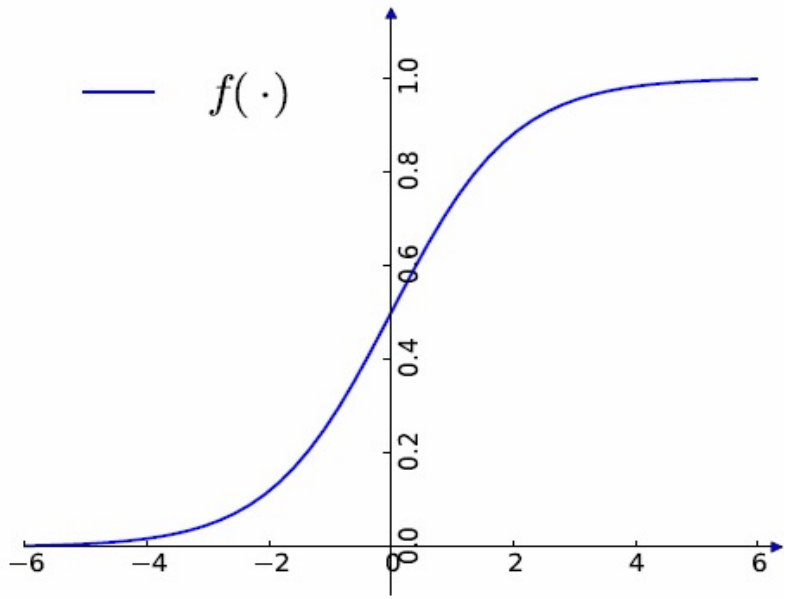




x_1	x_2	y	desired
0	0	$f(-30) \approx 0$	0
0	1	$f(-10) \approx 0$	0
1	0	$f(-10) \approx 0$	0
1	1	$f(10) \approx 1$	1

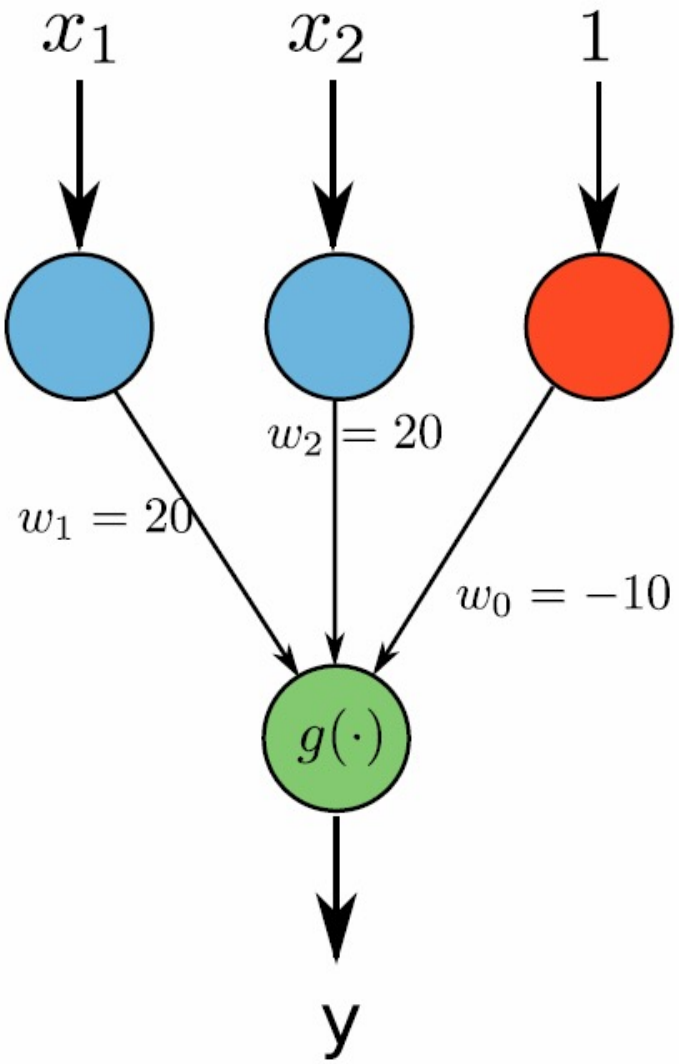
Solving the Logical OR problem

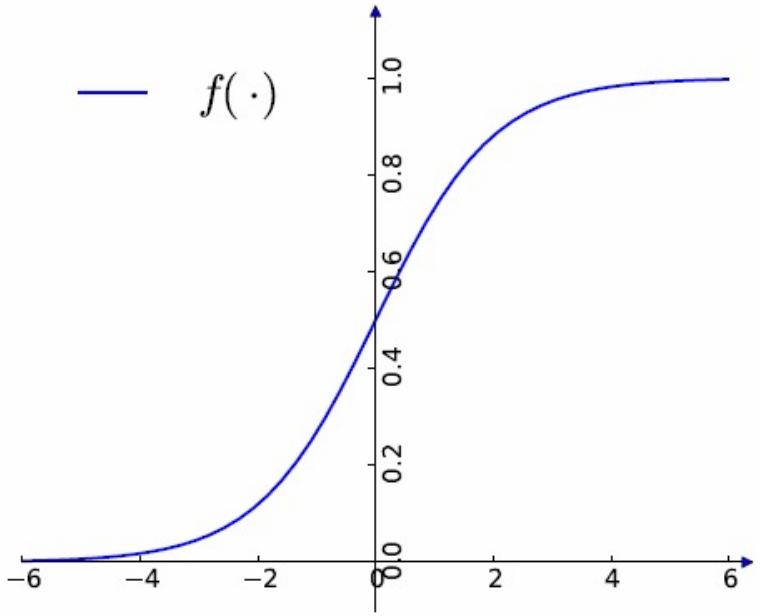




x_1	x_2	y	desired
0	0	?	0
0	1	?	1
1	0	?	1
1	1	?	1

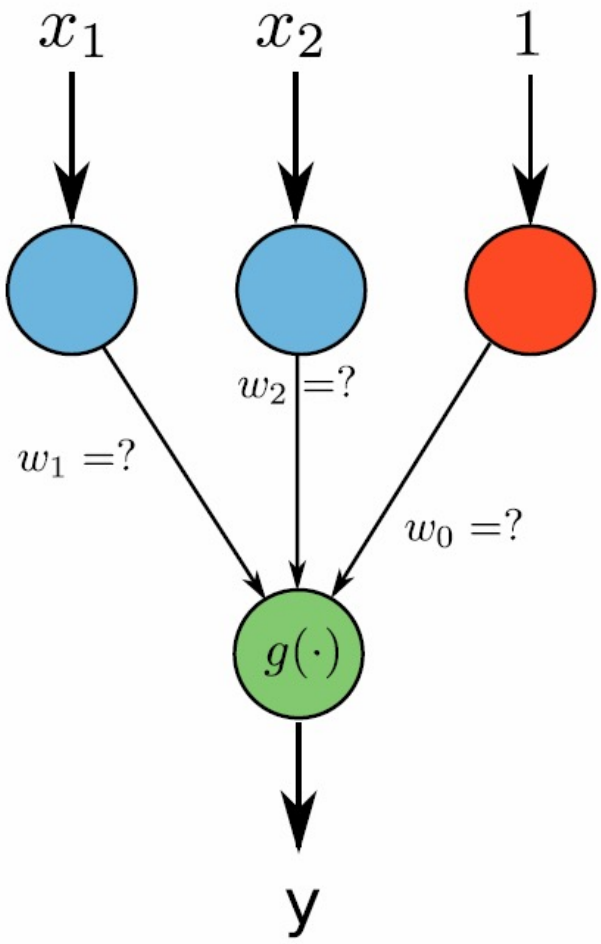
Solving the Logical OR problem

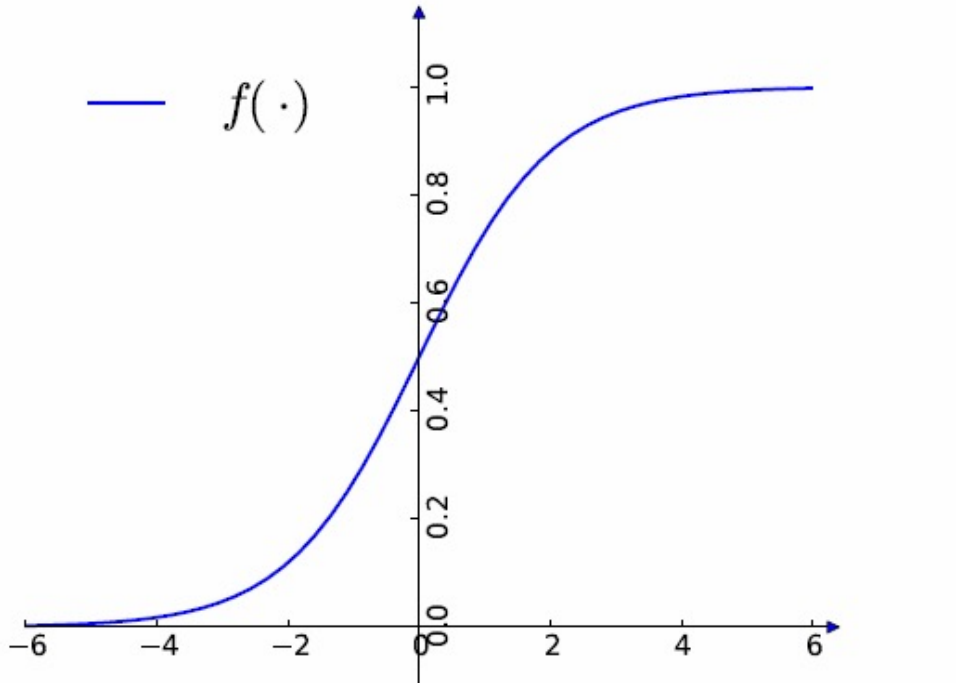




x_1	x_2	y	desired
0	0	$f(-10) \approx 0$	0
0	1	$f(10) \approx 1$	1
1	0	$f(10) \approx 1$	1
1	1	$f(30) \approx 1$	1

Solving the Logical AND with NOT as operands





X_1	X_2	NOT X_1	NOT X_2	y	desired
0	0	1	1	?	1
0	1	1	0	?	0
1	0	0	1	?	0
1	1	0	0	?	0

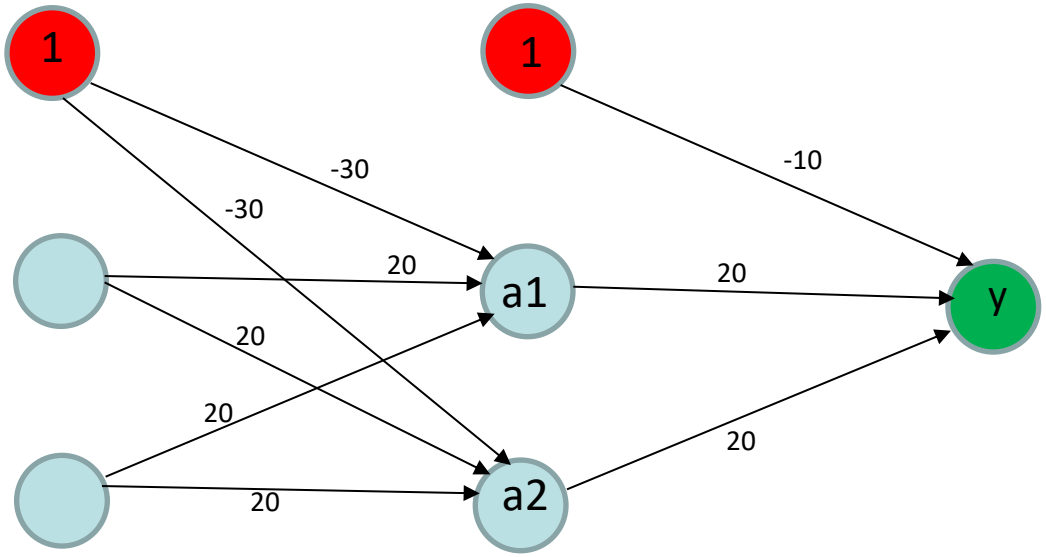
Solving the Logical XNOR problem

- The XNOR problem is more difficult than the logical AND problem.
- It cannot be solved by a single neuron as it is a 2 stage process
- (X1 XNOR X2) = a1 OR a2 where a1=(X1 AND X2) and a2=(NOT X1 AND NOT X2)
- This can be seen from the following truth table

X1	X2	a1	a2	a1 OR a2	X1 XNOR X2
0	0	0	1	1	1
0	1	0	0	0	0
1	0	0	0	0	0
1	1	1	0	1	1

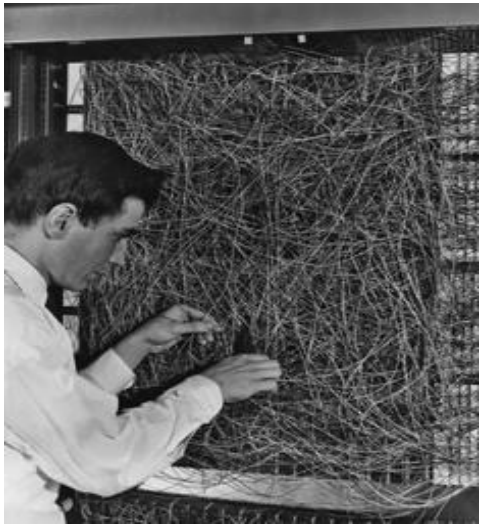
Neural Net for Solving the Logical XNOR Problem

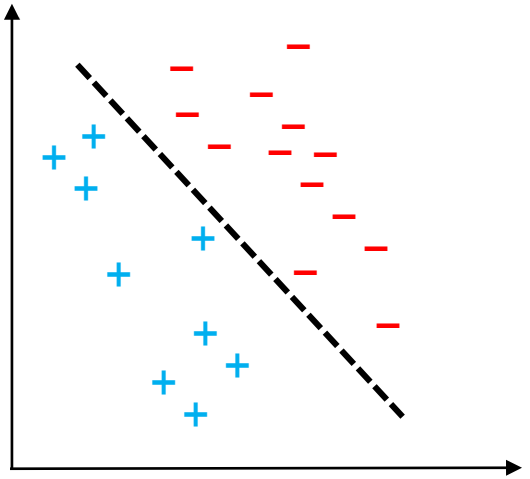
- a1 and a2 can be computed in parallel and so 2 neurons can be assigned to do the computation in the hidden (intermediate layer).



Perceptron Algorithm

- The perceptron algorithm is a **supervised** machine learning algorithm
  - Developed in the late 1950 by Frank Rosenblatt
    - The perceptron. A probabilistic model for information storage and organization in the brain“. In: *Psychological Reviews*, 65,1958.
- **Aim**: Learn the correct weights for the input patterns in order to ultimately draw a linear boundary to separate a linearly separable binary data set





Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Perceptron Algorithm

- The network is presented with several **training patterns** and computes the respective **outputs**
  - The connection weights w_i are **modified**, such that they are proportional to the product of
    - The input patterns x_i
    - The difference between the output y and the desired output d

Iteration number

$$w_i(t+1) = w_i(t) + \eta(d - y)x$$

step size, $0.0 < \eta < 1.0$

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Perceptron Algorithm

- The **Perceptron Algorithm**
  1. Initialize the weights and threshold with small random values
  2. Communicate training patterns $\boldsymbol{x}$ and compute the output $\boldsymbol{y}$
  3. Update the weights $\boldsymbol{w}_i$
  4. Back to step 2 until
     - A predetermined number of iterations is reached
     - The error is less than a predefined threshold

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

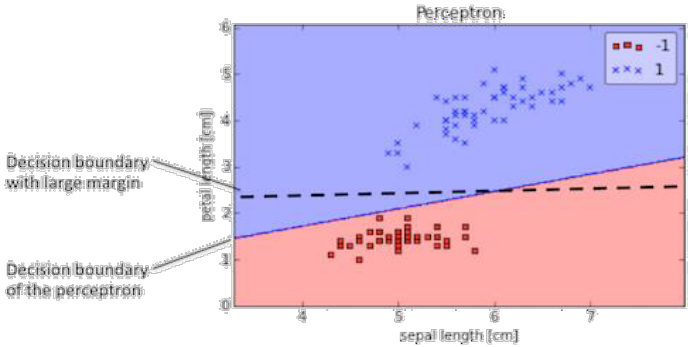
Network Performance

- To find the optimal weights across all the connections, the overall performance of the network can be measured.
- – A common cost function

$$E = \frac{1}{2}\sum_{i=1}^{n}\|y_i - d_i\|^2$$

Limitations of Simple Perceptrons

- Simple perceptrons have some **drawbacks**…
  - The perceptron algorithm only converges, if the two classes are actually **linearly separable**
    - Otherwise the learning algorithm never stops updating weights due to **inevitable** misclassifications
  - The algorithm usually does not terminate with a **maximum margin** classifier

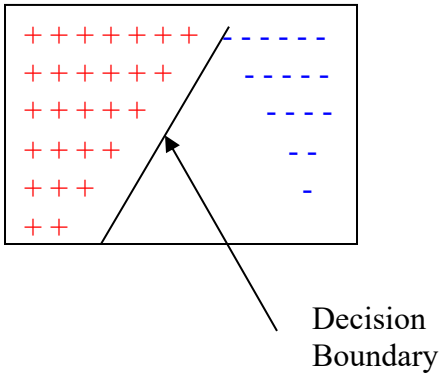


Sebastian Raschka,"Single-Layer Neural Networks and Gradient Descent", 2015

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

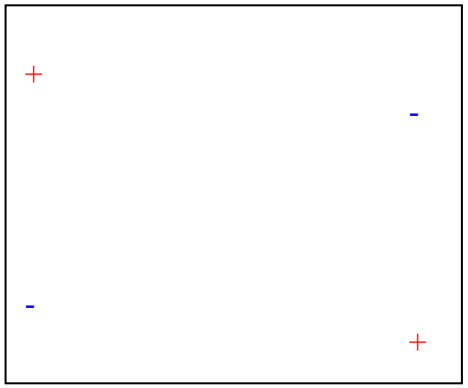
Limitations of Simple Perceptrons

- Simple perceptrons can be used to classify problems which are linearly separable
- For such problems a single line can be drawn which separates the two classes with zero (or near zero) error



Limitations of Simple Perceptrons

- However simple perceptrons cannot solve non linear classification problems such as the XOR problem
- These types of problems can only be solved by adding another layer (called the hidden layer) of neurons to the network



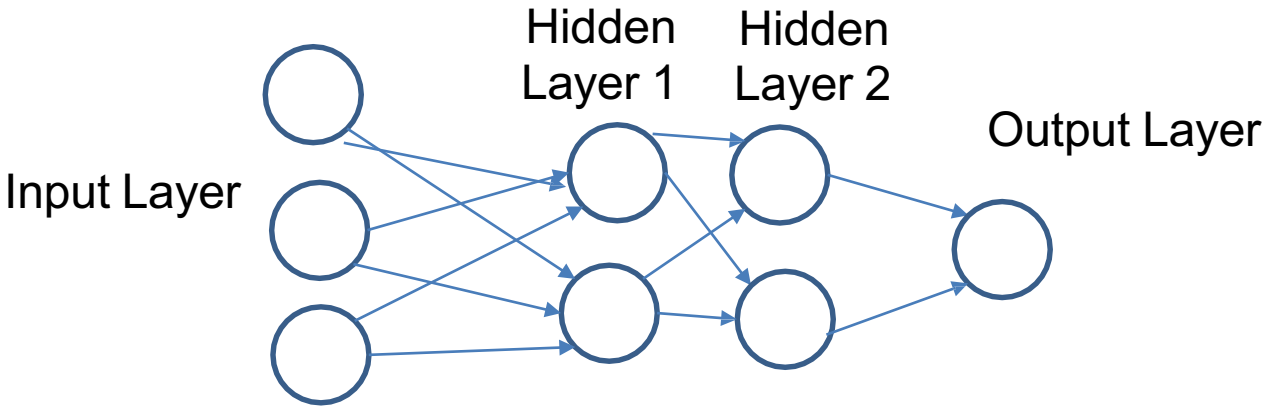
Multi-layer Perceptron

- **Multi-layer perceptrons** (MLPs) avoid some problems of single layer perceptrons
  - Introduced by M. Minsky and S. Pa pert in1969
  - MLPs comprises one or more hidden layers
  - Whereas a single perceptron can only learn linear functions, MLP **can also learn non-linear functions**

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

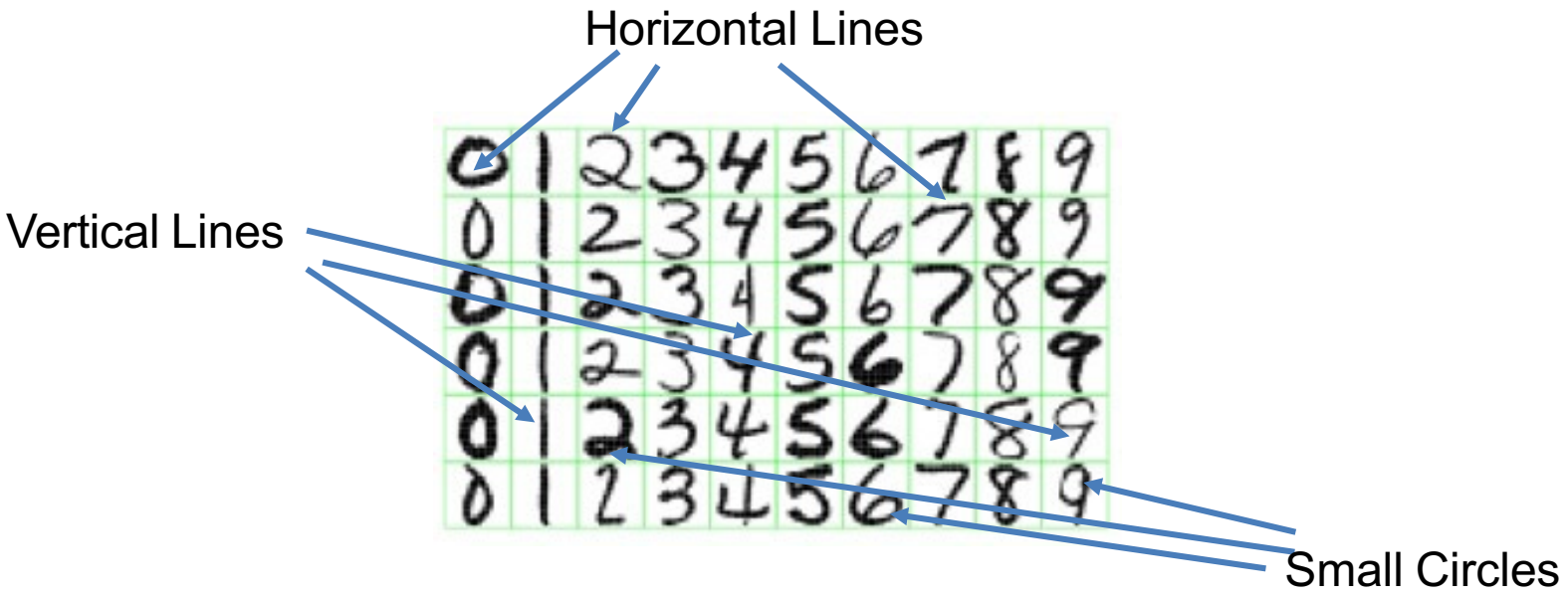
Multi-layer Perceptron

- Each hidden layer progressively identifies and extracts **higher-level Features** of the input
  - The more layers a network comprises, the higher-level features are learned



Feature Detection

- Neurons are trained to detect specific features, by transforming the weighted input with some **activation function**
  - – Example: Handwritten digits on U.S. postal envelopes



Back Propagation

- **Back propagation** is the most common learning algorithm to train multiple layers of non-linear features
  - Also known as the **delta learning algorithm**
  - Deltas are propagated backwards from output to hidden layers and to input in order to adjust errors
  - Through backward error transmission, it modifies the weights of the previous layers, so that the final output is more accurate
  - Uses **Gradient descent** to minimize the error

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Major Parameters for Multi Layer Perceptrons

- **Learning rate** – this determines the size of the "steps taken" in the weight adjustment process – larger steps means learning takes place quicker but accuracy may suffer
- **Number of epochs** – the number of times that the training dataset is scanned – larger the value the more accurate the model (generally 100 or more)
- The **number of hidden neurons** used – generally chosen as (attributes+classes)/2
- **Momentum** – some implementations add a term called the momentum to the current weight – this is a small fraction of the update value from the previous iteration; the momentum makes the learning process smoother

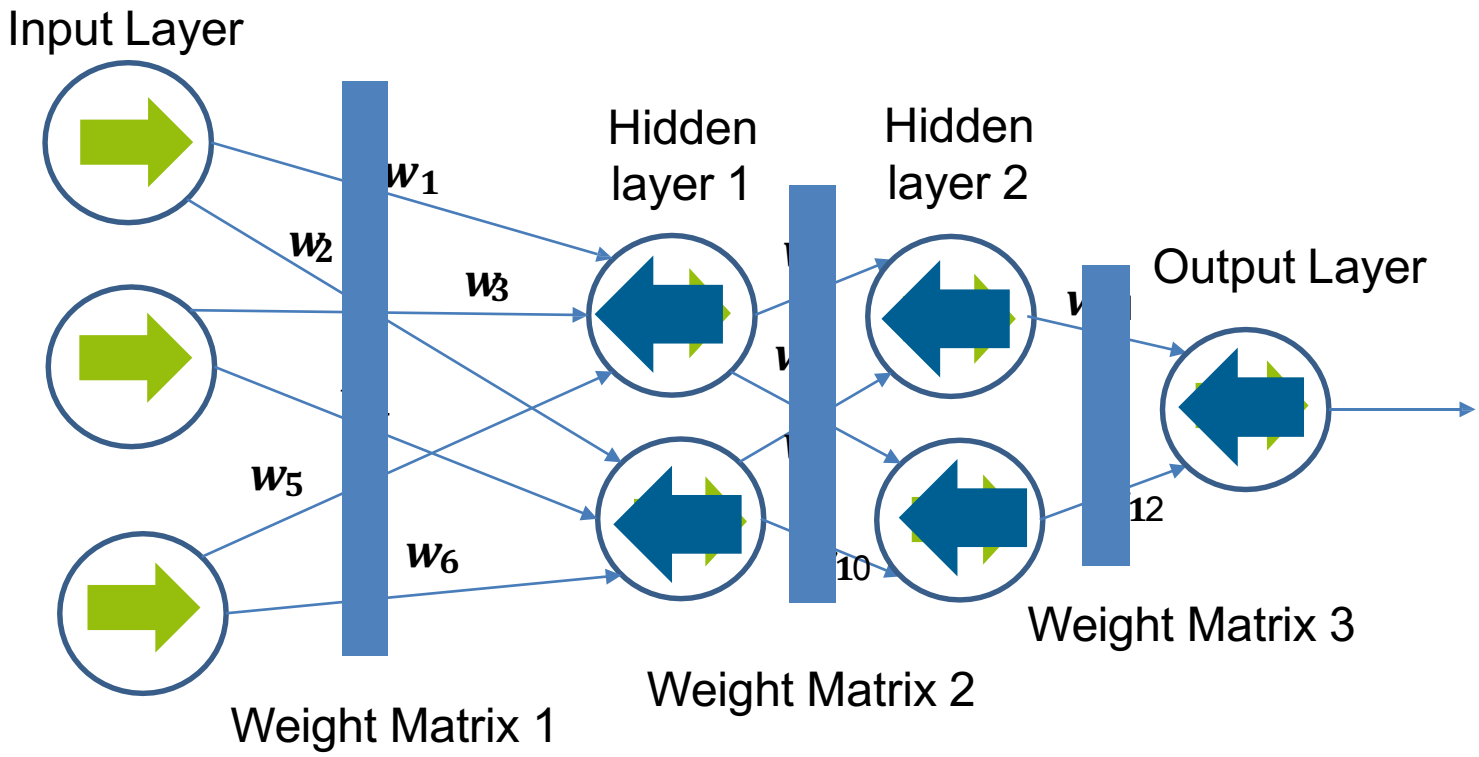
Back Prop. Algorithm

- The **Back Propagation Algorithm**
  1. Initially all weights are randomly assigned
  2. Feed the network with a labeled training data set
  3. For every input, the output produced by the neural network is compared with the expected output
  4. The error is propagated back to the previous layers
  5. Each layer re-computes the new weights and the deltas, which is to be updated later

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Back Prop.Algorithm

6. Repeat step 4 and 5 till the first hidden layer is reached
7. Update all the weights



Back Propagation Issues

- Also back propagation has some serious
- **drawbacks**…
  - Back propagation is very **slow**, especially within networks comprising several hidden layers
  - Like single layer perceptrons, it may terminate at a **poor local optimum**
  - As a supervised approach, it requires **a lot of**
    - labeled training data

In short, weight-learning algorithms fell short when dealing with multi-layered Networks until….

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Neural Networks - Strengths

- Neural Nets work well with datasets containing noise
- Have consistently good accuracy rates across several domains
- Can be used for both supervised (classification and numeric prediction) as well as unsupervised learning

Neural Networks - Weaknesses

- Lack the ability to explain their behaviour (unlike Decision Trees and Naïve Bayes)
- In some cases, overtraining can cause over fitting – i.e. the NN has a good accuracy on training data but poor accuracy on the test data
- With large datasets training time can be large – very much larger than the Decision Tree and Naïve Bayes methods

Next lecture

- Deep Learning Continued

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides