Robotics / Intelligent Robotics

Machine Learning – Neural Networks/Deep learning

Luís Paulo Reis, Nuno Lau, David Simões, Armando Sousa

lpreis@fe.up.pt

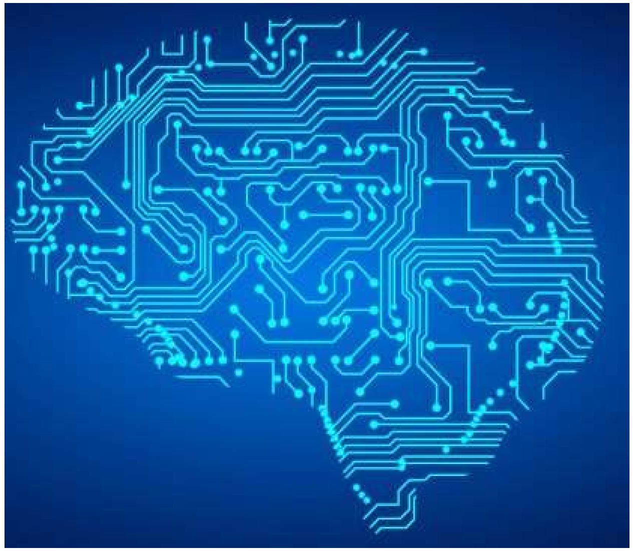
Director of LIACC – Artificial Intelligence and Computer Science Lab.

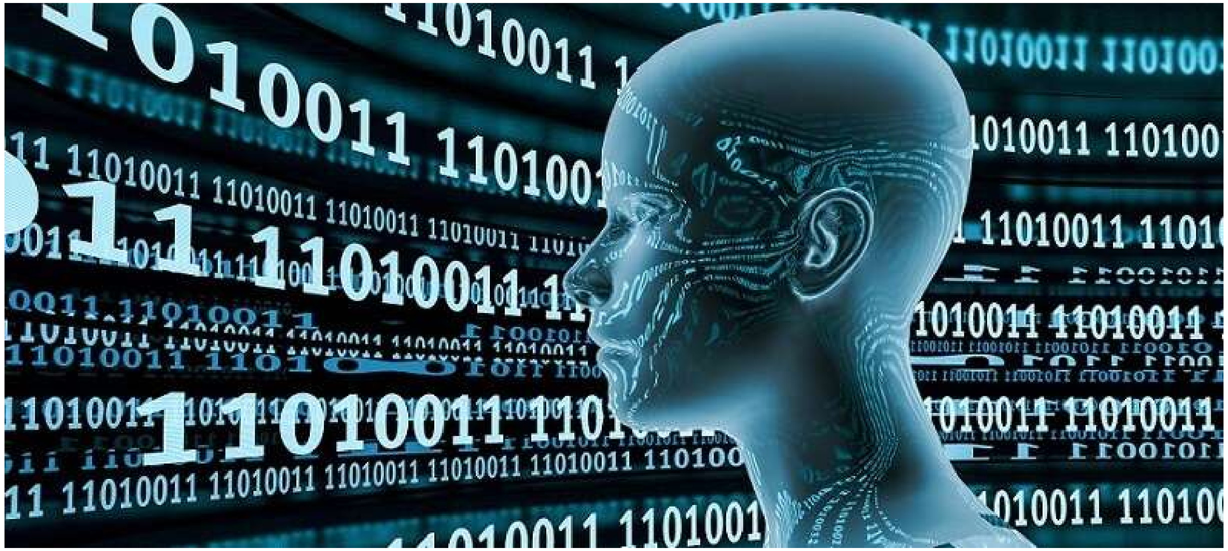
Associate Professor at DEI/FEUP – Informatics Engineering Department, Faculty of Engineering of the University of Porto, Portugal

President of APPIA – Portuguese Association for Artificial Intelligence

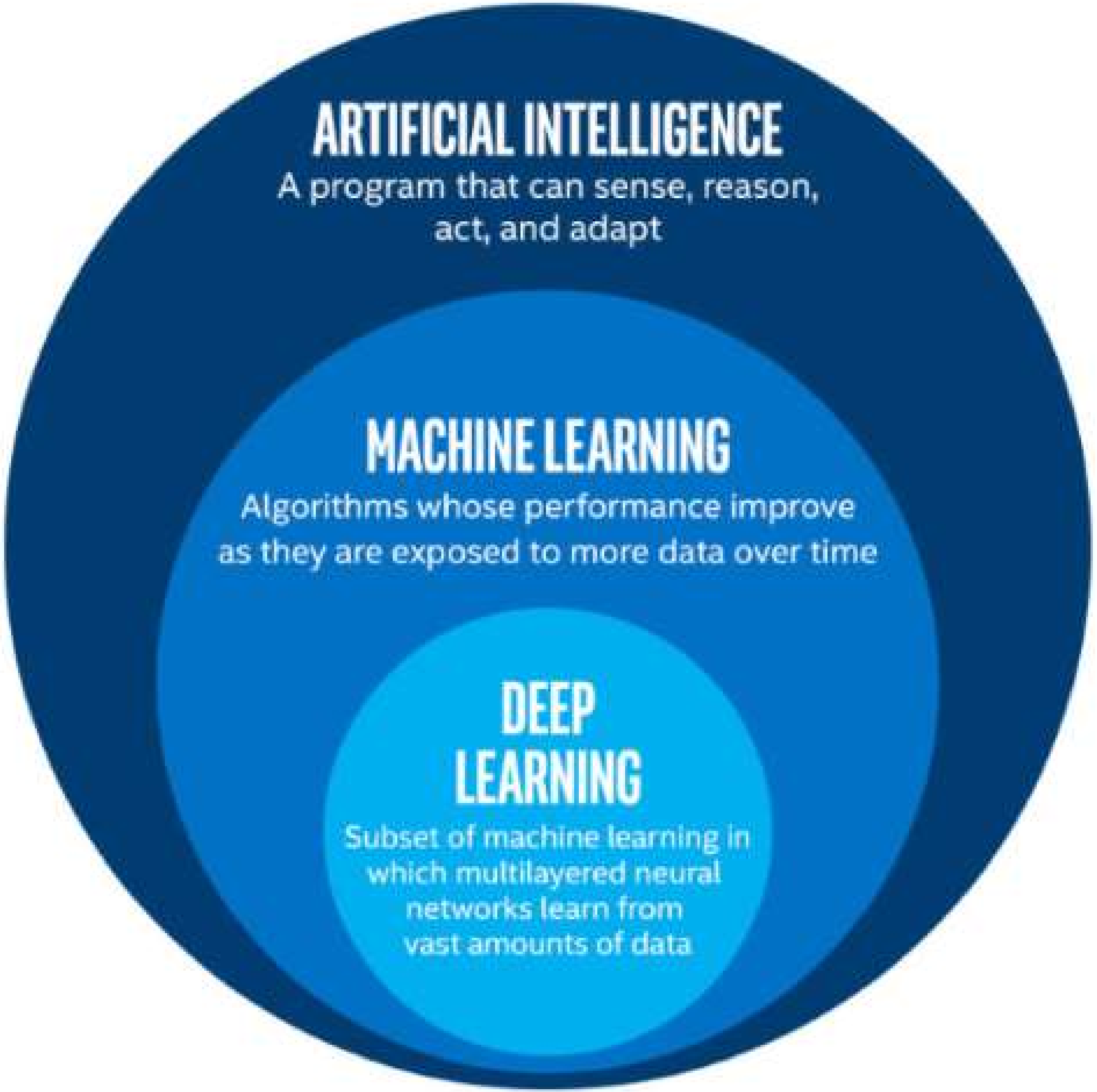
Machine Learning

- Machine learning is a field of artificial intelligence that gives computer systems the **ability to "learn"** (e.g., progressively **improve performance** on a specific task) from data/results of their actions, without being explicitly programmed

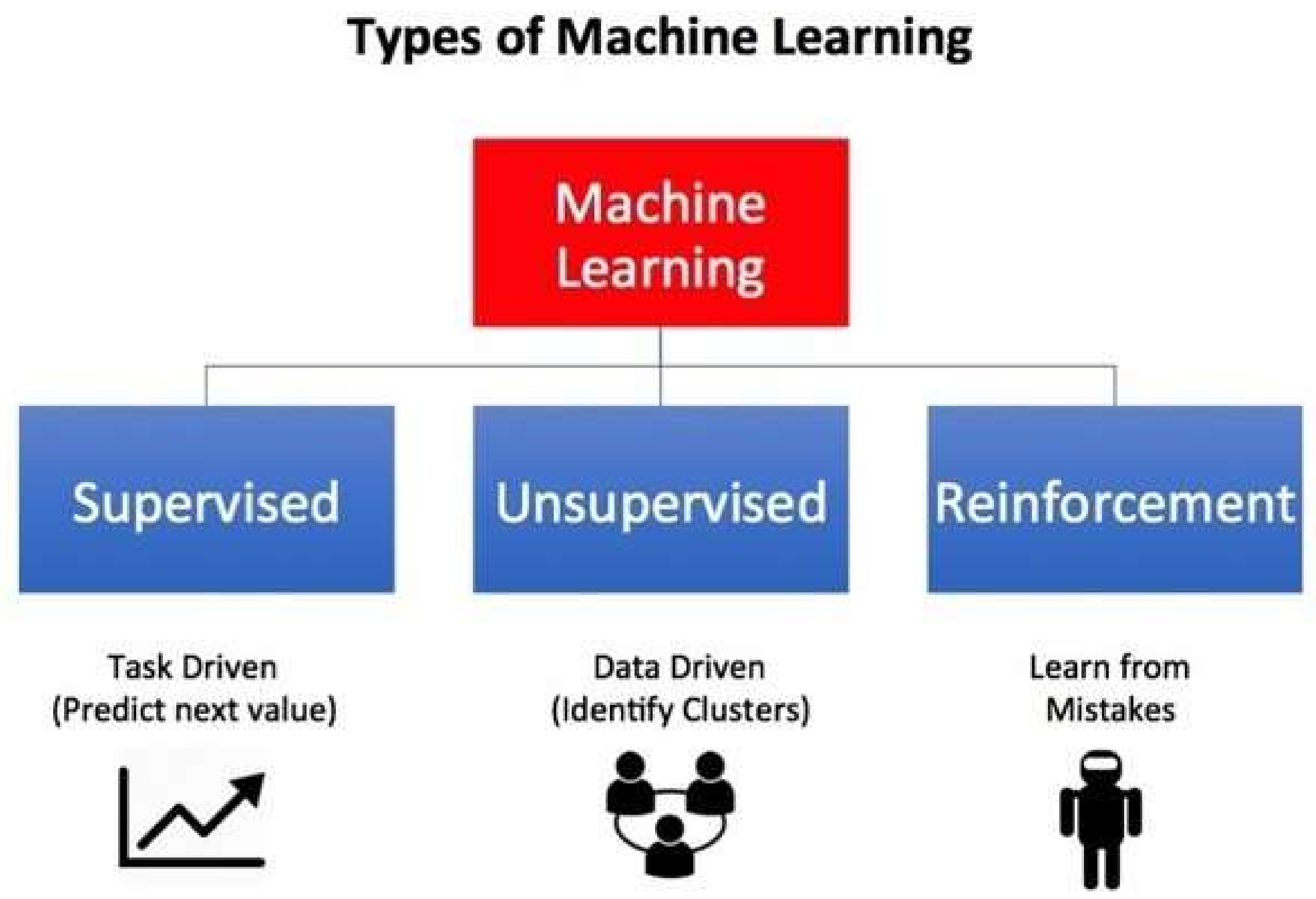




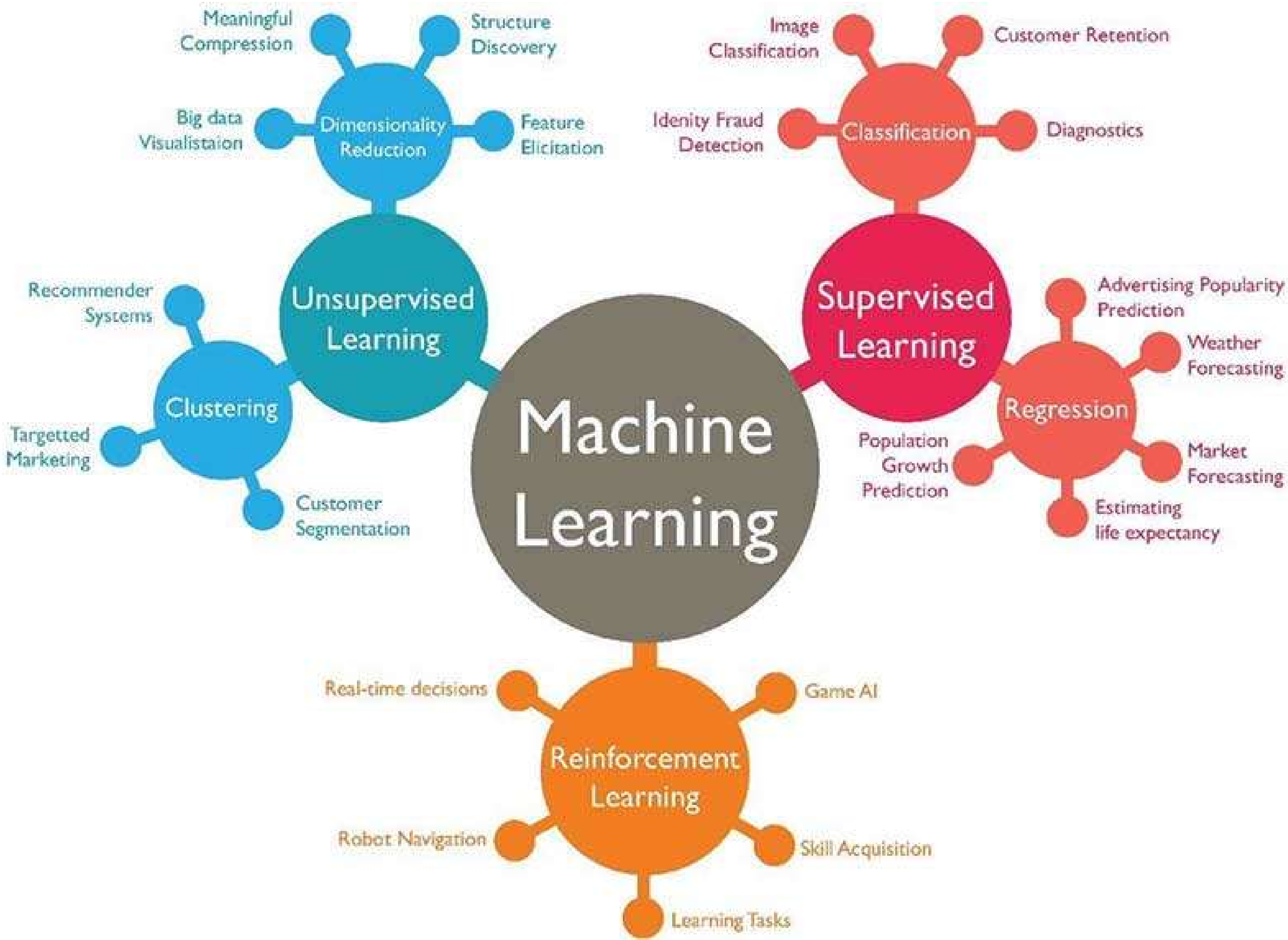
Machine Learning vs. Artificial Intelligence



Machine Learning - Types

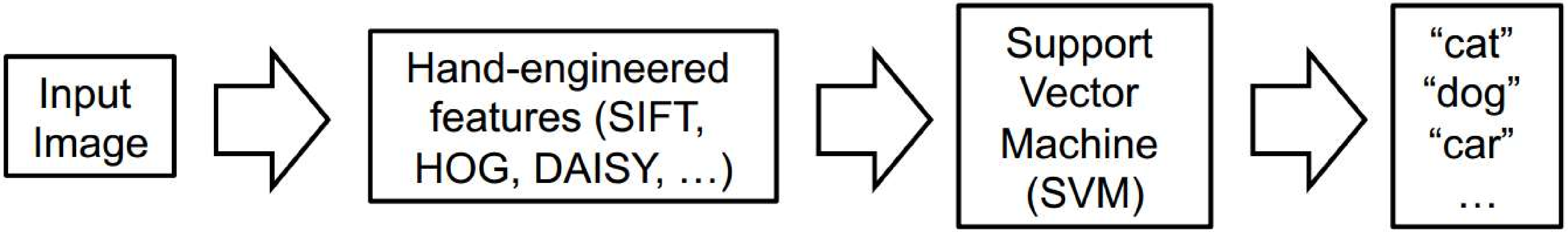


Machine Learning

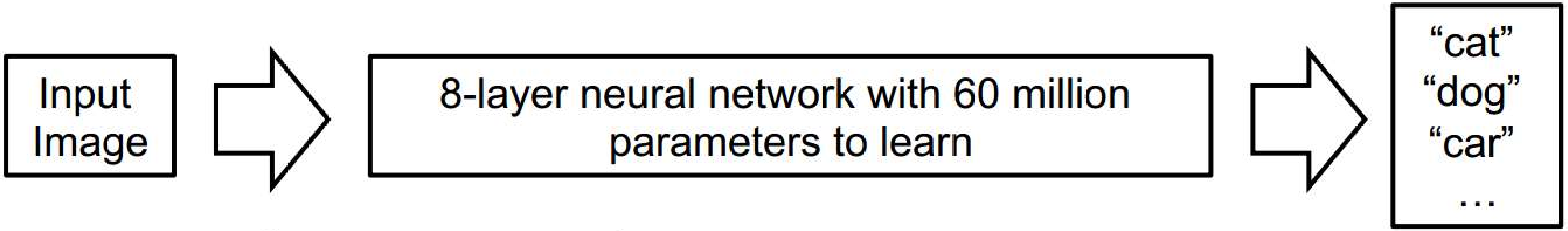


Machine Learning - Supervised

- State-of-the-art object detection until 2012:

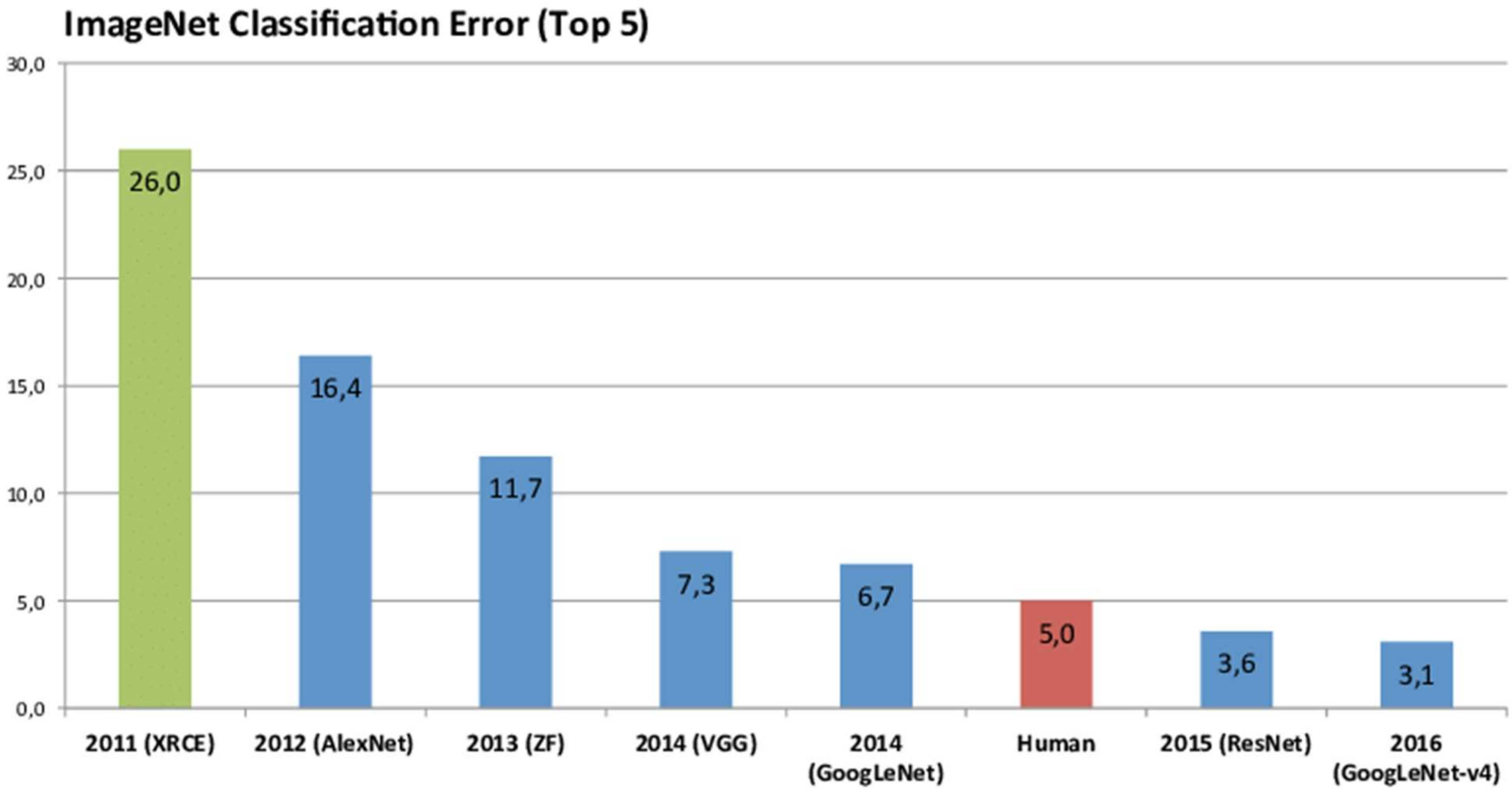


- Deep Supervised Learning (Krizhevsky, Sutskever, Hinton 2012; also LeCun, Bengio, Ng, Darrell, …):



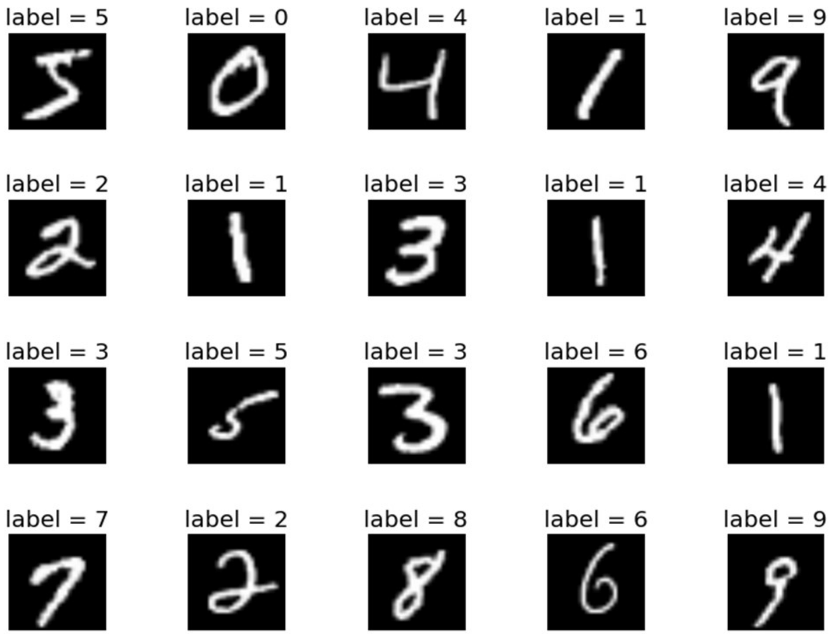
  - ~1.2 million training images from ImageNet [Deng, Dong, Socher, Li, Li, Fei-Fei, 2009]

Machine Learning - Supervised



Machine Learning - Supervised

- One of the main foci of Data Science and Data Mining
- Goal: to learn from imitation
  - Training set with examples

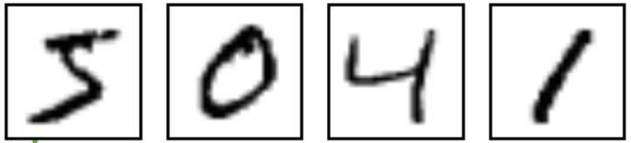


? ? ? ?

  - Minimize the prediction error (a.k.a, the Loss)

Machine Learning - Supervised

- Example:
- Prediction: f(x)=y'
- Loss function:
  L(x) = |y'-y|



10 2 -1 7

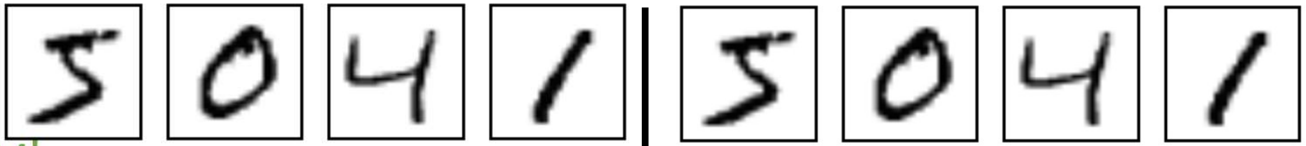
5 0 4 1

- First iteration:
  - L(5)=| 5 -10|=5
  - L(0)=| 0 - 2|=2
  - L(4)=|4-(-1)|=5
  - L(1)=| 1 - 7|=6

- Total loss: 18

Machine Learning - Supervised

- Example:
- Prediction: f(x)=y'
- Loss function:
  L(x) = |y'-y|



10 2 -1 7

9 1 0 6

5 0 4 1

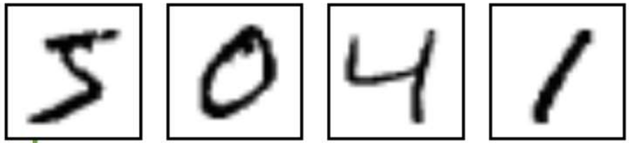
5 0 4 1

- Second iteration:
  - L(5)=|5-9|=4
  - L(0)=|0-1|=1
  - L(4)=|4-0|=4
  - L(1)=|1-6|=5

- Total loss: 14

Machine Learning - Supervised

- Example:
- Prediction: f(x)=y'
- Loss function: L(x) = |y'-y|



10	2	-1	7
9	1	0	6
8	0	1	5

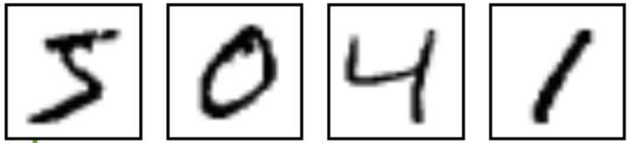
5	0	4	1
5	0	4	1
5	0	4	1

- Third iteration:
  - L(5)=|5-8|=3
  - L(0)=|0-0|=0
  - L(4)=|4-1|=3
  - L(1)=|1-5|=4

- Total loss: 10

Machine Learning - Supervised

- Example:
- Prediction: f(x)=y'
- Loss function:
  L(x) = |y'-y|



10	2	-1	7
9	1	0	6
8	0	1	5
7	0	2	4
6	0	3	2
5	0	4	2
5	0	4	1

5	0	4	1
5	0	4	1
5	0	4	1
5	0	4	1
5	0	4	1
5	0	4	1
5	0	4	1

- n^{th} iteration:
  - L(5)=|5-5|=0
  - L(0)=|0-0|=0
  - L(4)=|4-5|=0
  - L(1)=|1-1|=0

- Total loss: 0

Machine Learning - Supervised

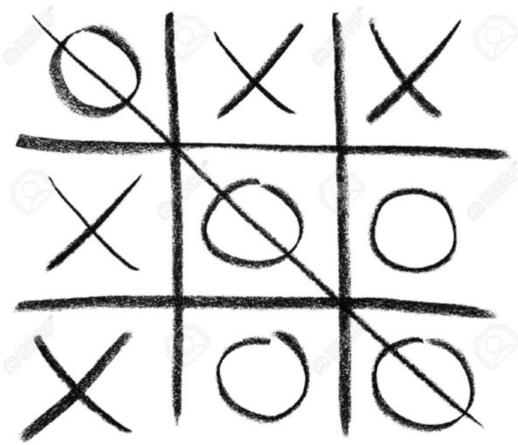
- Requires knowing many answers to your question
  - Sometimes not possible
- Data can be shaped (new features), normalized (scaling), cleaned (outliers), improved (redundancy)
- Bias / Variance
- Amount of training data
- Dimensionality
- Noise
- Complexity

Chihuahua or Muffin?

Croissant or Dog?

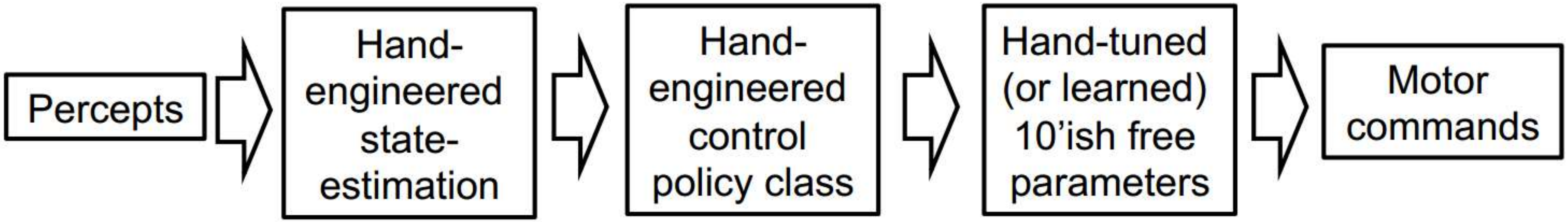
Machine Learning - Reinforcement

- Also known as Reward-based Learning
- Goal: to learn from experiments
  - No data set, just interactions with the environment
  - Interactions will give rewards
    - +1 for winning
    - -1 for losing
  - Maximize the reward

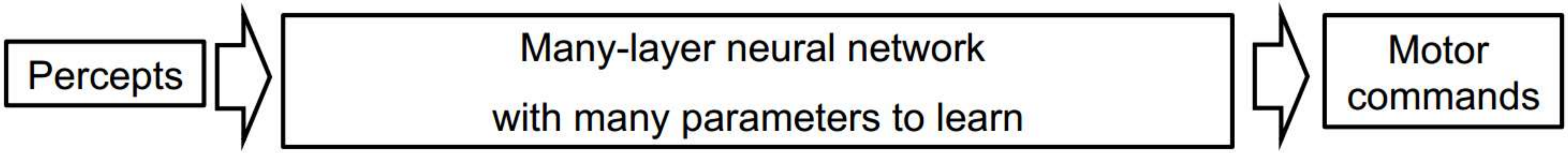


Machine Learning - Reinforcement

- **Current state-of-the-art robotics**

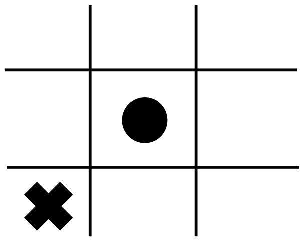


- **Deep reinforcement learning**

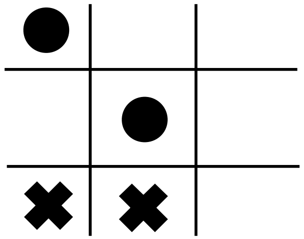


Machine Learning - Reinforcement

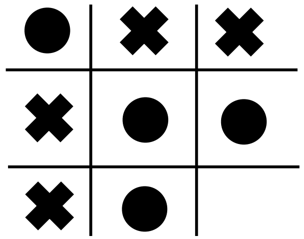
- Example:
  - Tic-tac-toe



a



b

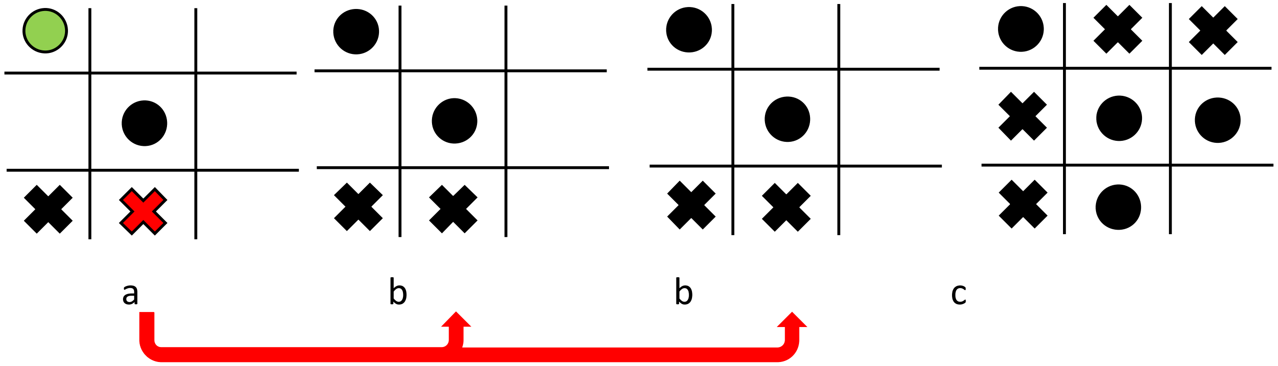


c

- State **a**. We don't know much about it, game could go either way.
  - Reward: Unknown. Depends on following states.
- State **b**. We can either win or lose.
  - Reward: -1 points if we lose, or 1 point if we win.
- State **c**. We tie.
  - Reward: 0 points.

Machine Learning - Reinforcement

- Example:

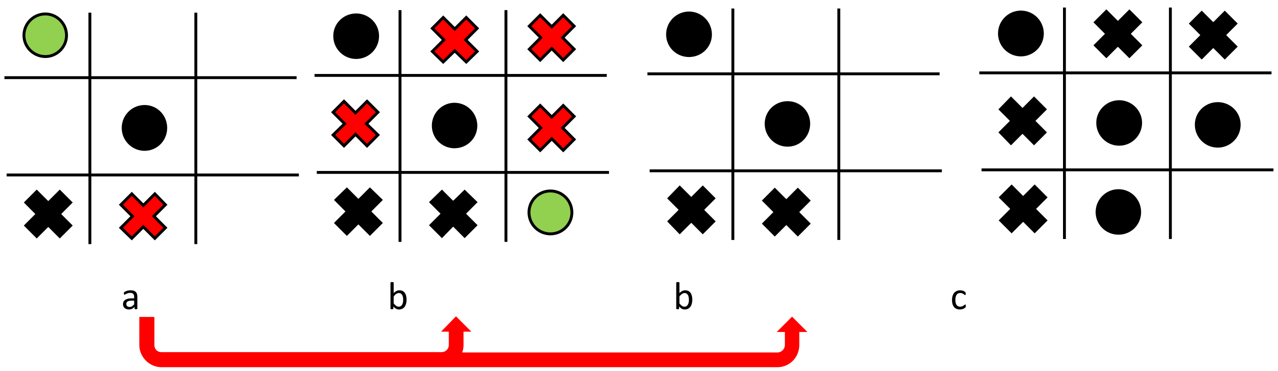


- Value function:
  - $V(x_t)=R+V(x_{t+1})$

- At state **a**, we try action. Opponent then always plays .
  - Reward: 0 points, plus the points of the next state.
- This leads to state **b**. We can either win on our move, or lose if we let opponent play again.
  - Reward: -1 points if we lose, or 1 point if we win.
- We also have an unrelated state **c**, where we tie.
  - Reward: 0 points.

Machine Learning - Reinforcement

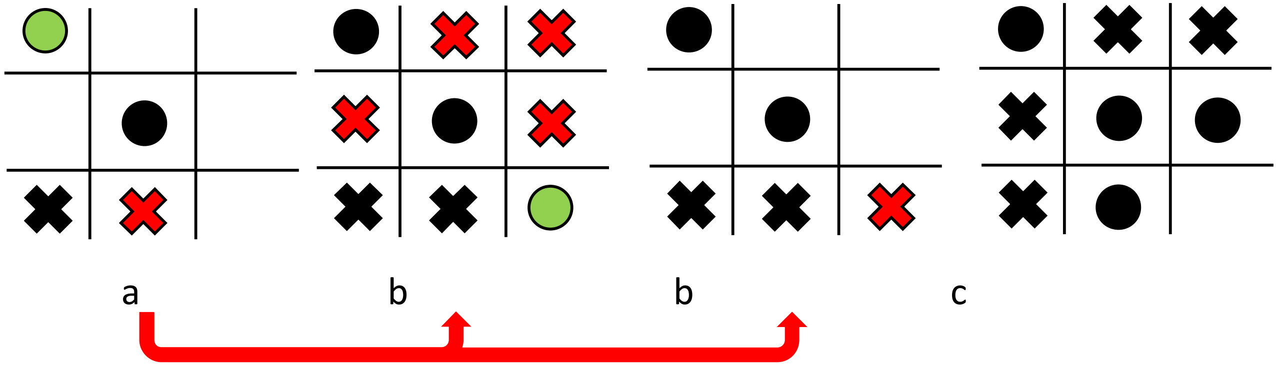
- Example:
- Value function:
  - $V(x_t)=R+V(x_{t+1})$



- At state **a**, we try ✖ action. Opponent then always plays ● .
  - Reward: 0 points, plus the points of the next state.
- This leads to state **b**. We can either win on our move, or lose if we let opponent play again.
  - Reward: -1 points if we lose, or 1 point if we win.
- We also have an unrelated state **c**, where we tie.
  - Reward: 0 points.

Machine Learning - Reinforcement

- Example:

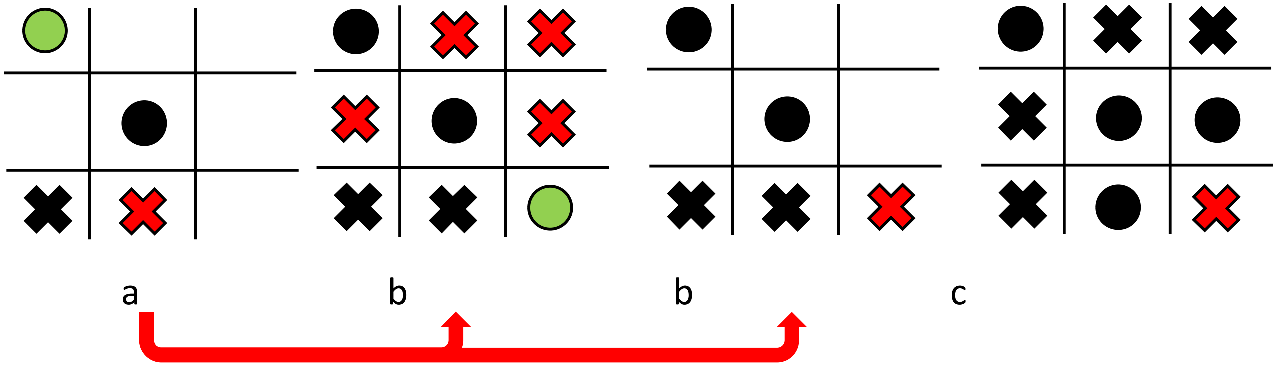


- Value function:
  - $V(x_t)=R+V(x_{t+1})$

- At state **a**, we try action. Opponent then always plays .
  - Reward: 0 points, plus the points of the next state.
- This leads to state **b**. We can either win on our move, or lose if we let opponent play again.
  - Reward: -1 points if we lose, or 1 point if we win.
- We also have an unrelated state **c**, where we tie.
  - Reward: 0 points.

Machine Learning - Reinforcement

- Example:

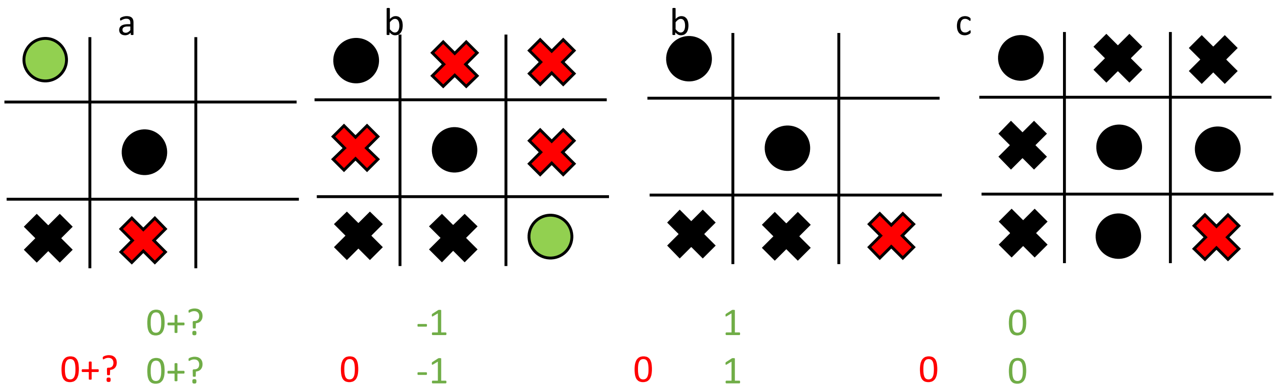


- Value function:
  - $V(x_t)=R+V(x_{t+1})$

- At state **a**, we try action. Opponent then always plays .
  - Reward: 0 points, plus the points of the next state.
- This leads to state **b**. We can either win on our move, or lose if we let opponent play again.
  - Reward: -1 points if we lose, or 1 point if we win.
- We also have an unrelated state **c**, where we tie.
  - Reward: 0 points.

Machine Learning - Reinforcement

- Example:



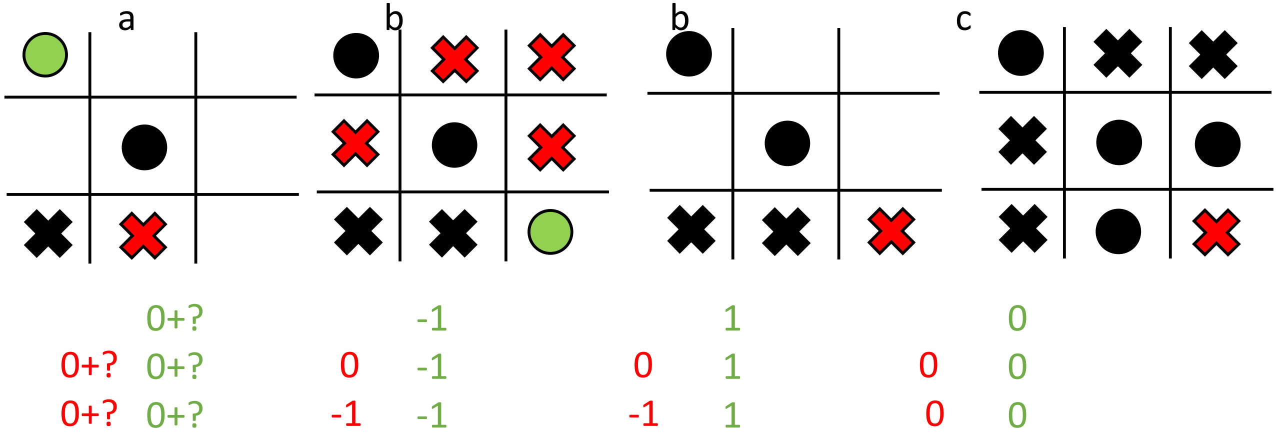
- Reward function:
  - $V(x_t)=R+V(x_{t+1})$

- First iteration:
  - V(a)=0+V(b)=0
  - V(b)=0
  - V(b)=0
  - V(c)=0

- All actions seem equally good

Machine Learning - Reinforcement

- Example:

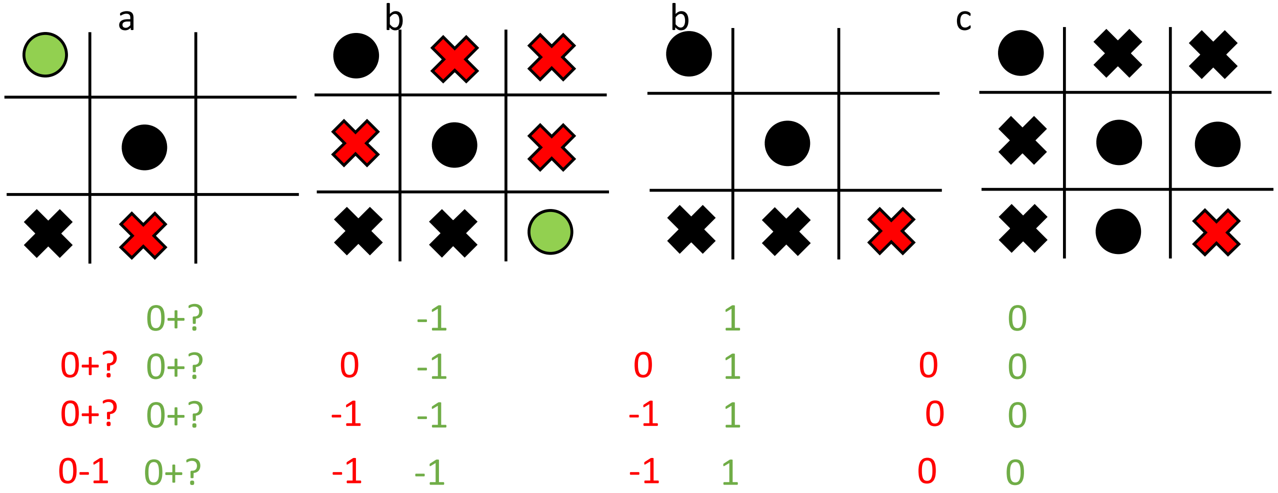


- Reward function:
  - $V(x_t)=R+V(x_{t+1})$

- Second iteration:
  - V(a)= 0+V(b)
  - V(b)=-1
  - V(b)=-1
  - V(c)= 0

Machine Learning - Reinforcement

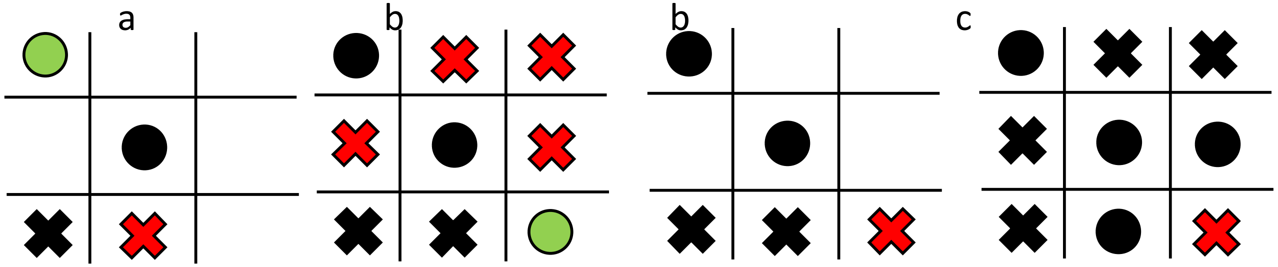
- Example:



- Reward function:
  - $V(x_t)=R+V(x_{t+1})$
- Third iteration:
  - V(a)= 0+V(b)=-1
  - V(b)=-1
  - V(b)=-1
  - V(c)= 0
- Avoid going into state "b", it looks bad!

Machine Learning - Reinforcement

- Example:



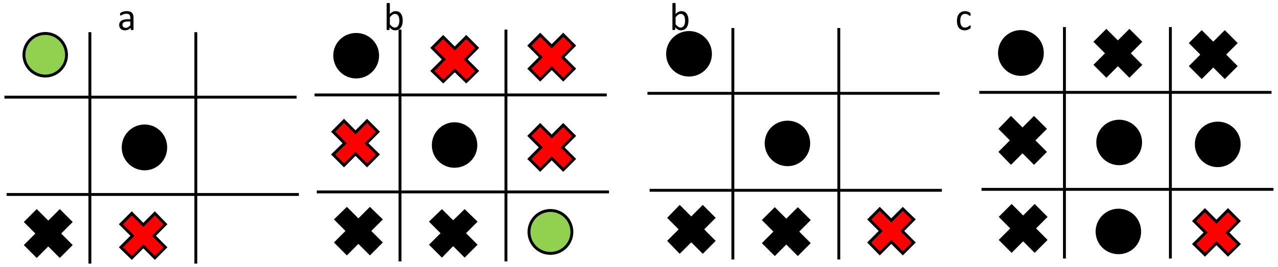
	0+?		-1		1		0
0+?	0+?	0	-1	0	1	0	0
0+?	0+?	-1	-1	-1	1	0	0
0-1	0+?	-1	-1	-1	1	0	0
0+?	0+?	-1	-1	1	1	0	0

- Reward function:
  - $V(x_t)=R+V(x_{t+1})$

- Fourth iteration:
  - V(a)= 0+V(b)
  - V(b)=-1
  - V(b)= 1
  - V(c)= 0

Machine Learning - Reinforcement

- Example:



- Reward function:
  - $V(x_t)=R+V(x_{t+1})$

	0+?		-1		1		0
0+?	0+?	0	-1	0	1	0	0
0+?	0+?	-1	-1	-1	1	0	0
0-1	0+?	-1	-1	-1	1	0	0
0+?	0+?	-1	-1	1	1	0	0
0+1	0+?	-1	-1	1	1	0	0

- Last iteration:
  - V(a)= 0+V(b)=1
  - V(b)=-1
  - V(b)= 1
  - V(c)= 0

- State "b" is good afterall, and "a" as well!

Machine Learning - Reinforcement

- Requires trying many (all) possible moves
  - Sometimes not possible or takes too long

- Game rewards numerically related, future rewards importance must be defined, penalties might be necessary

- Exploration / Exploitation
- Amount of training data
- Multi-agent learning
- Noise
- Complexity

Machine Learning - Comparison

- **Supervised**

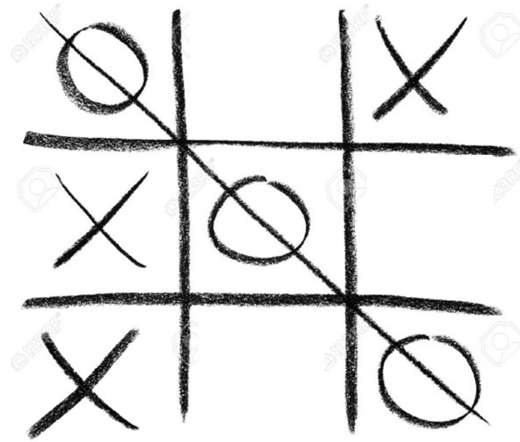
- Input: image of a digit
- Output: digit

- Maximize accuracy (by finding prediction function with minimum error)

- **Reinforcement**

- Input: image of the game state
- Output: action

- Maximize reward (by finding value function with minimum error)

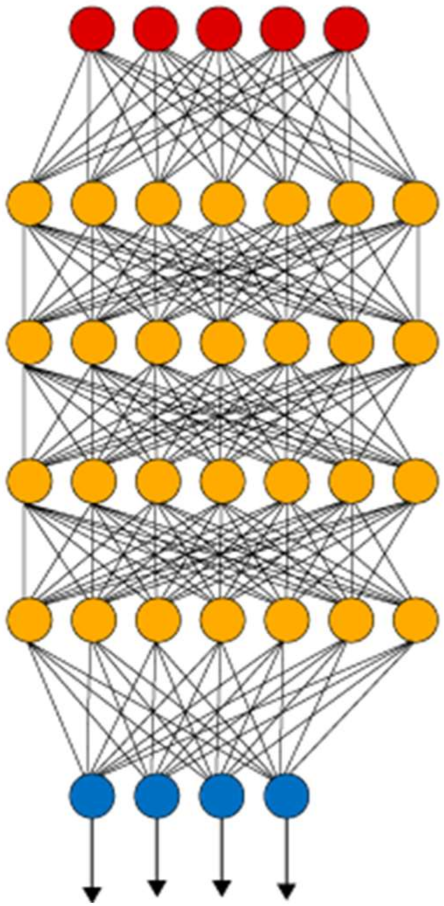


Neural Networks

- Neural Networks
  - Buzz-word
  - Solves everything apparently
  - Deep Learning

- Resources:
  - Deep Learning Course, by Google (Coursera)
  - Deep Learning Specialization Course, by Udacity
  - Deep Learning, by Goodfellow, Bengio & Courtville
  - Lectures on Deep Learning, by Palazzi

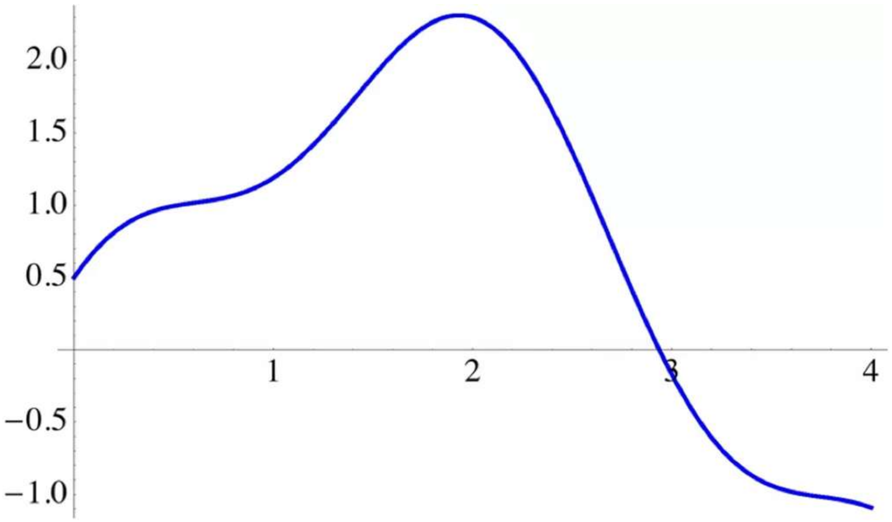
- Algorithm that can approximate any function
  - But what does this even mean?

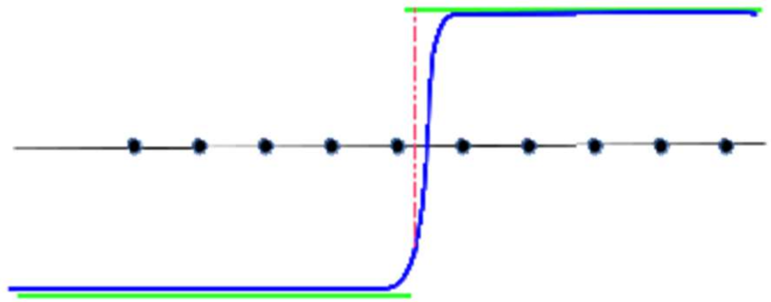


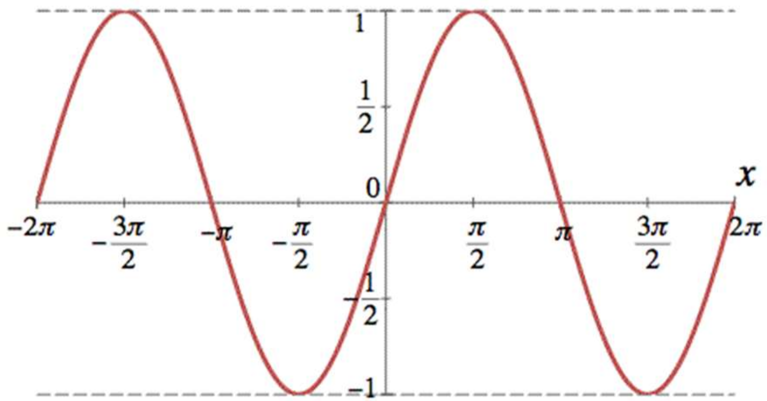
Neural Networks - Theory

- A deep NN with arbitrarily high complexity can approximate any function with arbitrarily high precision

$$f(x) = y, \quad x \in R^1, \quad y \in R^1$$

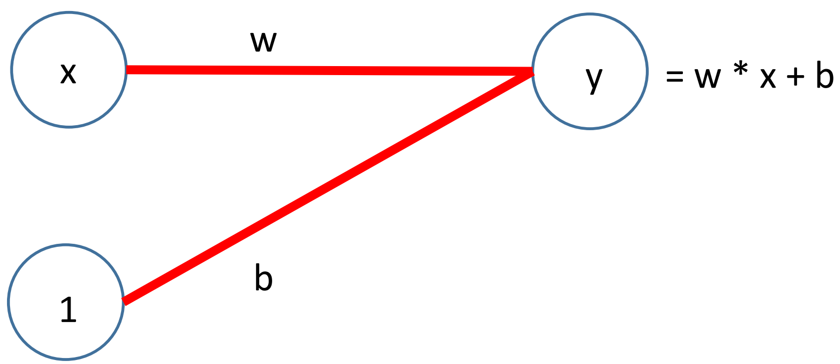






Neural Networks - Theory

- But how? $x \in R^1$, $y \in R^1$, no hidden layers



The simplest network.

Just an input x, an output y.

There is usually a bias term, like a weight that is multiplied by 1 and added.

What should be the values of w and b to make this network work like:

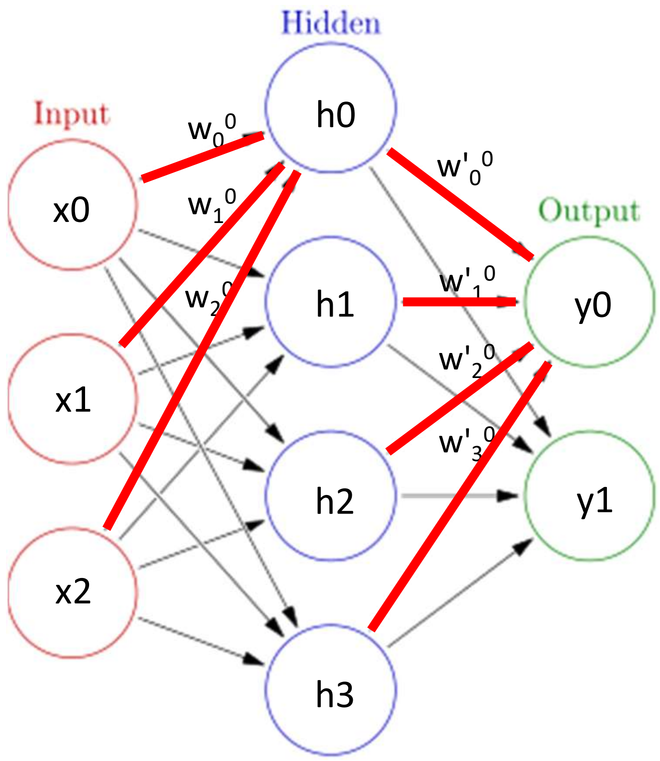
$y = x$

$y = (1-x)$

$y = 2x-1$

Neural Networks - Theory

- But how? $x \in R^3$, $y \in R^2$, a 4-wide hidden layer



A network is basically many matrices of weights which are multiplied by vectors of nodes.

Layers have activation functions, g(a), we'll come back to those. An input is given, and the hidden nodes are calculated.

$$h^j = g[\sum_i(w_i^j\, x_i) + b^j]$$
$$h^0 = g[\,(w_0^0\, x_0) + (w_1^0\, x_1) + (w_2^0\, x_2) + b^0]$$

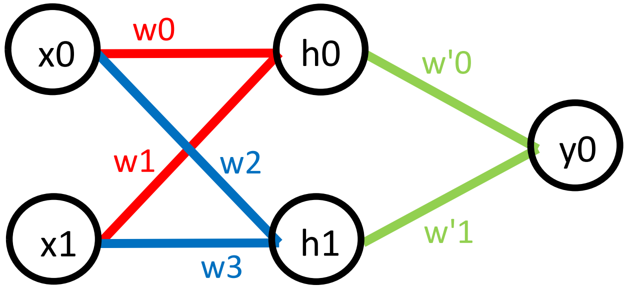
After the hidden nodes have values, we calculate the output nodes. The output layer might not have an activation function.

$$y^j = \sum(w'^j_i\, h_i + b'^j)$$
$$y^0 = (w'^0_0\, h_0) + (w'^0_1\, h_1) + (w'^0_2\, h_2) + (w'^0_3\, h_3) + b'^0$$

Neural Networks - Theory

- Simple practical example:

$x \in R^2$, $y \in R^1$, a 2-wide hidden layer



Given inputs:
(x0, x1) = (1, 2)

And weights:
(w0, w1, b0) = (0.5, 1, 0)
(w2, w3, b1) = (1, 0.5, 0)
(w'0, w'1, b') = (1, -1, 0)

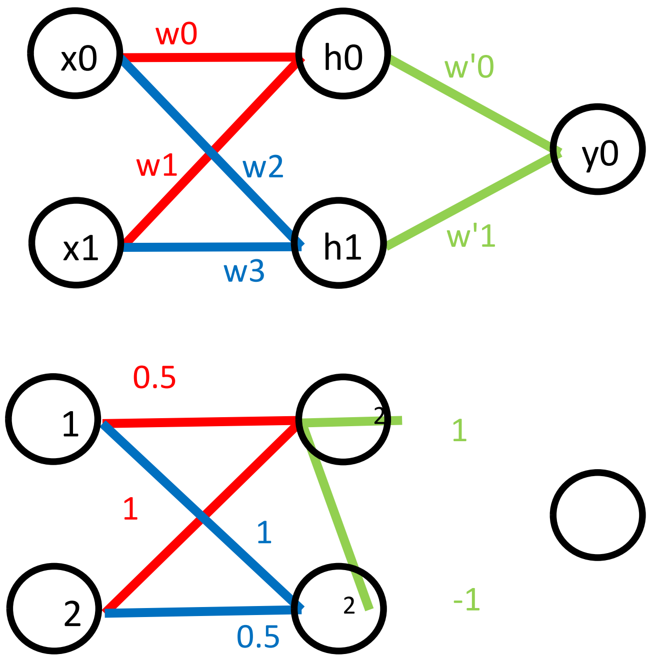
When the layer uses a quadratic activation function:
$g(a) = a^2$

What is the output y0?
Remember:
$h = g[\sum(w*x) + b]$
$y = \sum(w'*h + b')$

Neural Networks - Theory

- Simple practical example:

x Є R^2, y Є R^1, a 2-wide hidden layer



Given inputs:
(x0, x1) = (1, 2)

And weights:
(w0, w1, b0) = (0.5, 1, 0)
(w2, w3, b1) = (1, 0.5, 0)
(w'0, w'1, b') = (1, -1, 0)

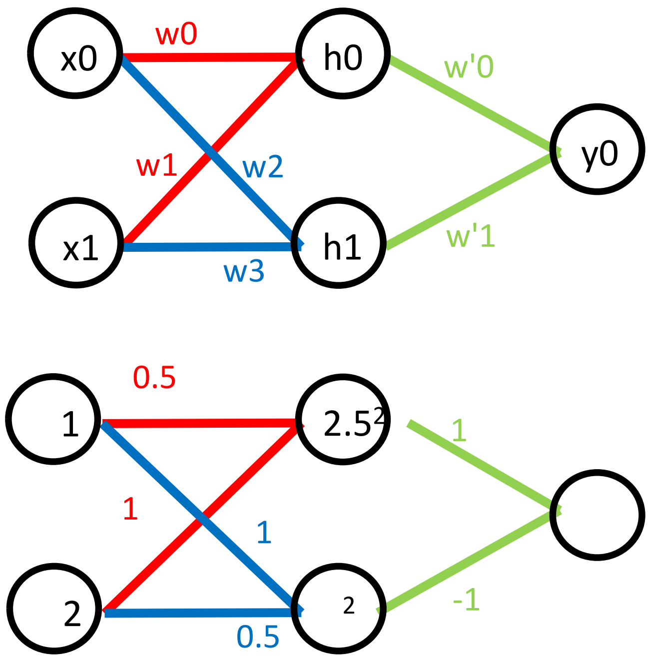
When the layer uses a quadratic activation function:
$g(a) = a^2$

What is the output y0?
Remember:
$h = g[\sum(w*x) + b]$
$y = \sum(w'*h + b')$

Neural Networks - Theory

- Simple practical example:

$x \in R^2$, $y \in R^1$, a 2-wide hidden layer



Given inputs:
(x0, x1) = (1, 2)

And weights:
(w0, w1, b0) = (0.5, 1, 0)
(w2, w3, b1) = (1, 0.5, 0)
(w'0, w'1, b') = (1, -1, 0)

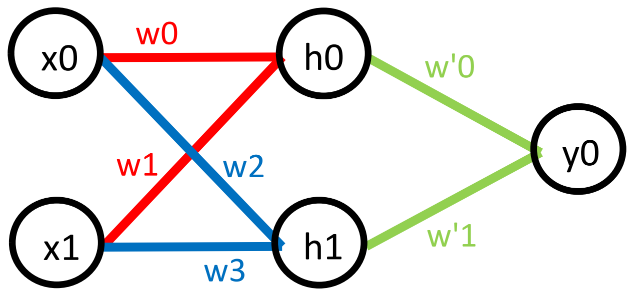
When the layer uses a quadratic activation function:
$g(a) = a^2$

What is the output y0?
Remember:
$h = g[\sum(w*x) + b]$
$y = \sum(w'*h + b')$

Neural Networks - Theory

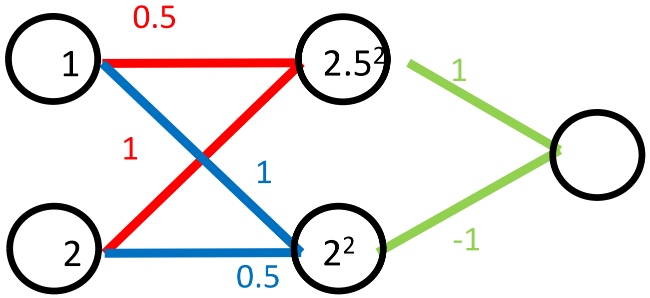
- Simple practical example:

$x \in R^2$, $y \in R^1$, a 2-wide hidden layer



Given inputs:
(x0, x1) = (1, 2)

And weights:
(w0, w1, b0) = (0.5, 1, 0)
(w2, w3, b1) = (1, 0.5, 0)
(w'0, w'1, b') = (1, -1, 0)



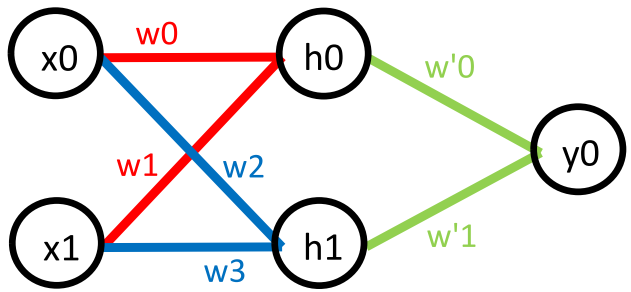
When the layer uses a quadratic activation function:
$g(a) = a^2$

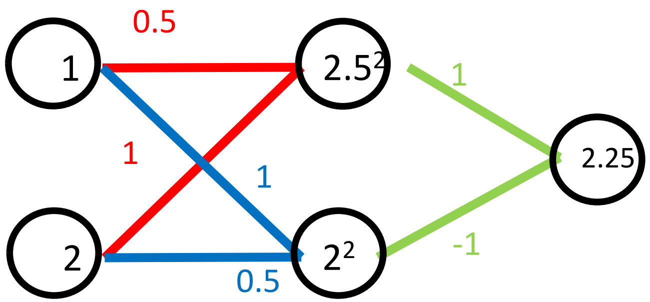
What is the output y0?
Remember:
$h = g[\sum(w*x) + b]$
$y = \sum(w'*h + b')$

Neural Networks - Theory

- Simple practical example:

$x \in R^2$, $y \in R^1$, a 2-wide hidden layer





Given inputs:
(x0, x1) = (1, 2)

And weights:
(w0, w1, b0) = (0.5, 1, 0)
(w2, w3, b1) = (1, 0.5, 0)
(w'0, w'1, b') = (1, -1, 0)

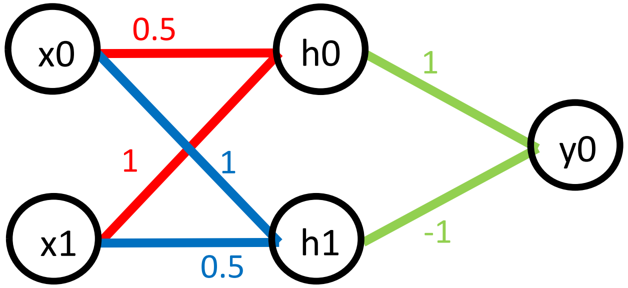
When the layer uses a quadratic activation function:
$g(a) = a^2$

What is the output y0?
Remember:
$h = g[\sum(w*x) + b]$
$y = \sum(w'*h + b')$

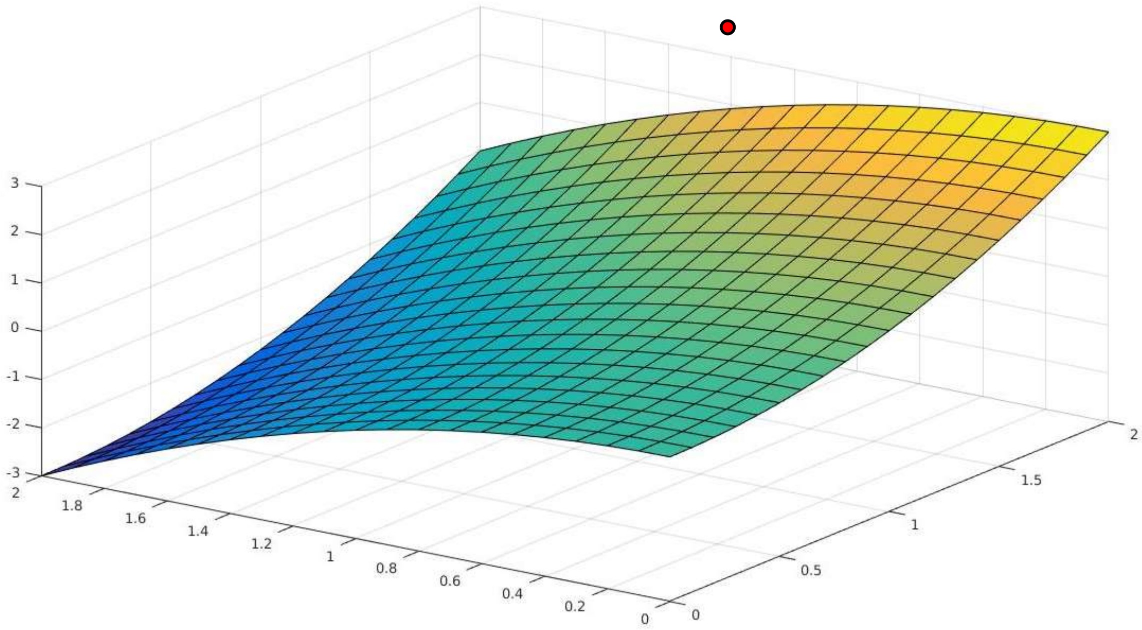
Neural Networks - Theory

- Simple practical example:

$x \in R^2$, $y \in R^1$, a 2-wide hidden layer



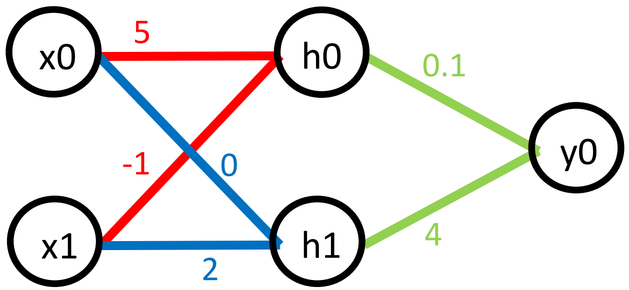
With these weights, this is the function that this NN outputs. By changing the weights, different functions are found.



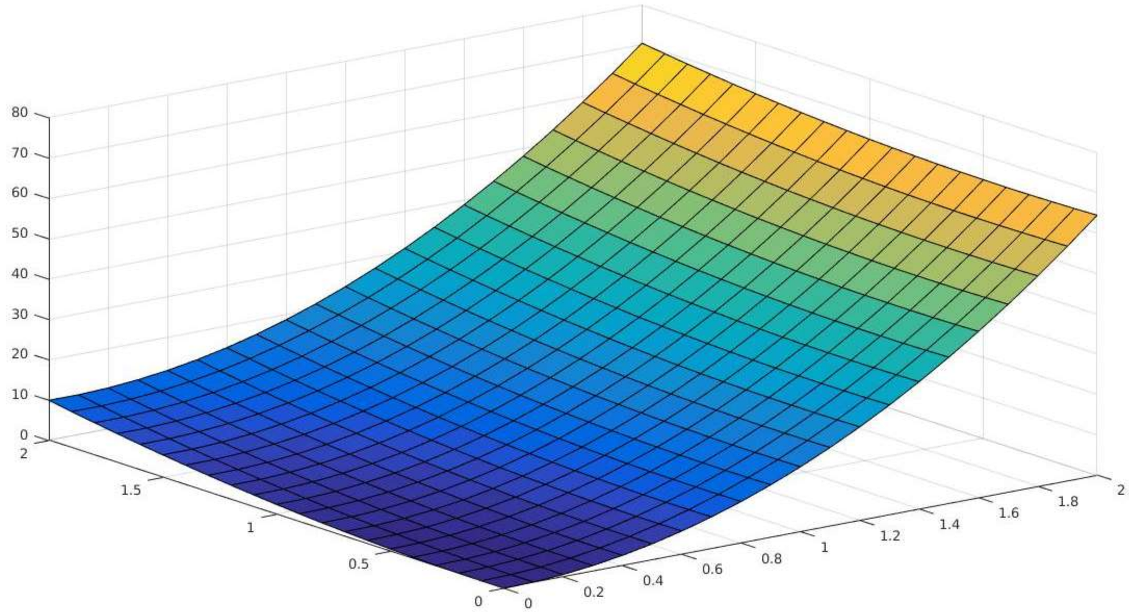
Neural Networks - Theory

- Simple practical example:

$x \in R^2$, $y \in R^1$, a 2-wide hidden layer



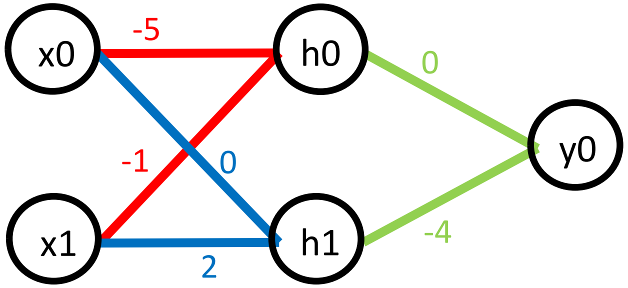
With these weights, this is the function that this NN outputs. By changing the weights, different functions are found.



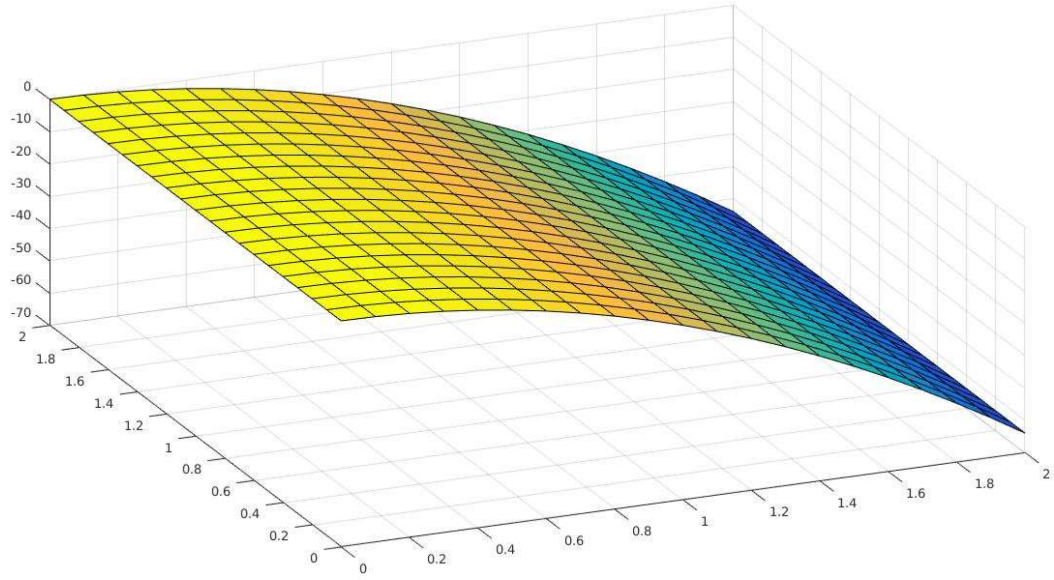
Neural Networks - Theory

- Simple practical example:

x ∈ R^2, y ∈ R^1, a 2-wide hidden layer



With these weights, this is the function that this NN outputs. By changing the weights, different functions are found.



Neural Networks - Theory

So the question now becomes:

- Given some network and a target function, how do we determine these weights to approximate this function?

Neural Networks - Theory

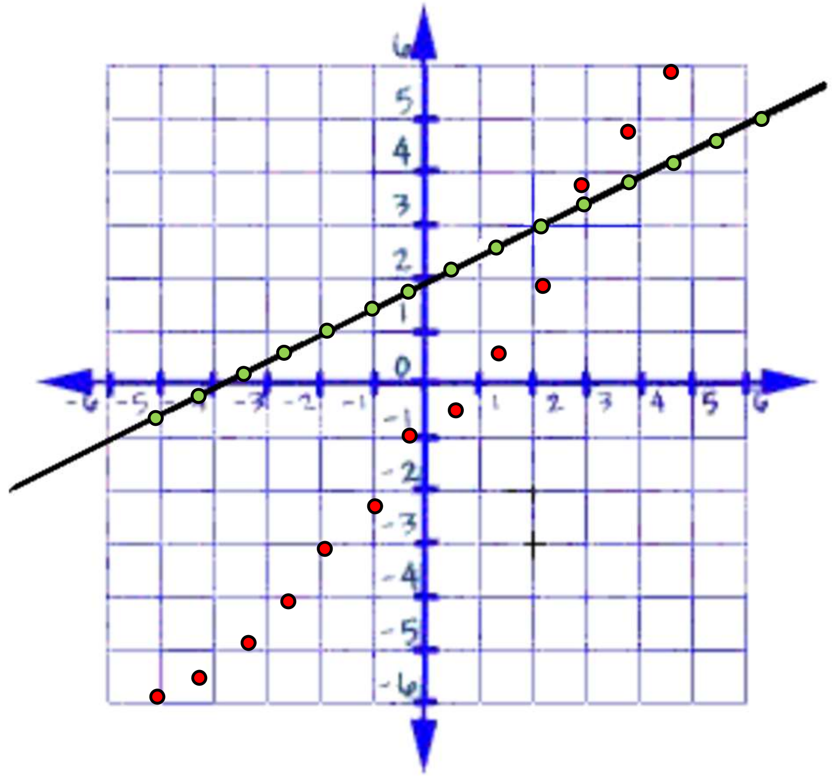
So the question now becomes:

- Given some network and a target function, how do we determine these weights to approximate this function?

BACKPROPAGATION

Neural Networks - Theory

- Backpropagation - Minimizes a loss function L



- Consider a line equation. We have a dataset of points on that line (green)
- Our network starts with random weights, and outputs random points (red)
- We want the NN to approximate the line function
- Our loss function measures the error of our network
    - What are possible loss functions we could use?

Neural Networks - Theory

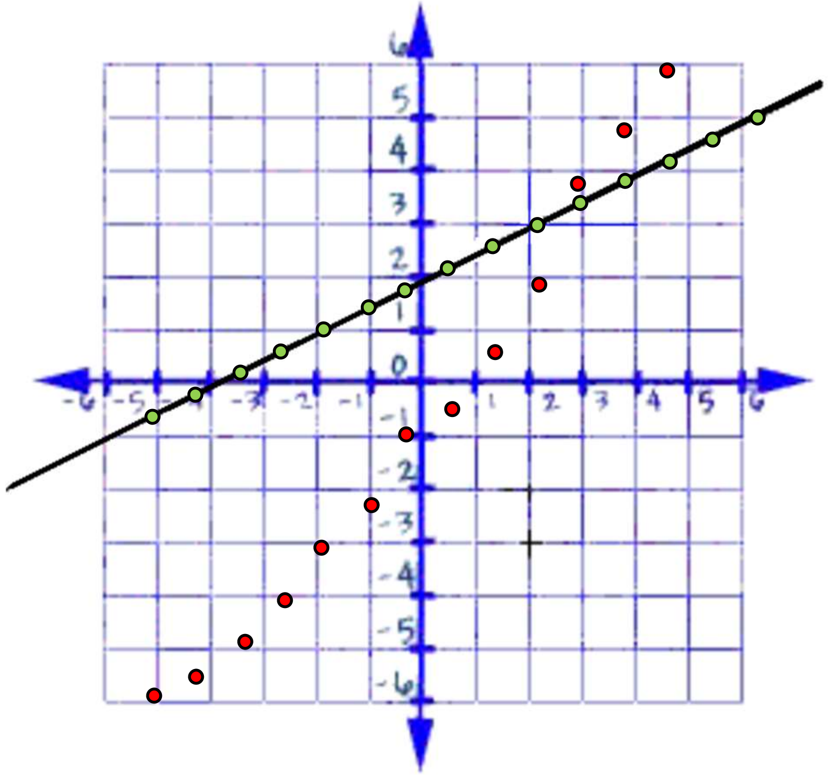
- Backpropagation - Minimizes a loss function L

- The absolute or squared difference between our known outputs (the ones on the training set) and the network outputs

$$L(y,y') = \sum(|y' - y|)$$

$$L(y,y') = \sum((y' - y)^2)$$

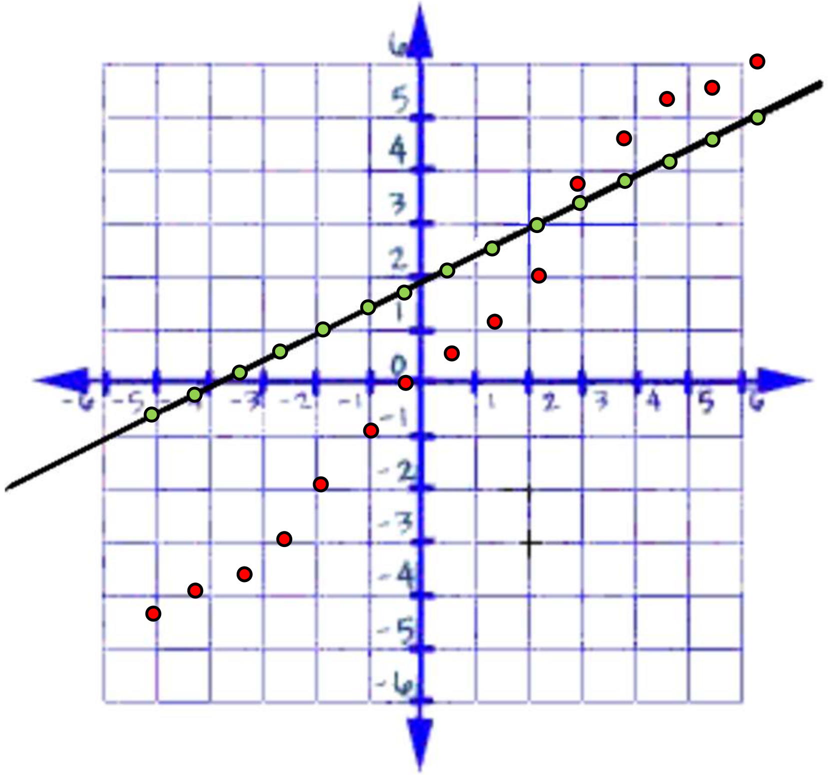
- Globally continuous
- Differentiable



Neural Networks - Theory

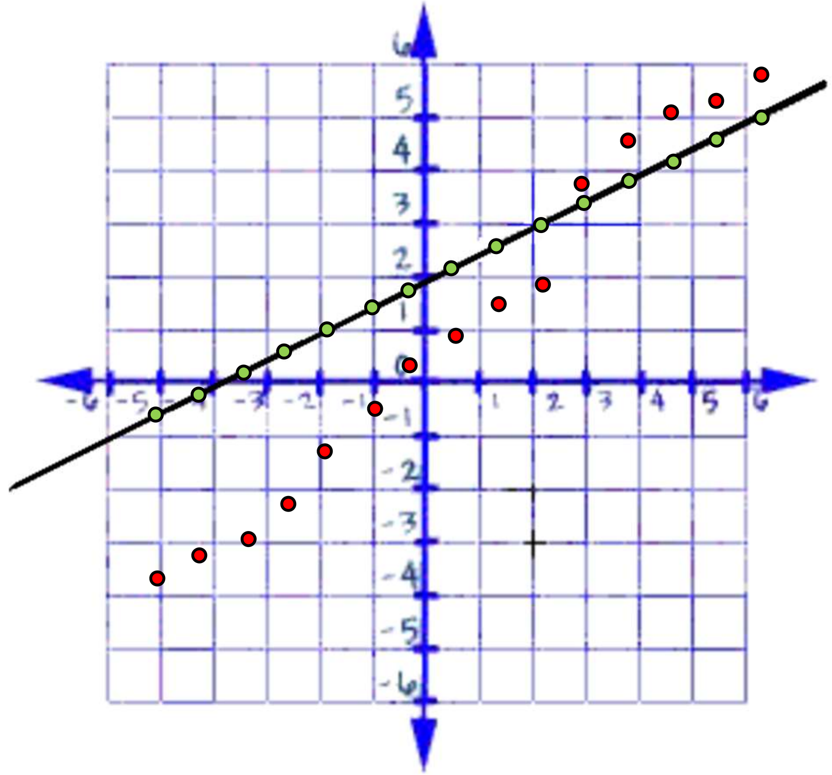
- Backpropagation - Minimizes a loss function L

- As we minimize the loss function, we get better and better results...



Neural Networks - Theory

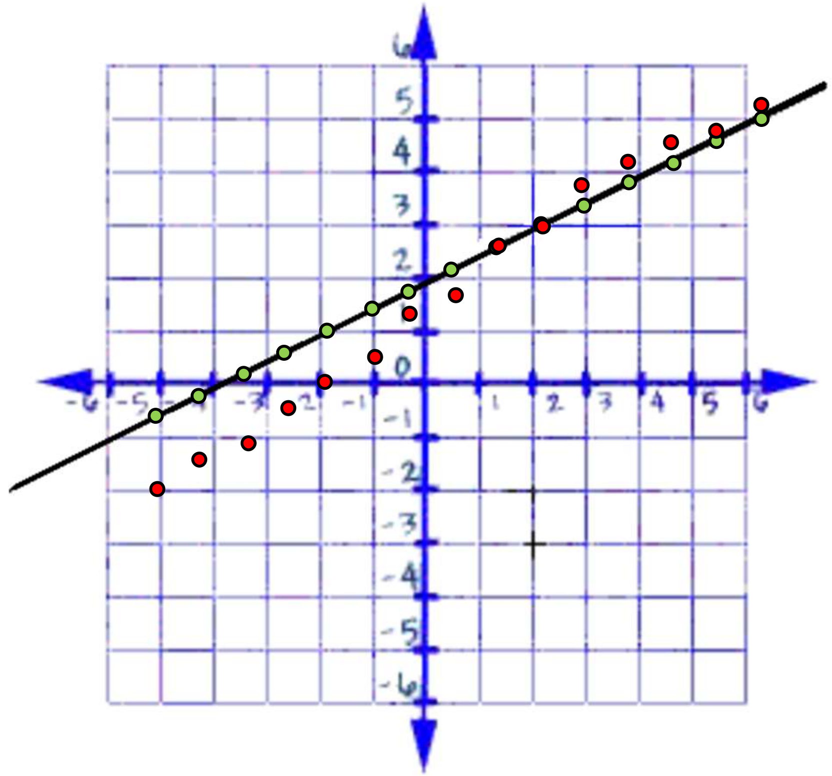
- Backpropagation - Minimizes a loss function L
  - As we minimize the loss function, we get better and better results...



Neural Networks - Theory

- Backpropagation - Minimizes a loss function L

- As we minimize the loss function, we get better and better results...

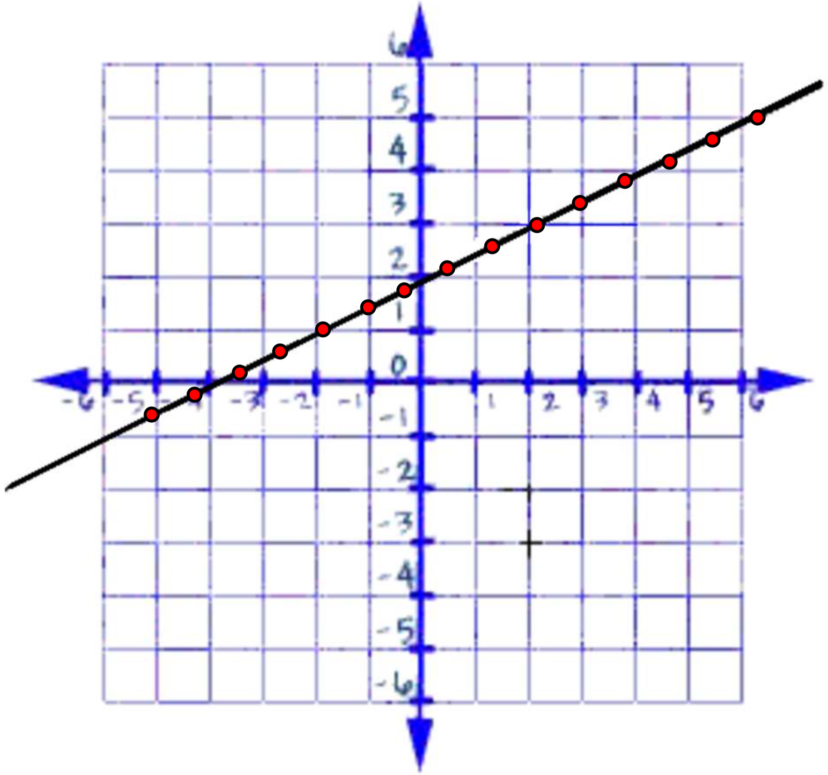


Neural Networks - Theory

- Backpropagation - Minimizes a loss function L

- As we minimize the loss function, we get better and better results...

...until our loss function is 0 (or close)



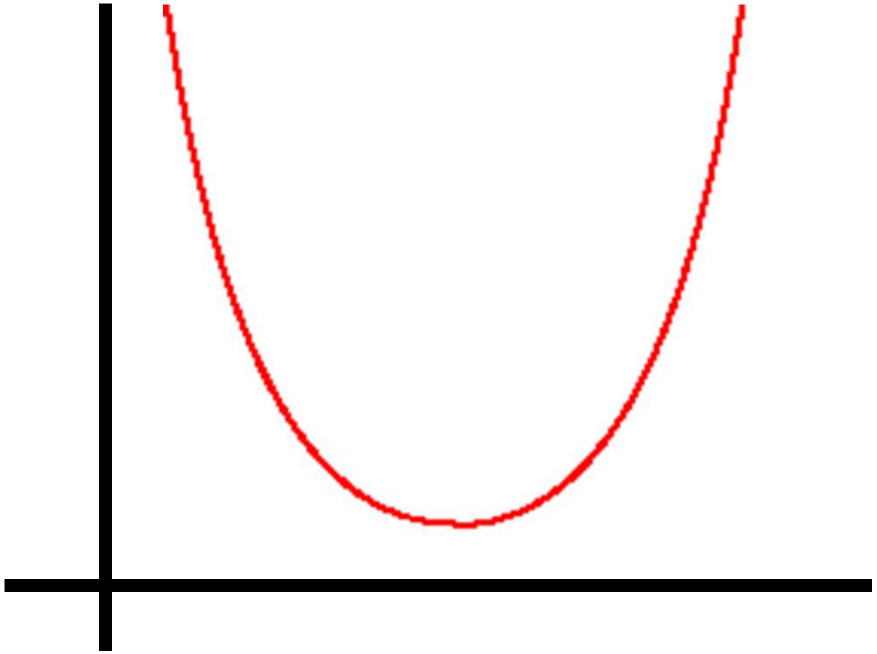
Neural Networks - Theory

- Backpropagation - Minimizes a loss function L

- As we minimize the loss function, we get better and better results...

...until our loss function is 0 (or close)

- Some loss functions don't approximate 0
- Noise also makes minimizing harder

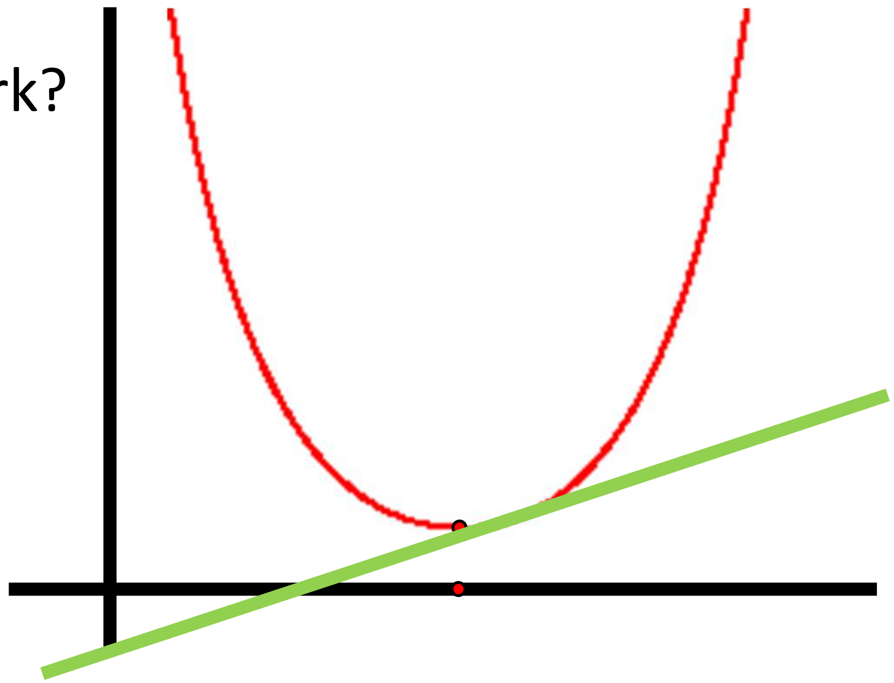


Neural Networks - Theory

- Backpropagation - Minimizes a loss function L using partial derivatives

- How does minimization actually work?
  - (Partial) Derivatives!

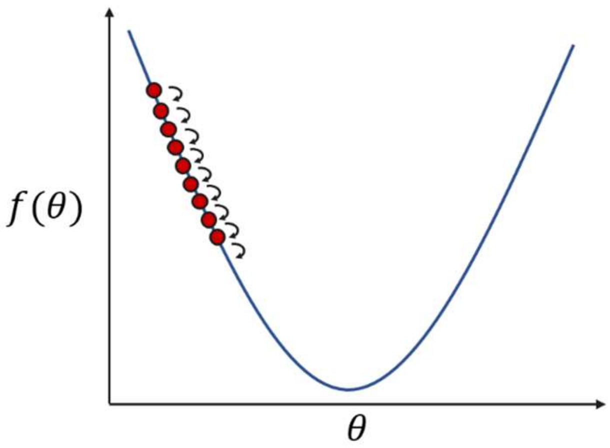
By finding the point where L'(y,y')=0, we find the minimum of L(y,y'). The partial derivates of each weight tell us how much that weight contributed to the error.



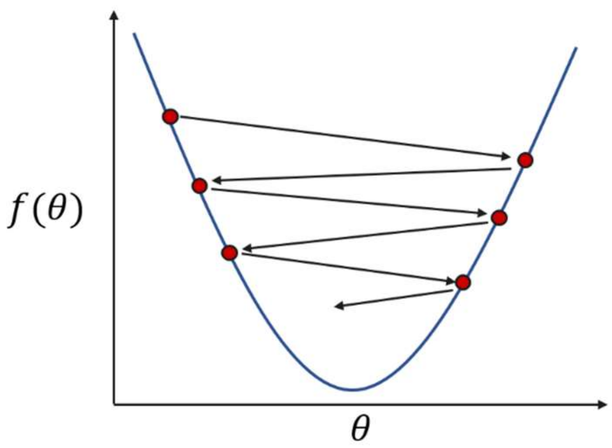
Neural Networks - Theory

- Backpropagation - Minimizes a loss function L using partial derivatives
- We shift each weight a small amount based on its partial derivative

- Amount depends on the Learning Rate (a hyper parameter)



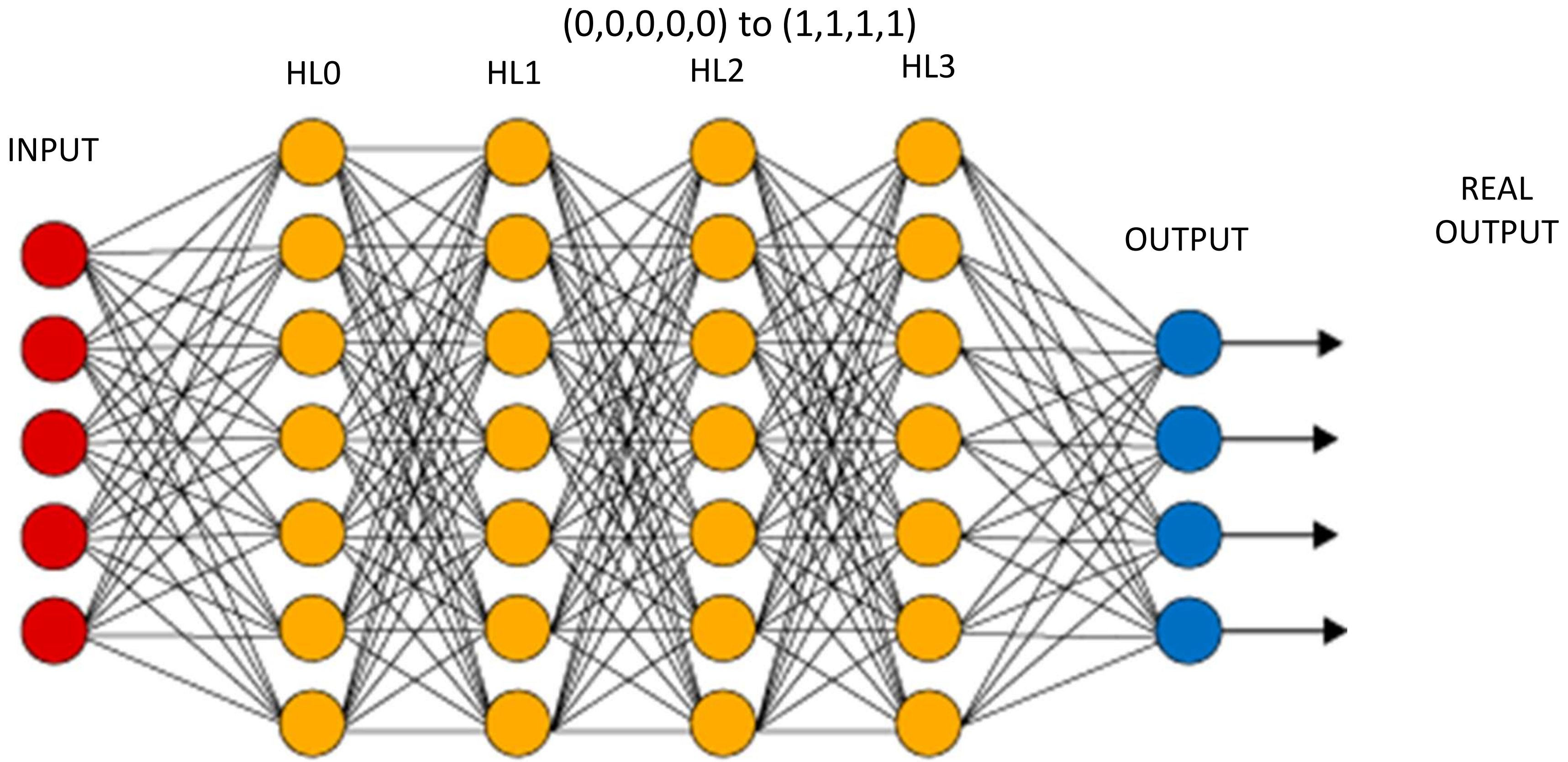
Learning Rate too small



Learning Rate too big

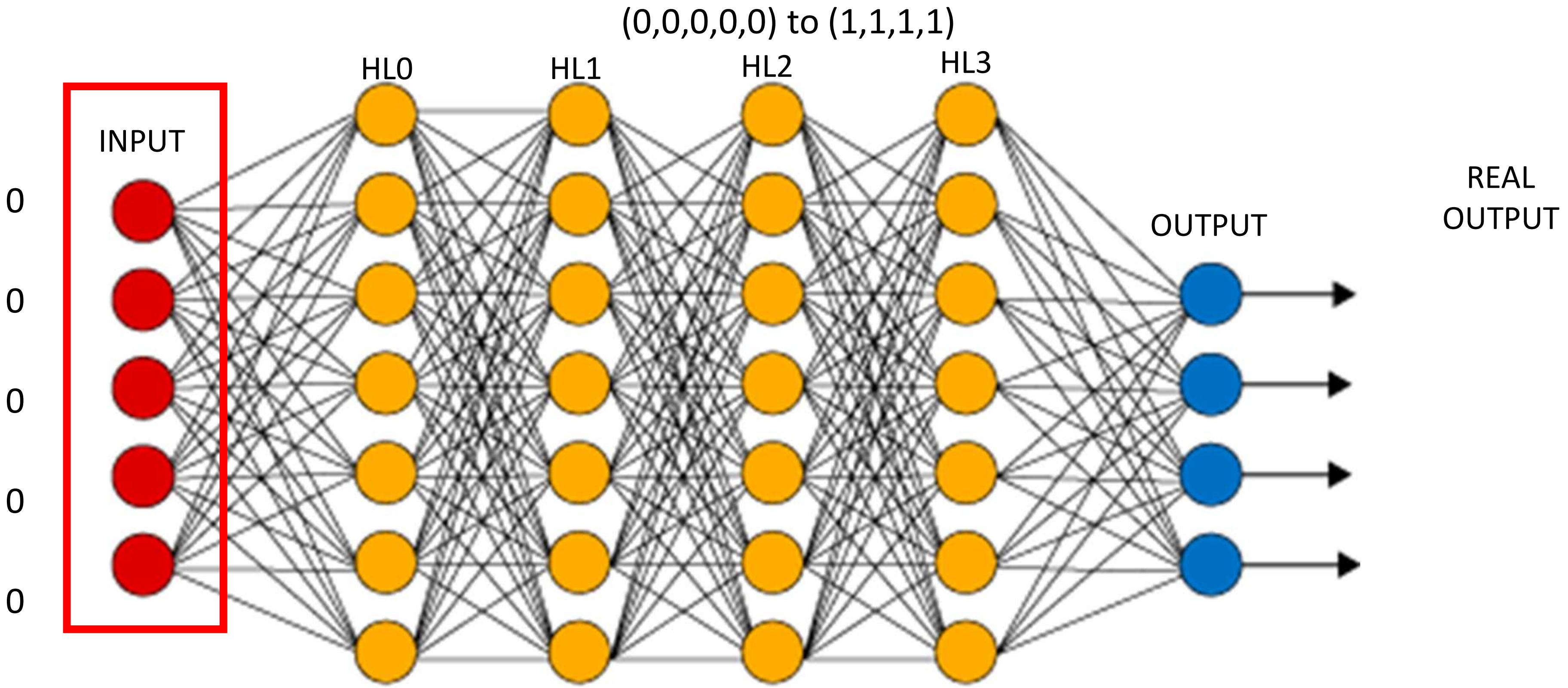
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



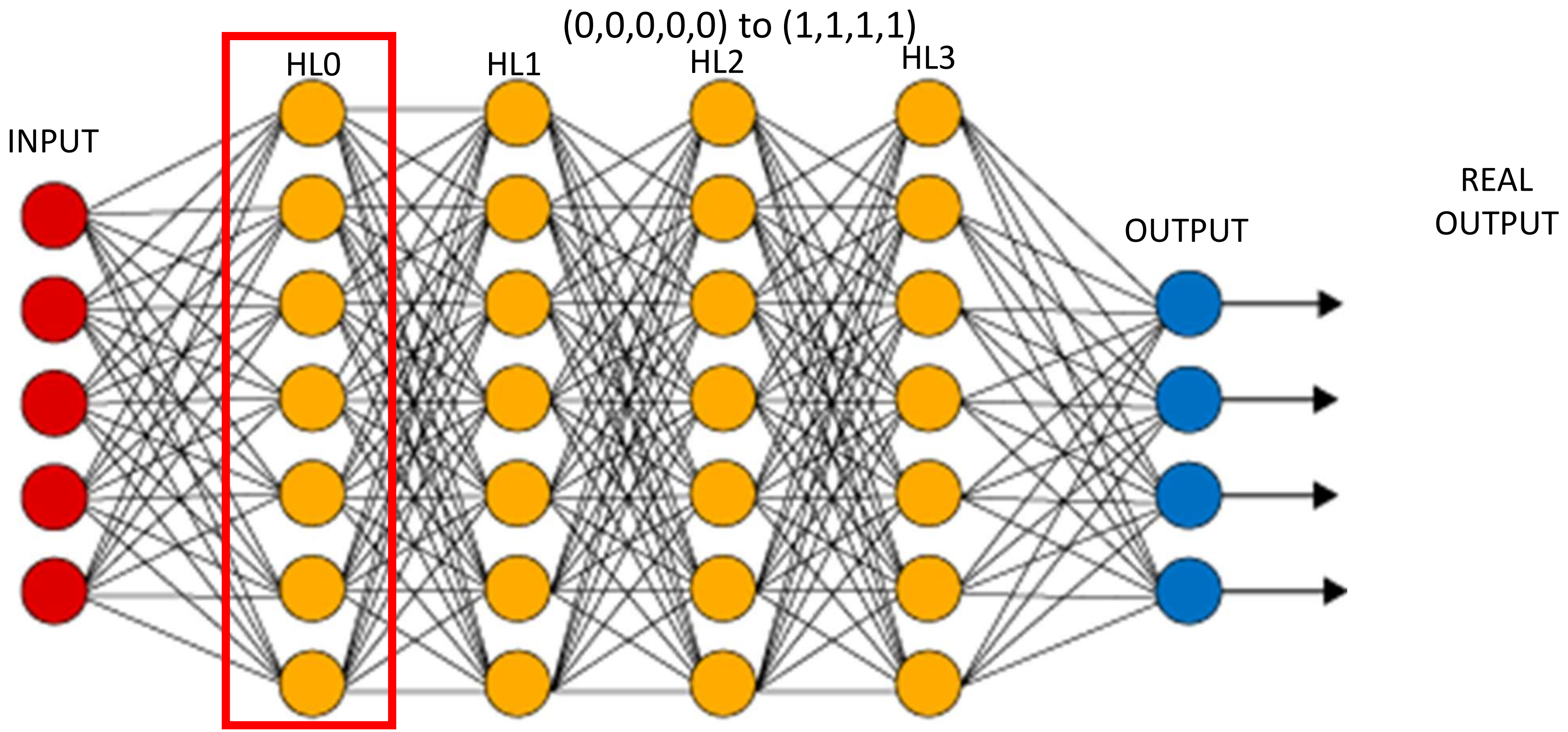
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



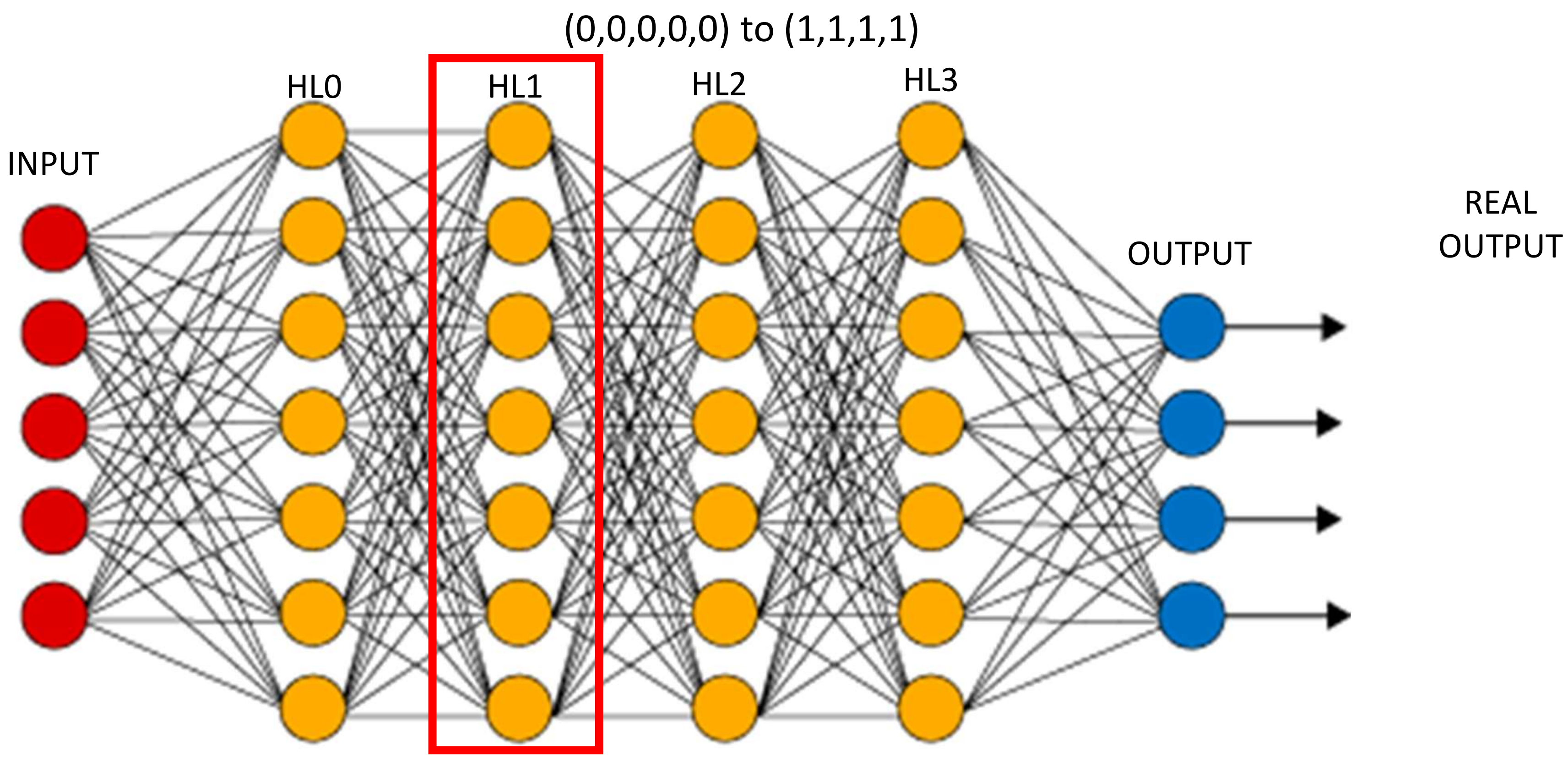
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



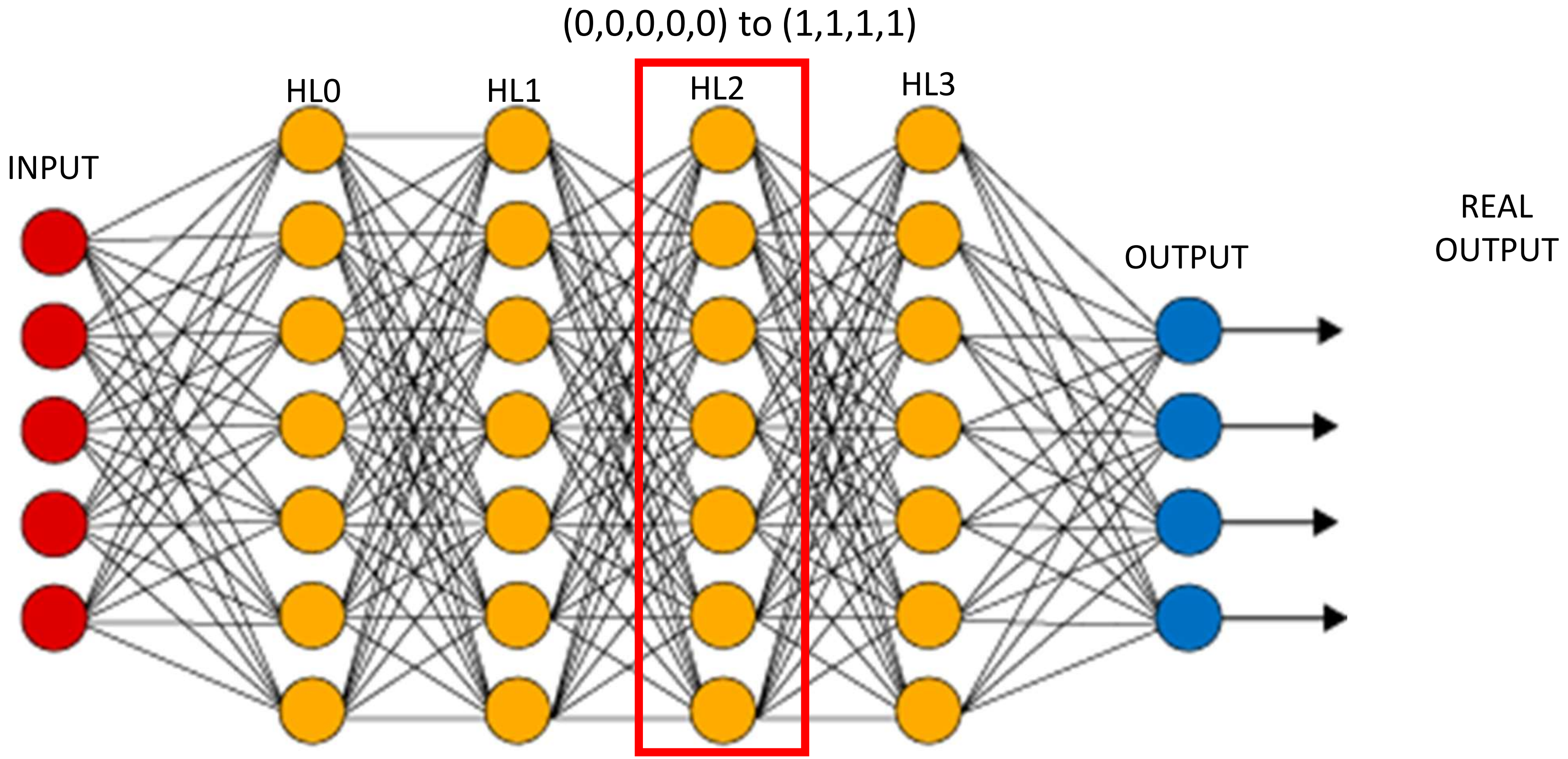
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



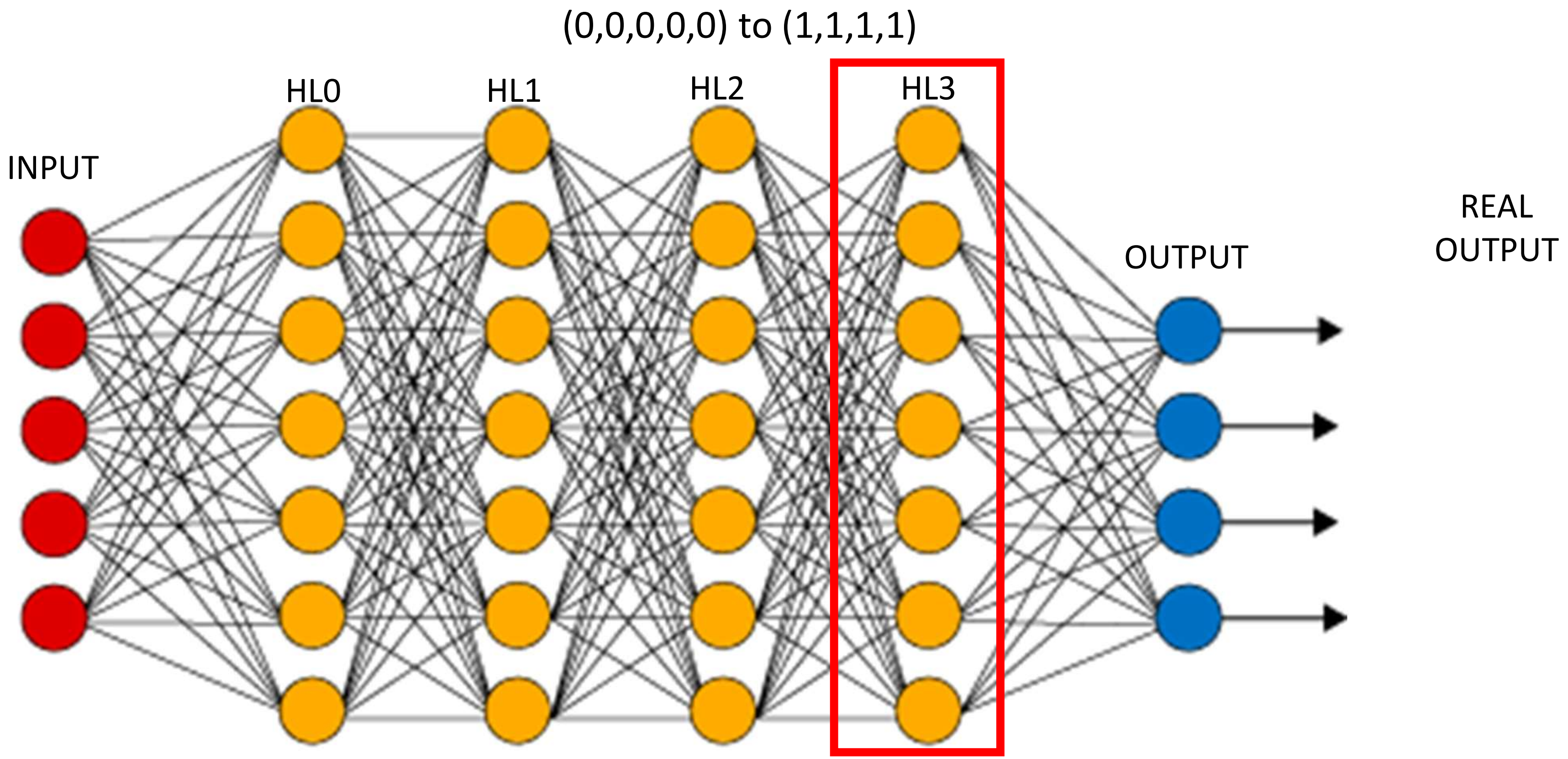
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



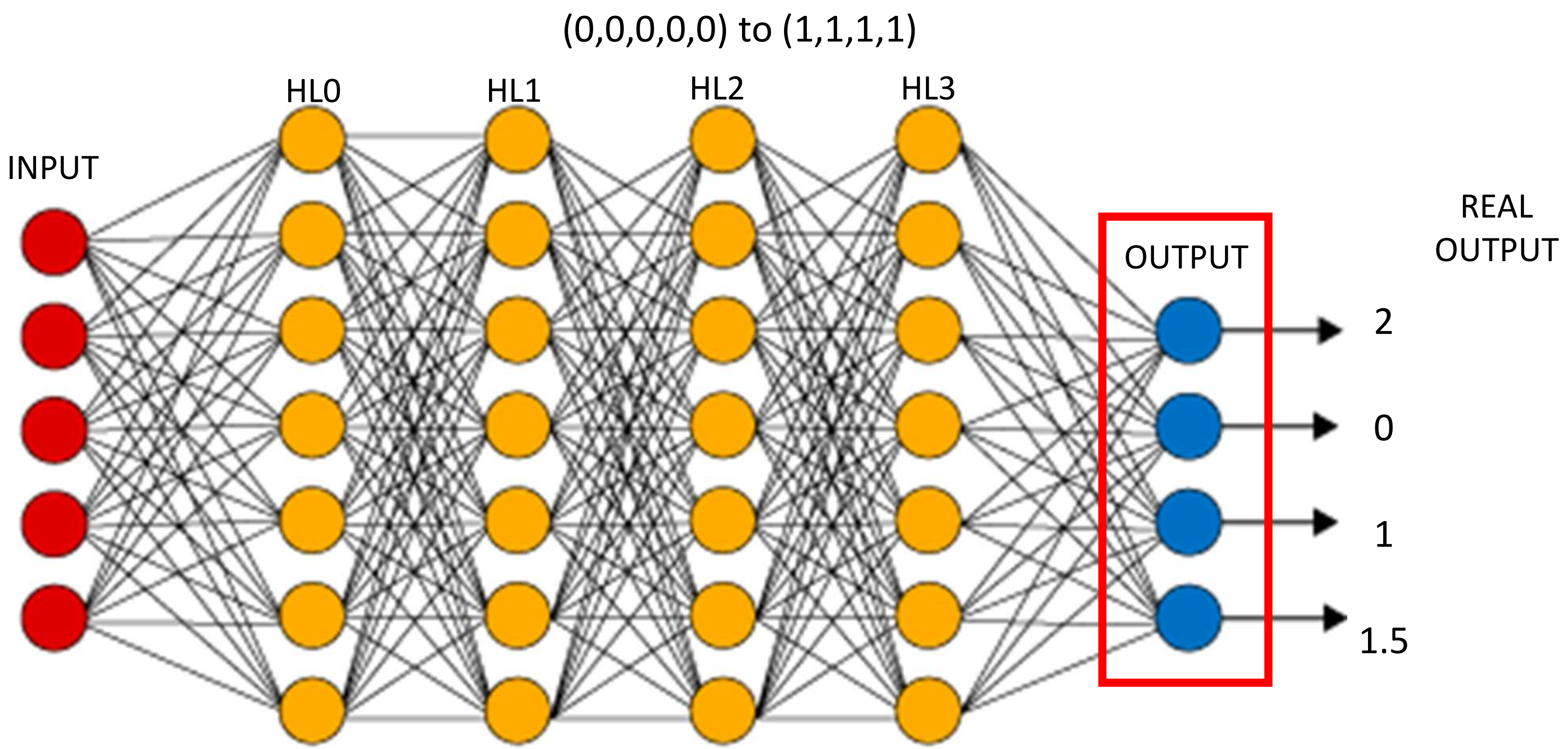
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



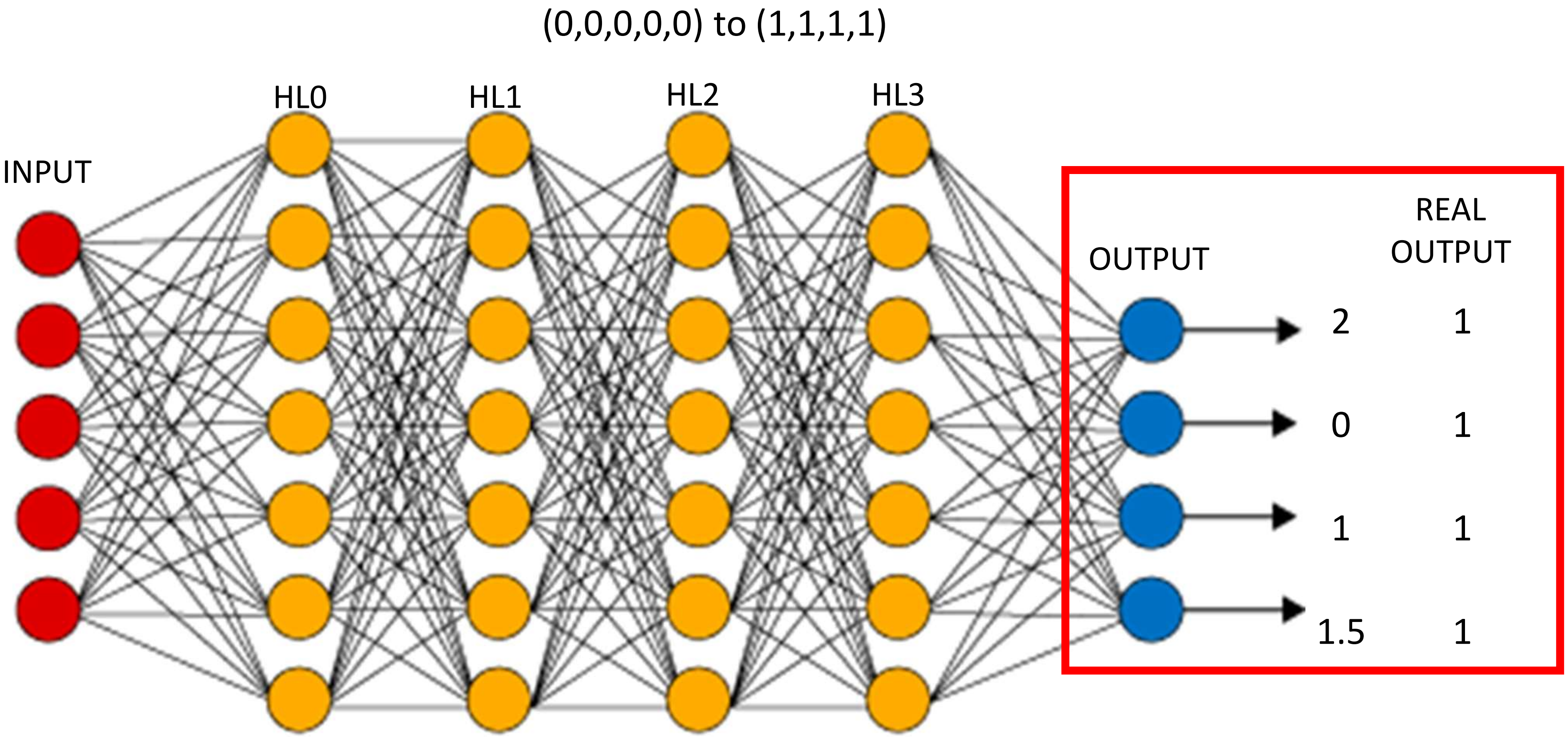
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



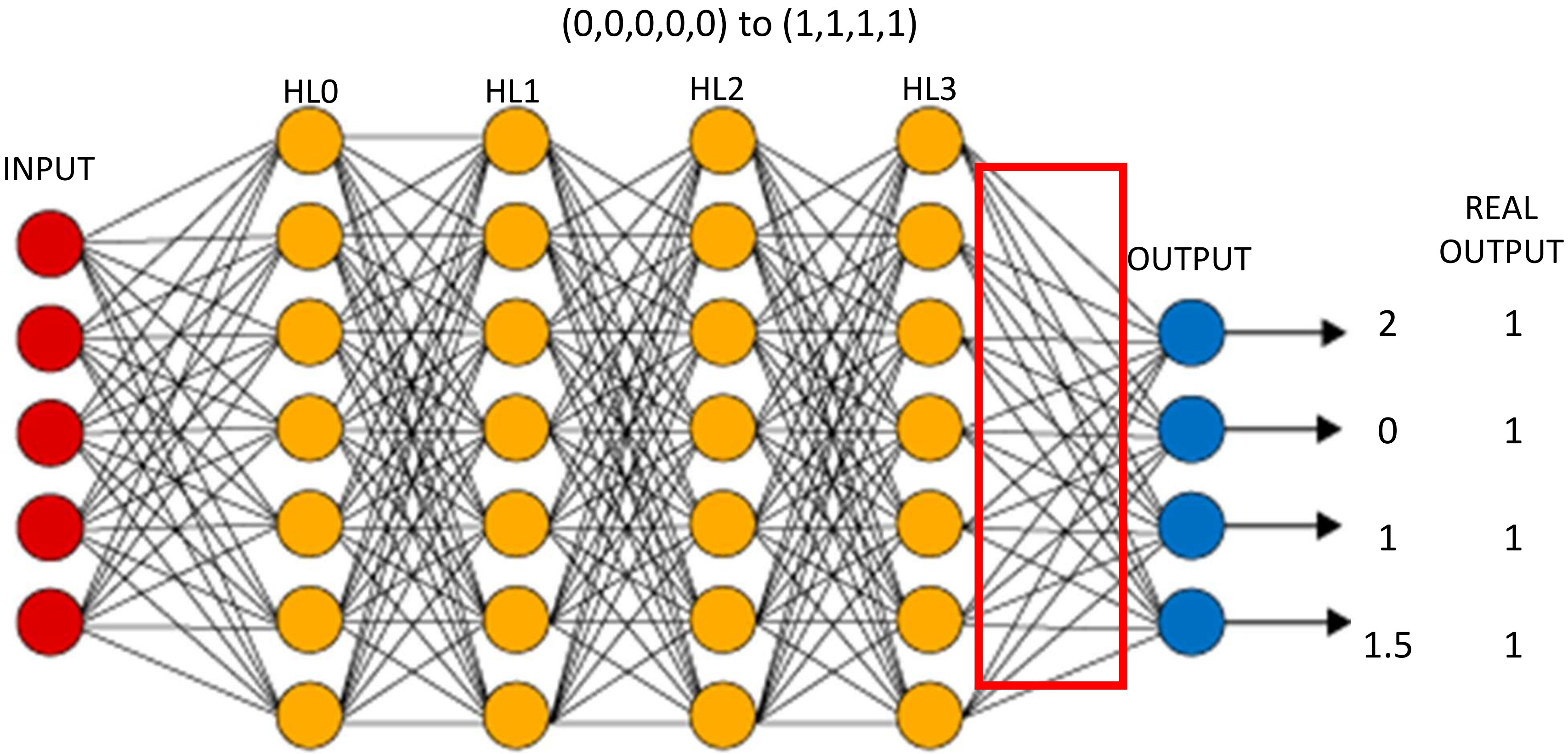
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



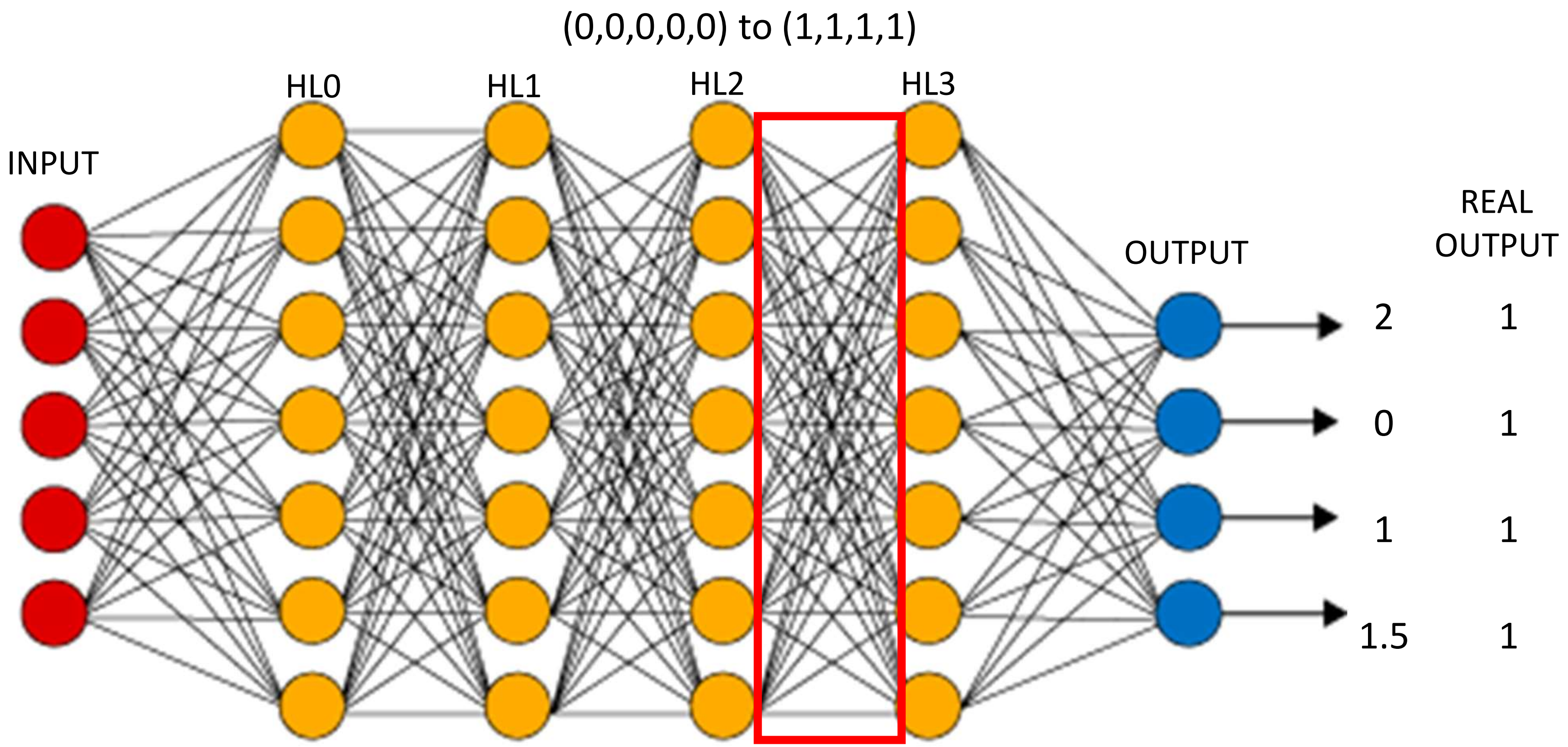
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



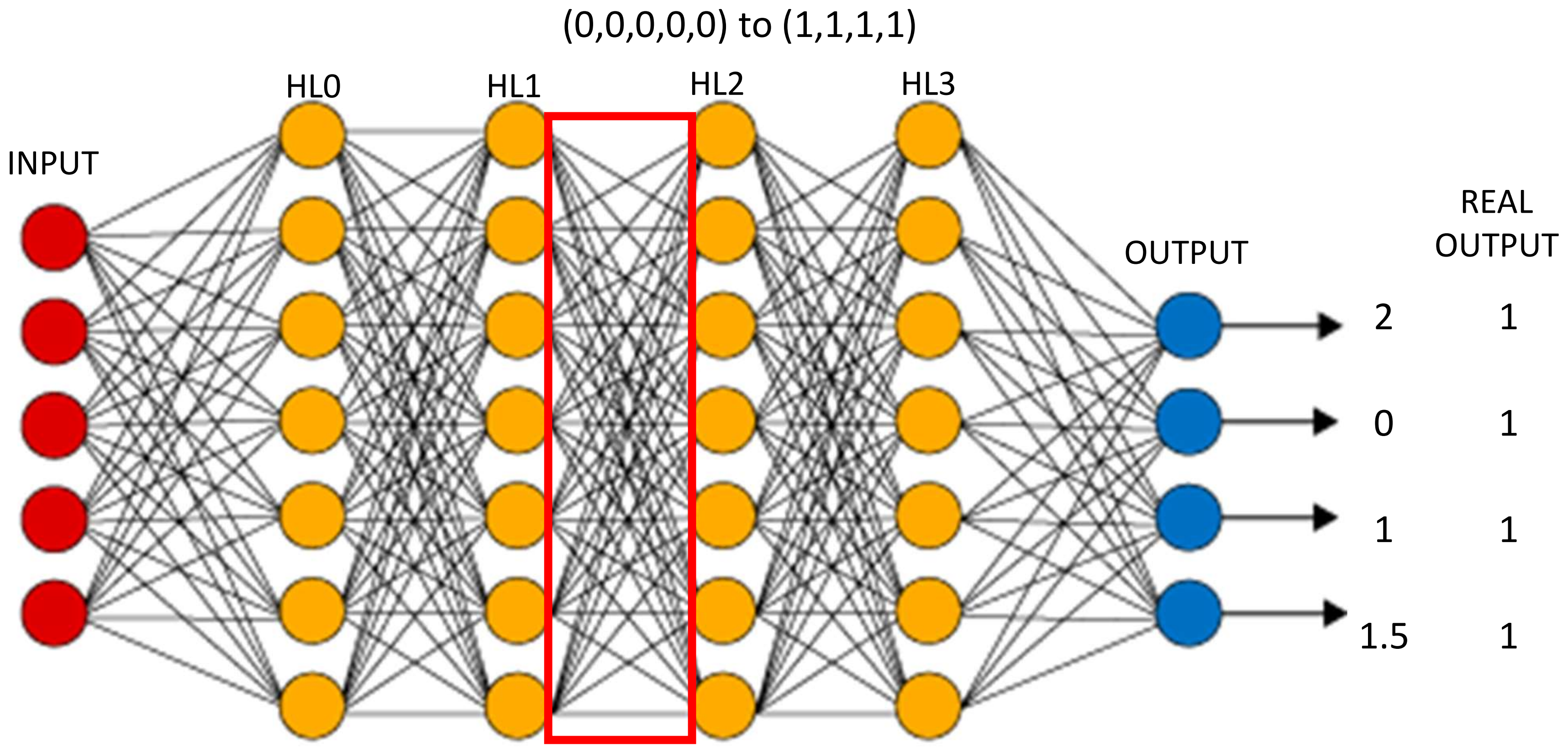
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



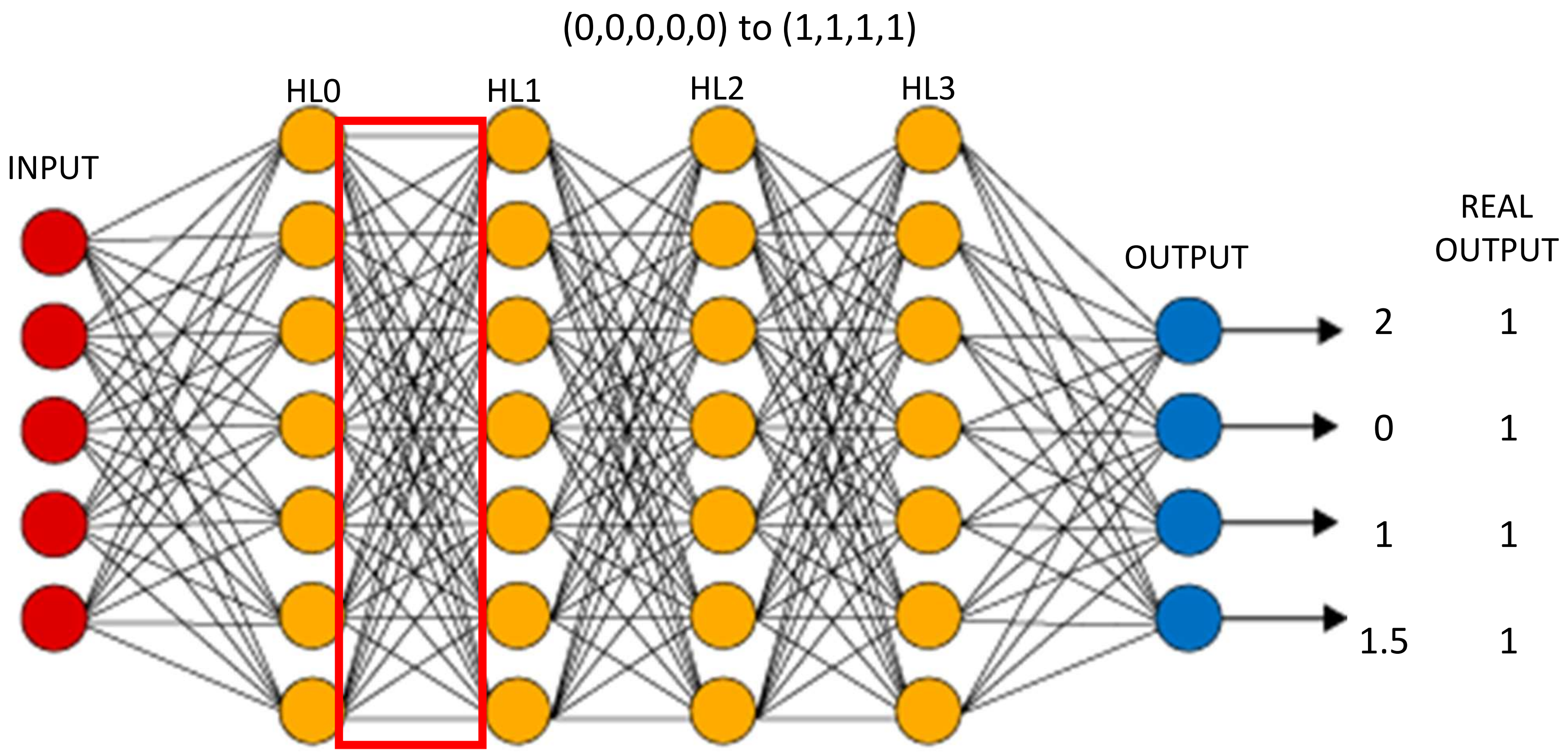
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



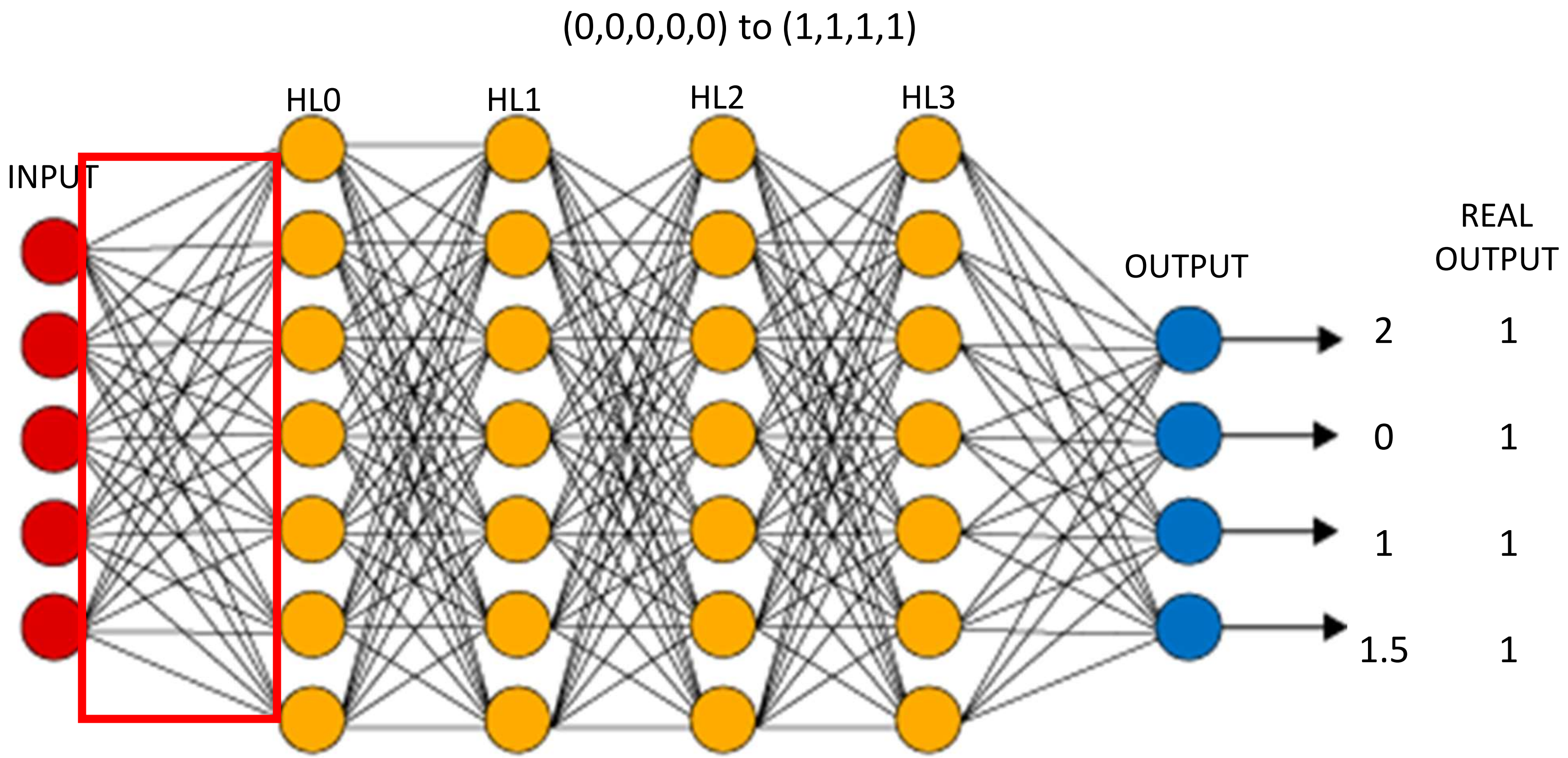
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



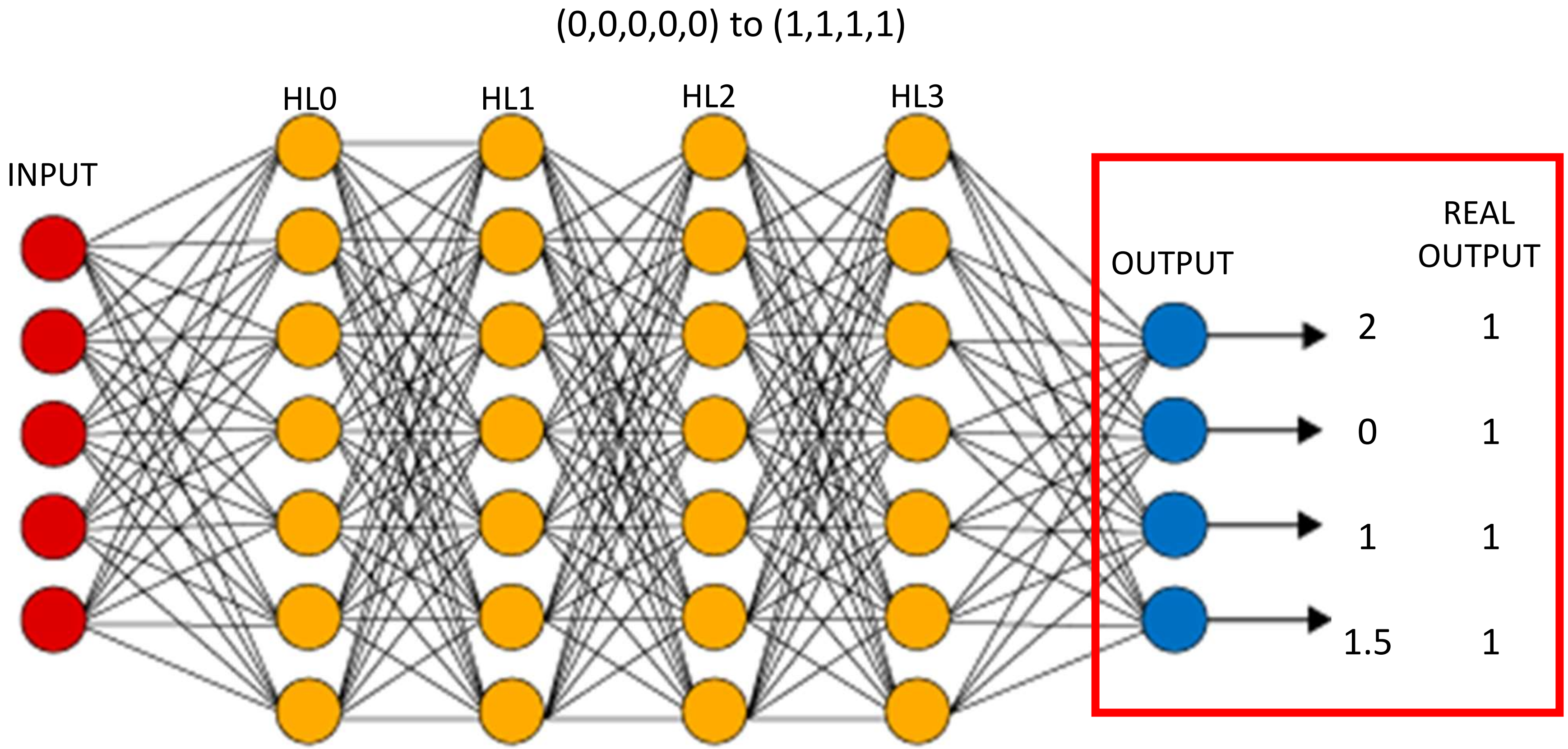
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



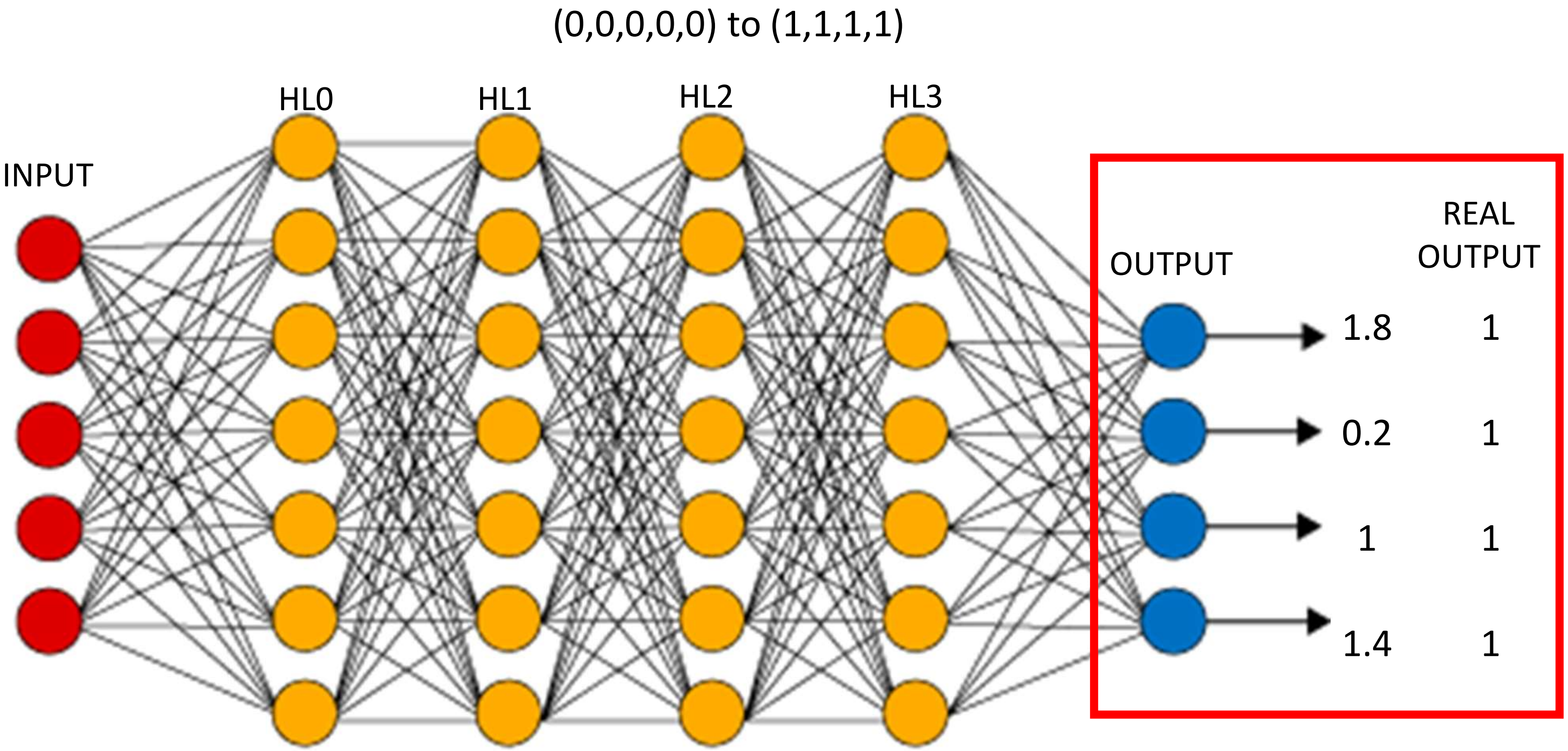
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



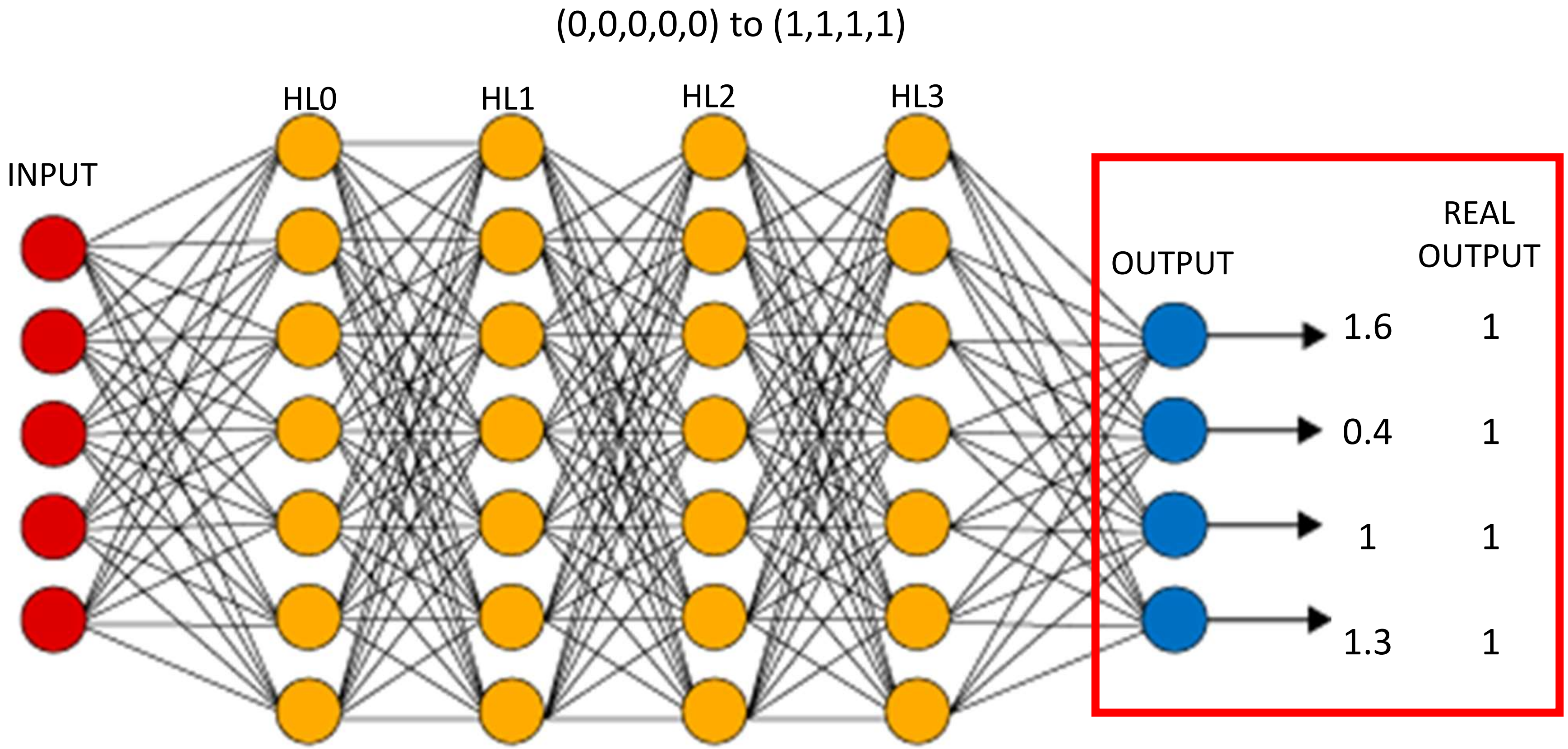
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



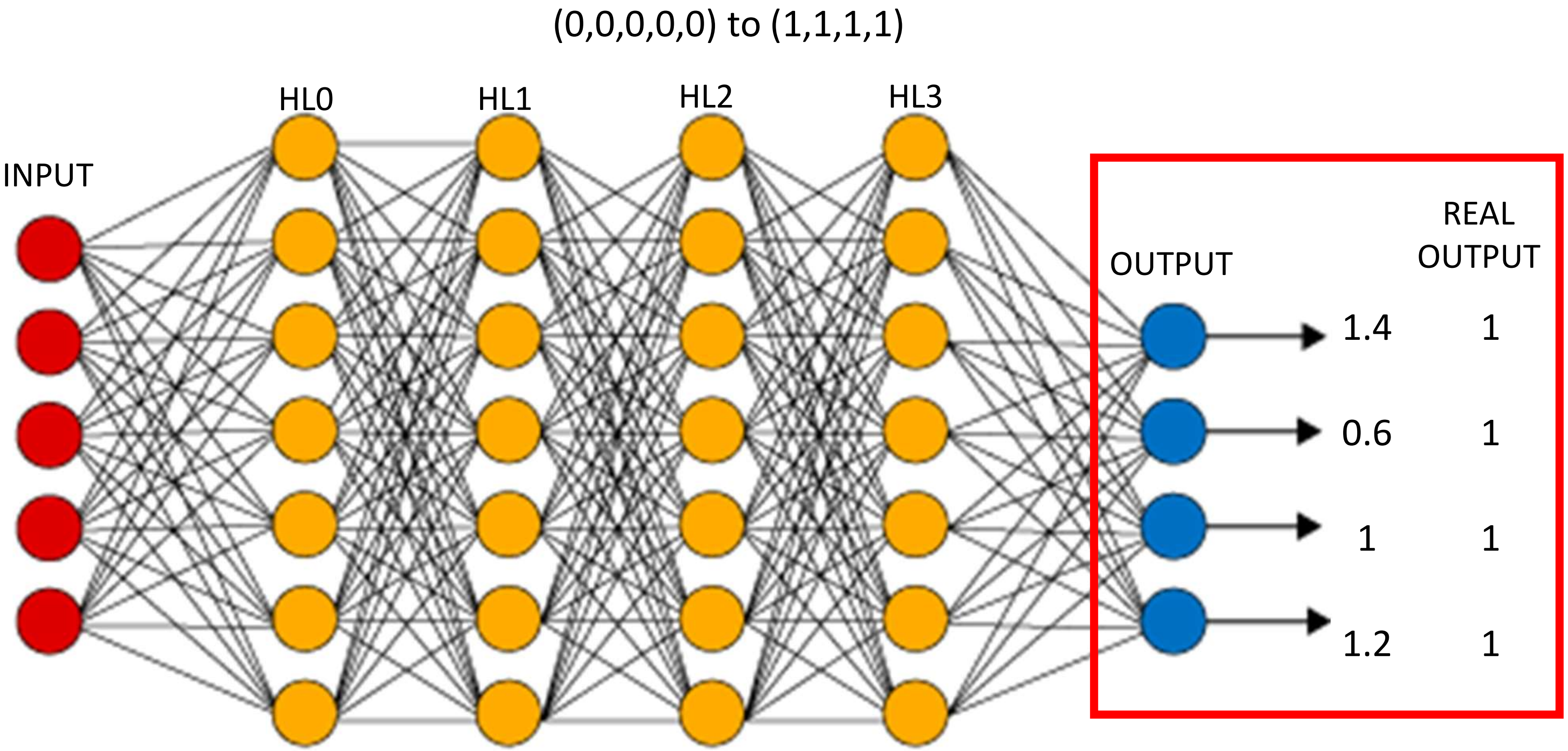
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



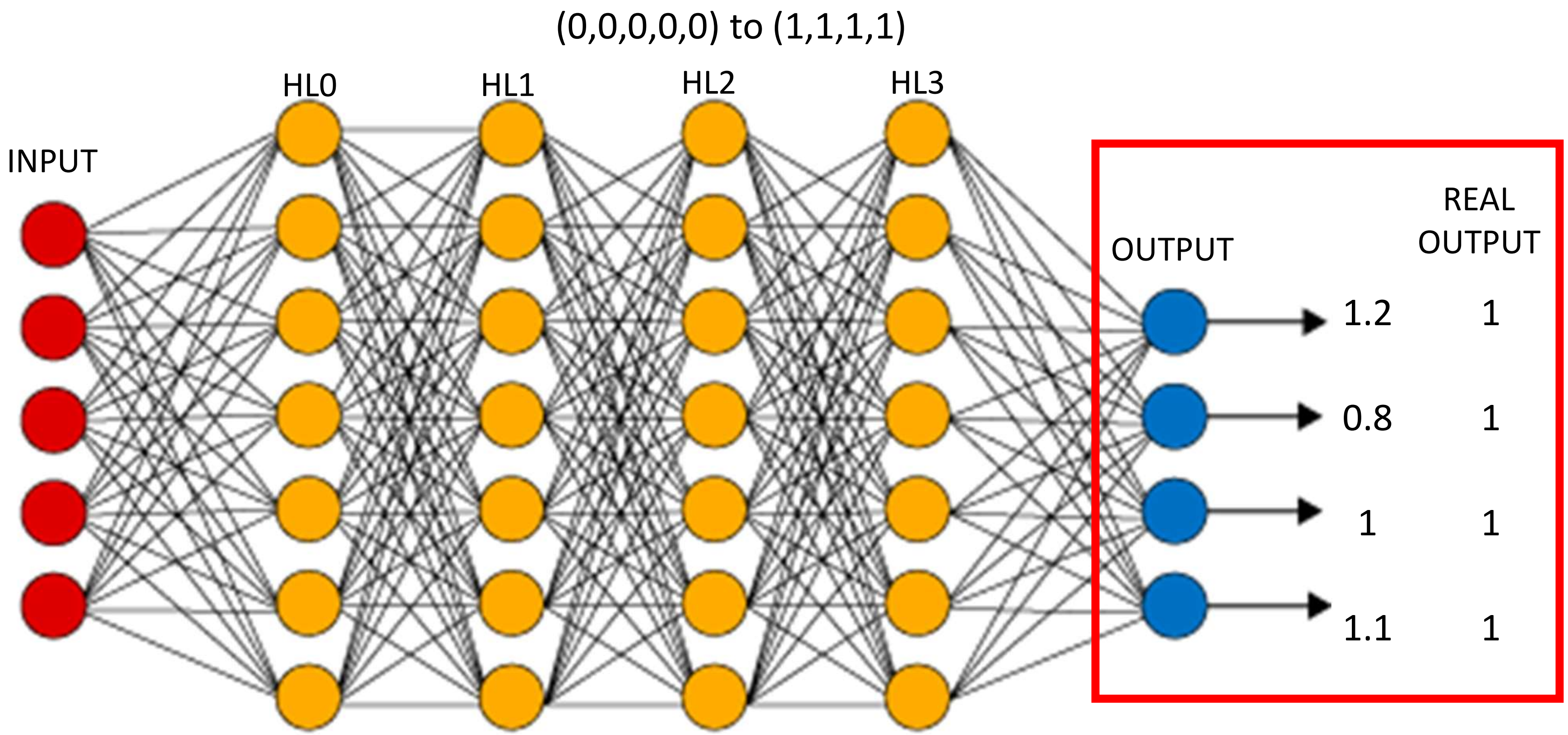
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



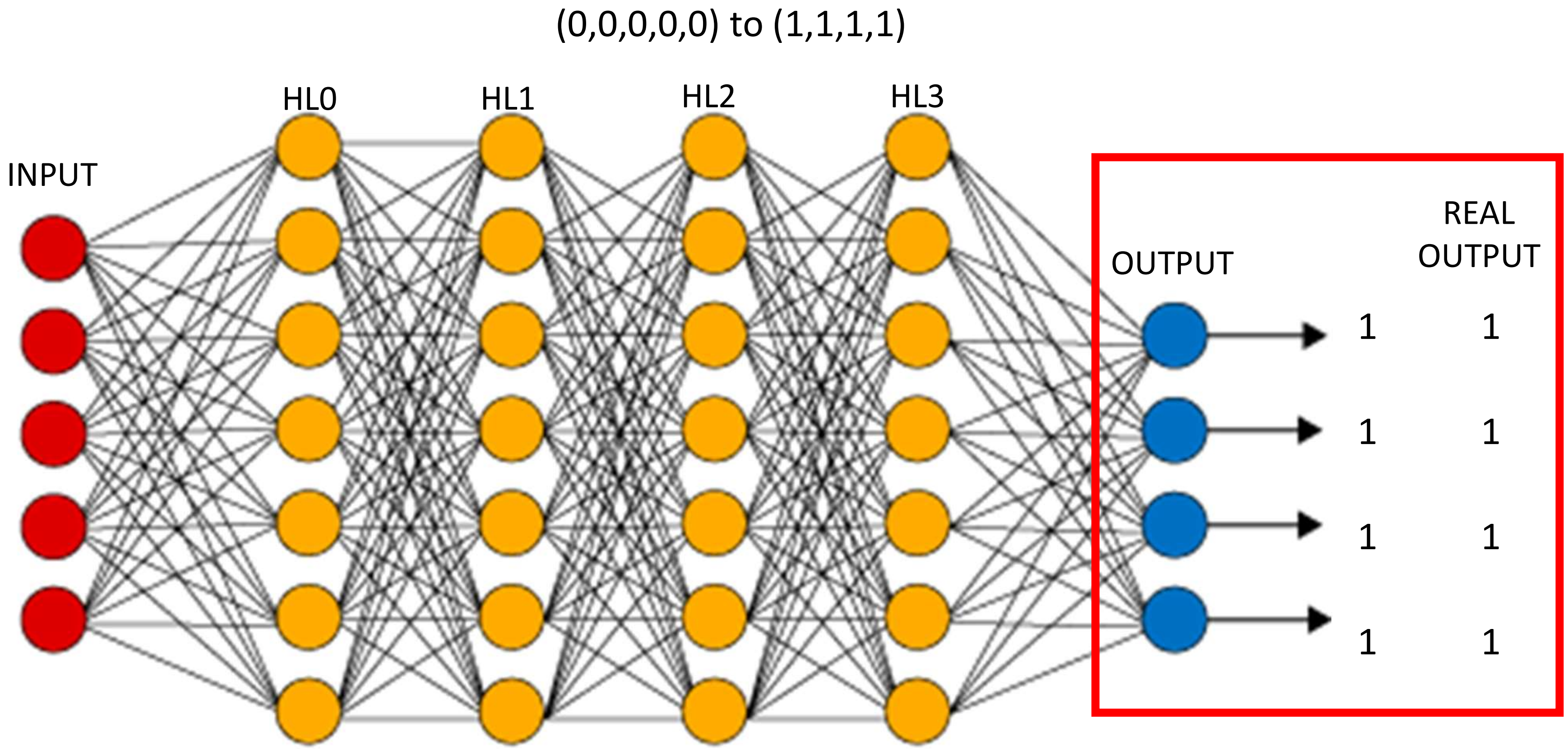
Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



Neural Networks - Theory

- Pipeline: forwards and backwards, row by row



Neural Networks - Theory

- Pipeline: forwards and backwards, **batch by batch**
  - To take advantage of GPUs and matrix calculations

- Given any sort of numerical dataset, we can say it represents a function f(INPUTS) = OUTPUTS
  - Classification - outputs are probabilities of classes
  - Regression - outputs are numerical values
  - Image processing - inputs are pixel values
  - Natural Language processing - inputs are letters
  - ...
- And a neural network can approximate this function

Neural Networks - Architectures

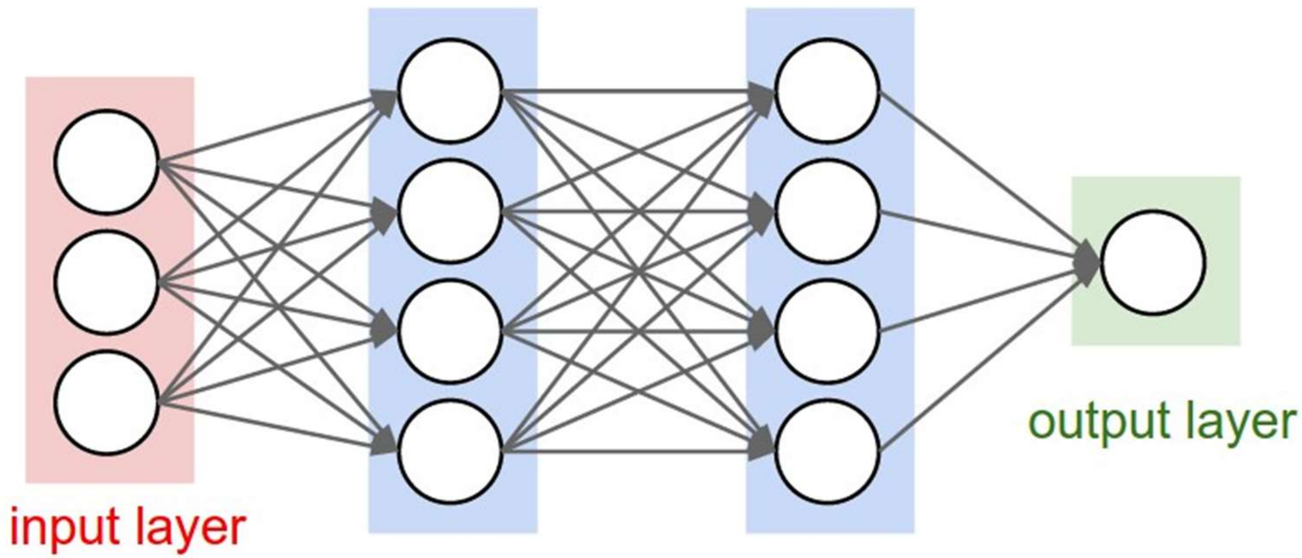
- Many different types of networks
  - No ideal architecture, just recommendations

- Layers
  - Type
  - Depth
  - Width

- Activation functions
- Loss functions
- Optimizer

Neural Networks - Architectures

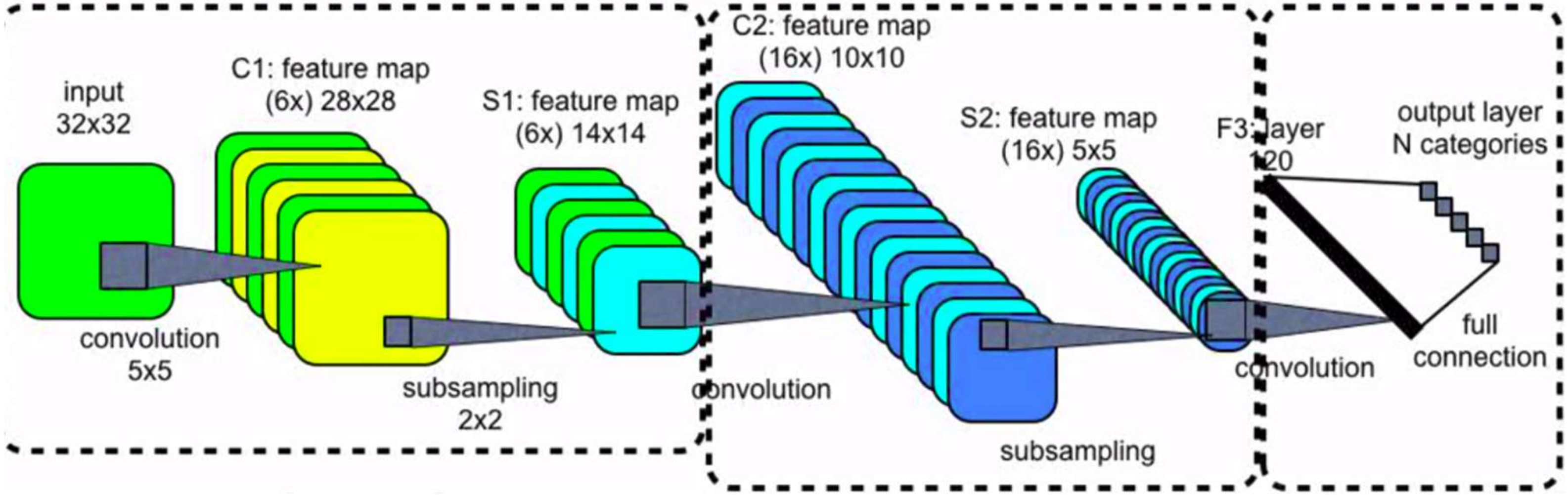
- Fully Connected Layers



- Used everywhere
- Just connect all nodes from previous layer to all nodes from next layer

Neural Networks - Architectures

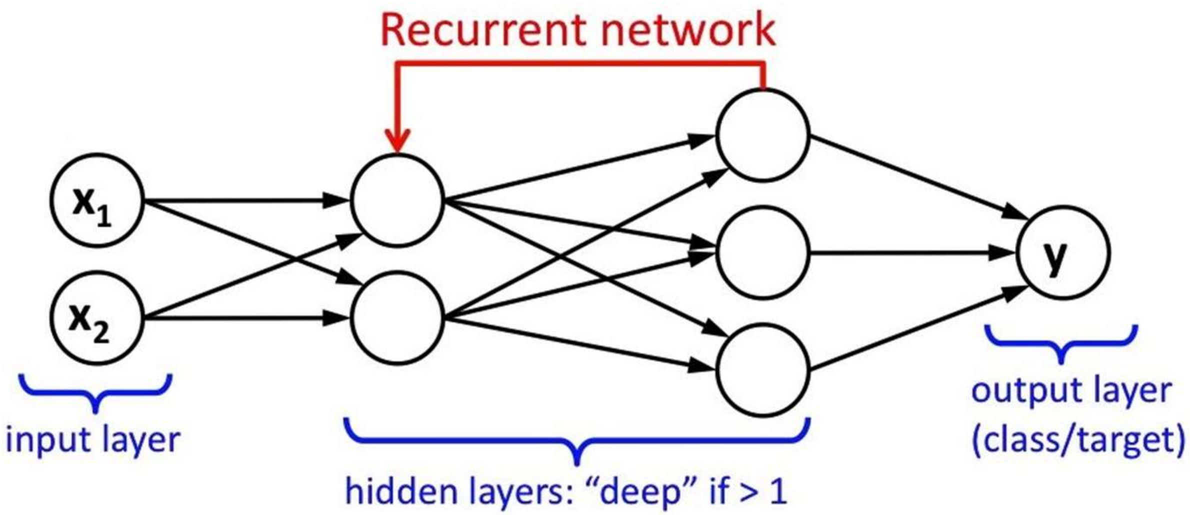
- Convolutional Layers



- Used with images
- Have a set of weights that processes many small parts of the previous layer

Neural Networks - Architectures

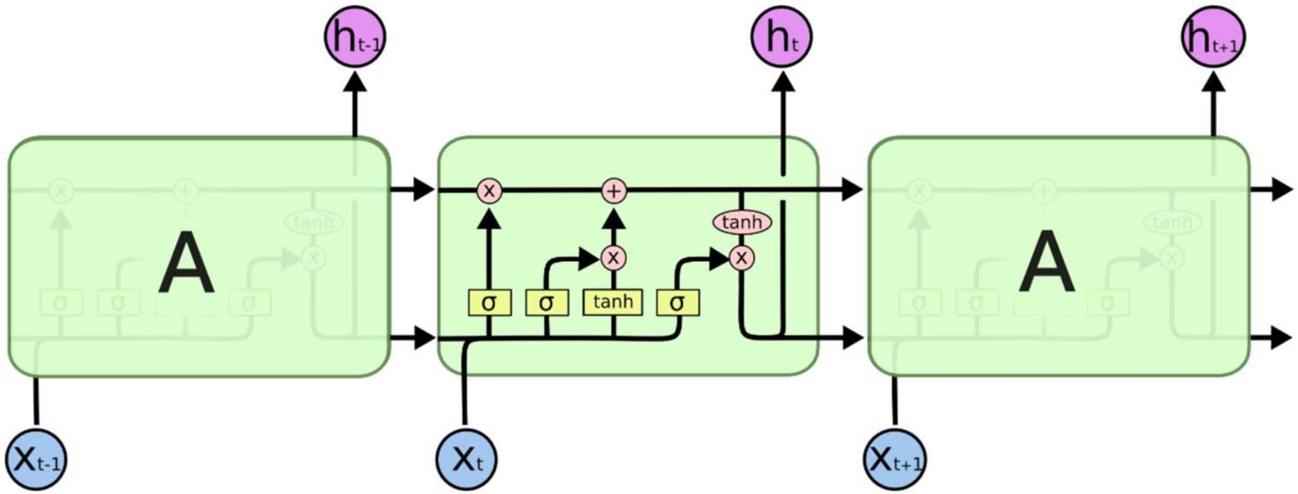
- Recurrent Layers



- Used when input has temporal dependencies
- The layer feeds previous output to other layers

Neural Networks - Architectures

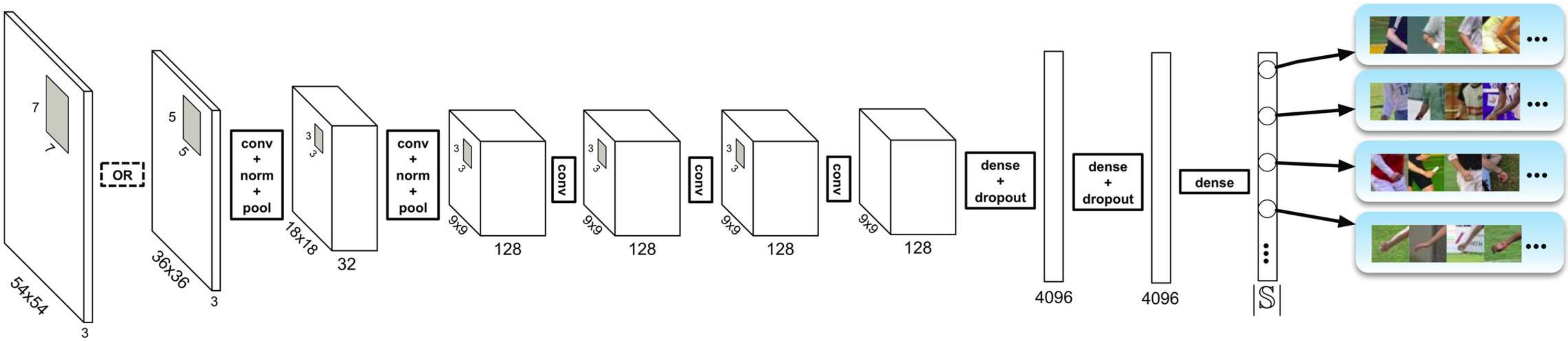
- Long Short-Term Memory (LSTM) Layers



- Used when input has temporal dependencies
- The layer saves information from forward passes to give to future forward passes

Neural Networks - Architectures

- Depth and Width

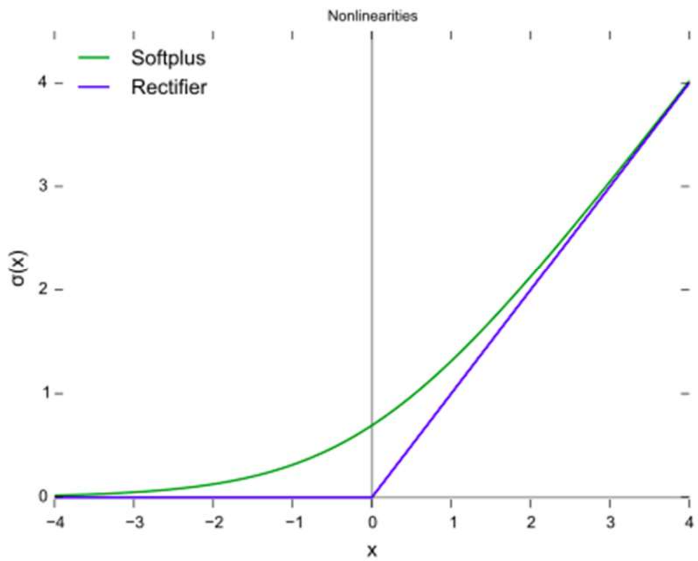


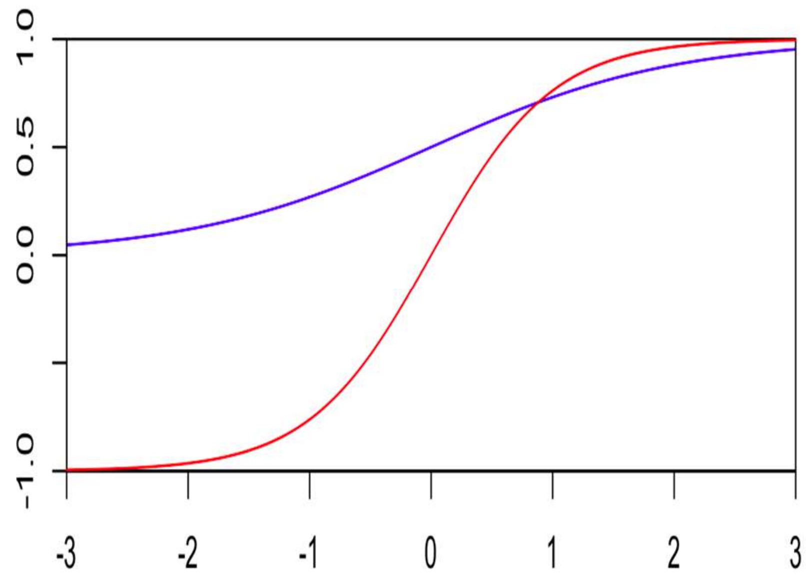
- Depends on the problem. No fixed formulae, but usually larger means capable of more complex functions
- Some proposals have networks finding their own ideal architectures

Neural Networks - Architectures

- Activation Functions
  - Allow for non-linearity
  - Without these, we could only approximate linear equations

- RELU
- ELU
- Softplus
- Sigmoid
- Tanh
- Identity
- …

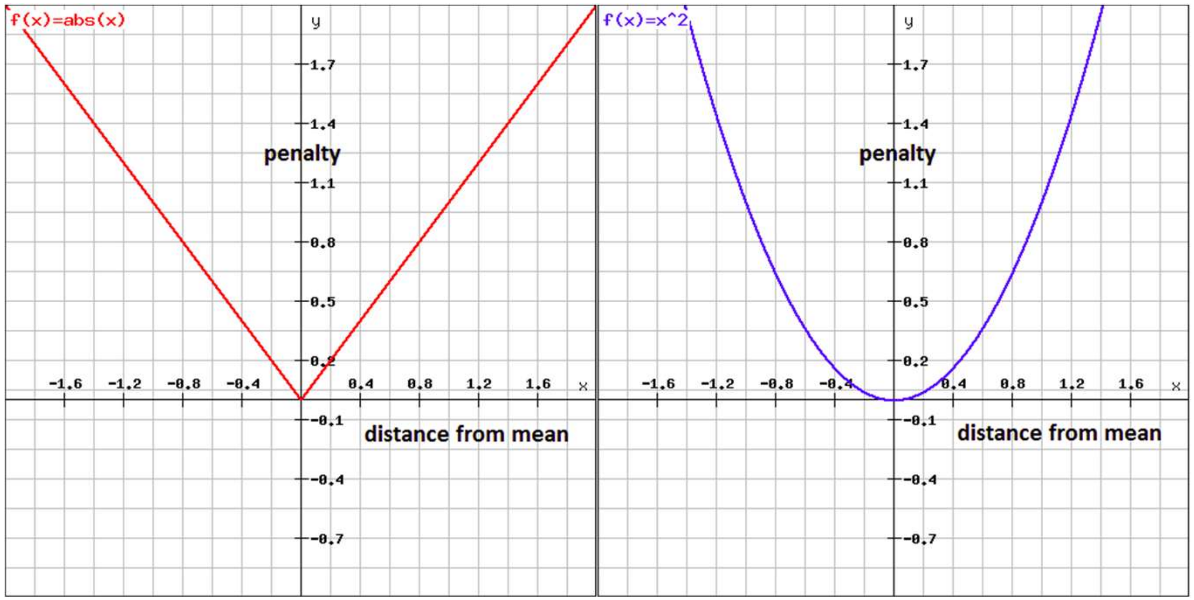




Neural Networks - Architectures

- Loss Functions
  - Affect learning process/speed

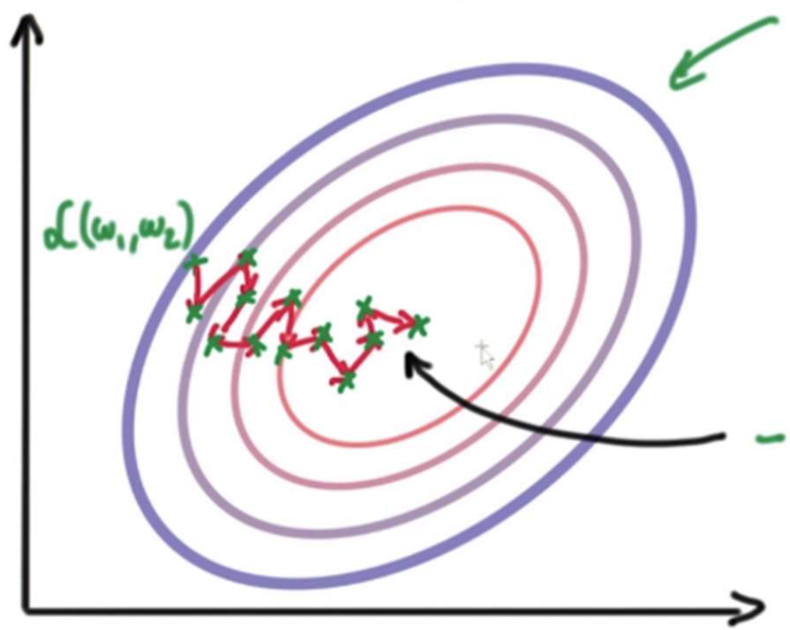
- Absolute difference
- Mean Squared
- Log difference
- Cross-entropy
- ...



Neural Networks - Architectures

- Optimizers
  - Affect learning process/speed
  - Some solve problems with vanishing/exploding gradients on very deep networks

- Stochastic Gradient Descent
- ADADelta
- ADAGrad
- ADAM
- RMSProp
- ...



Neural Networks - Hyper-Parameters

- Input / output size (problem-dependent)
- Learning Rate
- Batch size
- Learning stop condition
  - Time
  - Performance
- Depth / width (layers)
- Kernel configurations (convolutional layers)
- ...

Neural Networks - Other issues

- Vanishing gradients
- Exploding gradients
  - On very deep networks, solved with some optimizers

- Overfitting
  - Drop-outs - turn on and off random nodes to create random noise

- If things don't work, no one really knows why
  - Black-box
  - We have no idea what the weights mean…

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Robotics / Intelligent Robotics

Machine Learning – Neural Networks/Deep learning

Luís Paulo Reis, Nuno Lau, David Simões, Armando Sousa

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.

Associate Professor at DEI/FEUP – Informatics Engineering Department, Faculty of Engineering of the University of Porto, Portugal

President of APPIA – Portuguese Association for Artificial Intelligence