Universidade do Minho
Departamento de Informática

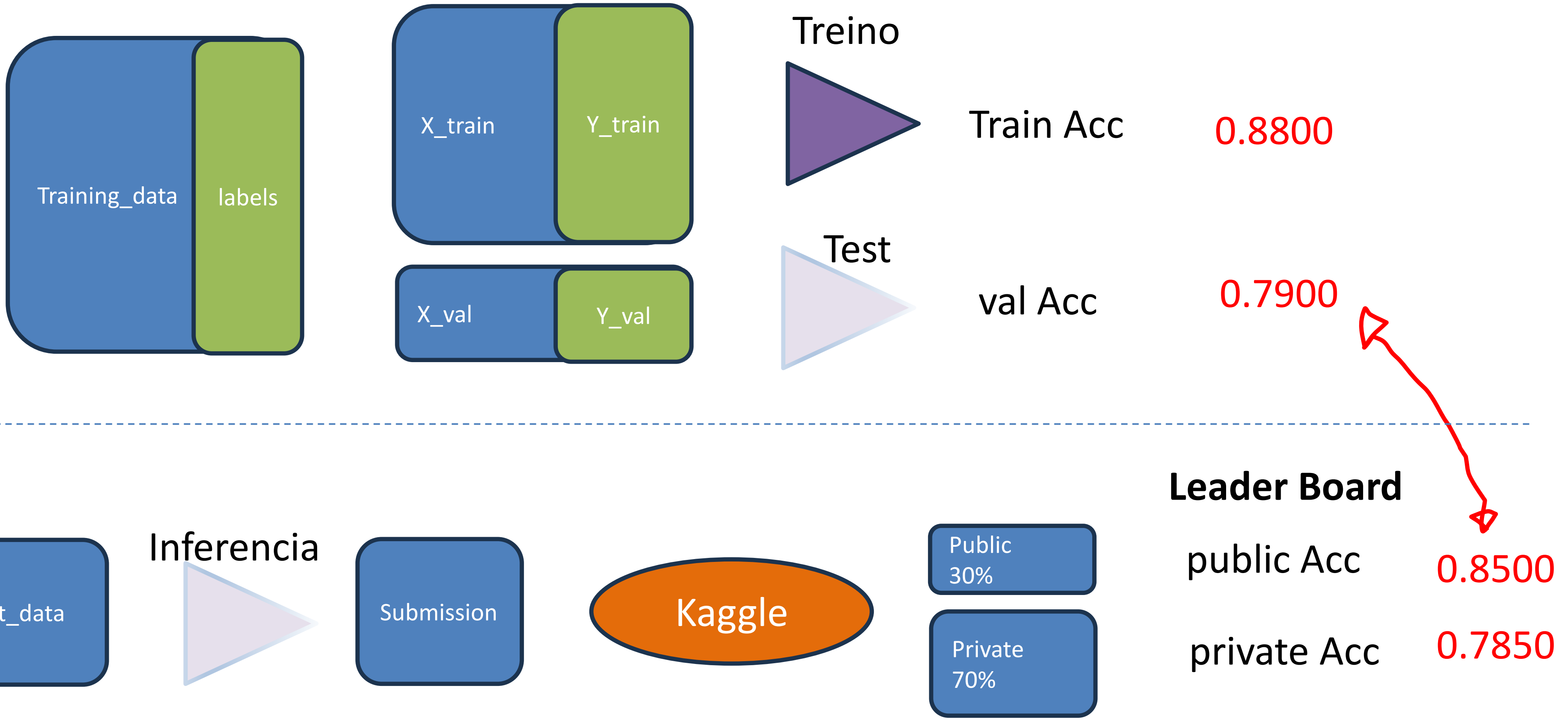
Synthetic Intelligence Lab

DADOS e APRENDIZAGEM AUTOMÁTICA

Artificial Neural Networks

Mestrado em Engenharia Informática
Mestrado em Inteligência Artificial
Mestrado em Engenharia e Ciência de Dados
Mestrado em Engenharia de Sistemas
Mestrado em Matemática e Computação
Mestrado em Cibersegurança

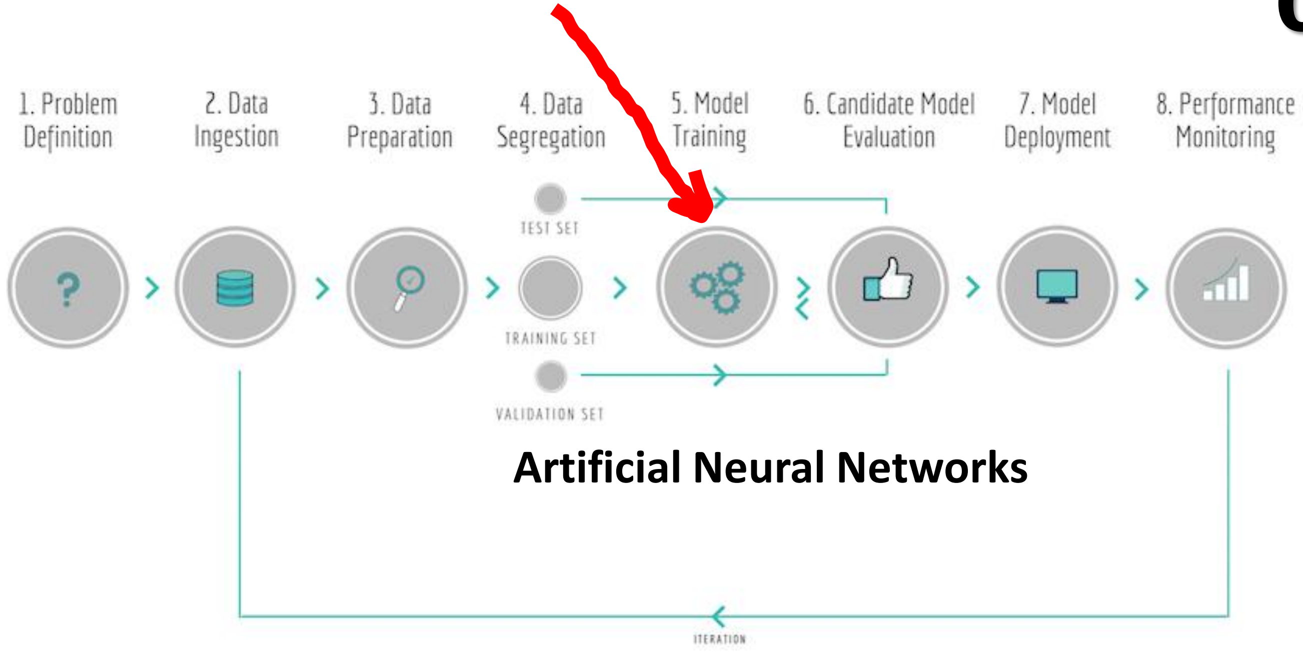
Overfitting



Universidade do Minho
Departamento de Informática

ISLab
Synthetic Intelligence Lab

Contents



Artificial Neural Networks

Analogy: natural learning

- The brain is a **highly complex, non-linear and parallel structure**.
- It has an ability to organize its neurons in order to **perform complex tasks**.
- A neuron is 5/6 times **slower** than a logic gate.
- The brain **overcomes slowness through a parallel structure**.
- The human cortex has **10 billion neurons** and **60 trillion synapses**!

What are artificial neural networks

Neural networks are **machine learning models** that follow an analogy with the functioning of the human brain.

An artificial neural network is a parallel processor, composed of simple processing nodes (neurons).

Knowledge is **stored in the connections** between neurons.

Knowledge is acquired **from an environment** (data), **through a learning process** (training algorithm) that adjusts the parameters of the network.

Benefits/reasons for success

Learning/generalisation - allows acquiring new knowledge of the environment

Massively parallel processing - allows complex tasks to be performed in a short timeframe

Non-linearity - useful for many real problems

Adaptability - can adapt their topology according to changes in the environment

Robustness and smooth degradation - able to **ignore noise** and irrelevant attributes; handle incomplete information efficiently

Flexibility - large applicability domain

Usability - can be used as "black boxes"; no explicit knowledge of the function to be learned is required

Common Types of Applications

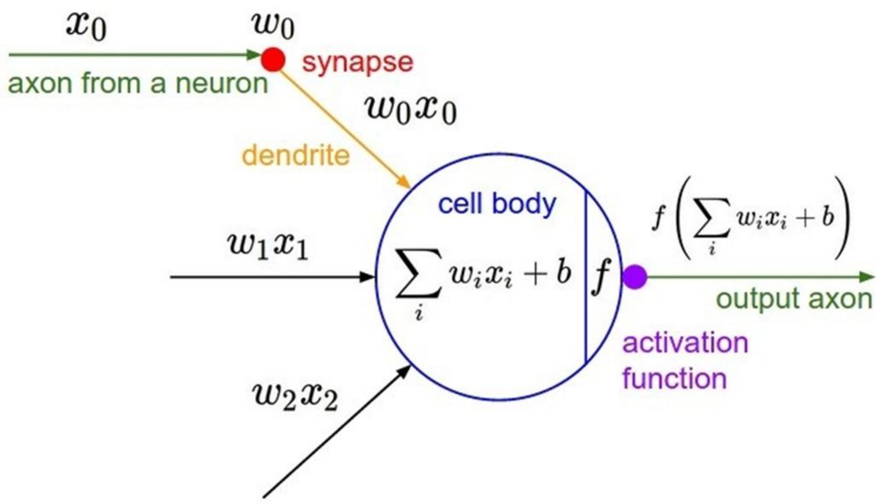
- Associative memory;
- Classification/Diagnosis;
- Pattern recognition;
- Regression;
- Control;
- Optimization;
- Data filtering / compression;
- …

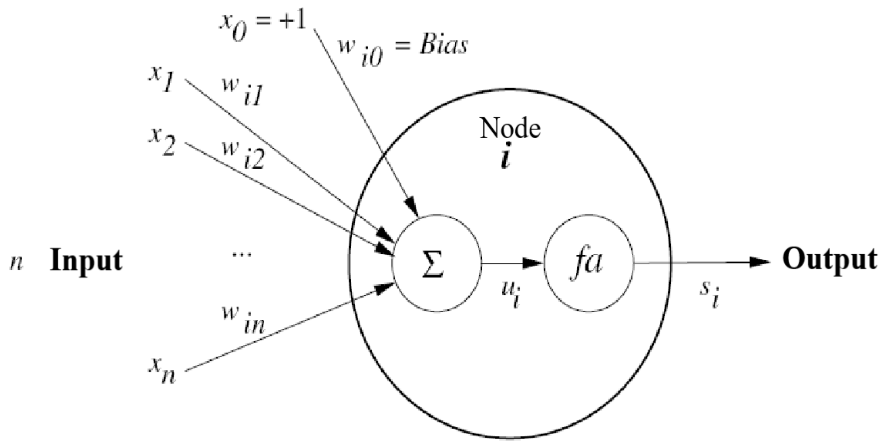
Universidade do Minho
Departamento de Informática

ISLab Synthetic Intelligence Lab

Artificial neurons (nodes)

- They receive a set of **inputs,** data or connections (x_i)
- A **weight** (numerical value) associated with each connection (w_i)
- Each neuron calculates its **activation** based on the input values and the weights of the connections
- The calculated signal is passed to the output after being filtered by an **activation function** ($f(\,)$)



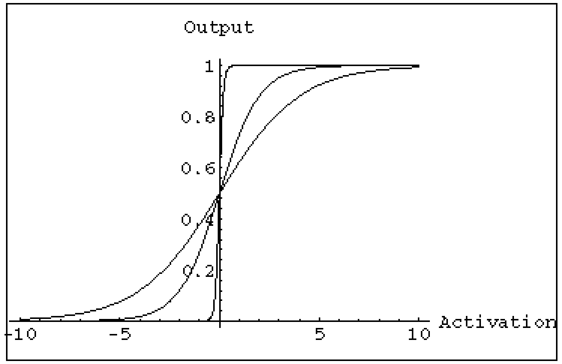


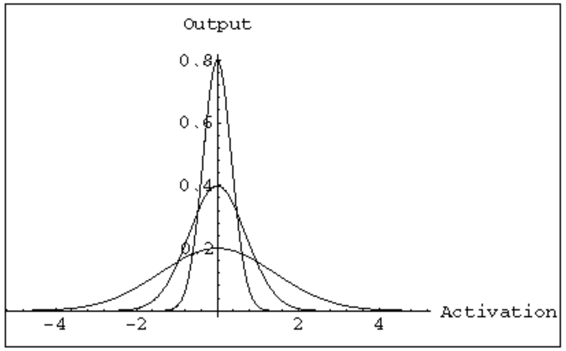
Universidade do Minho
Departamento de Informática

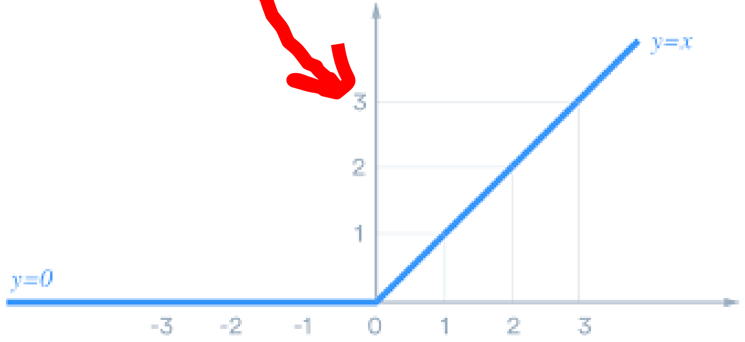
Synthetic Intelligence Lab

Activation functions

- Sigmoid/ logistic
- Linear
- Hyperbolic tangent (Tanh)
- Gaussian
- RelU (linear rectified)



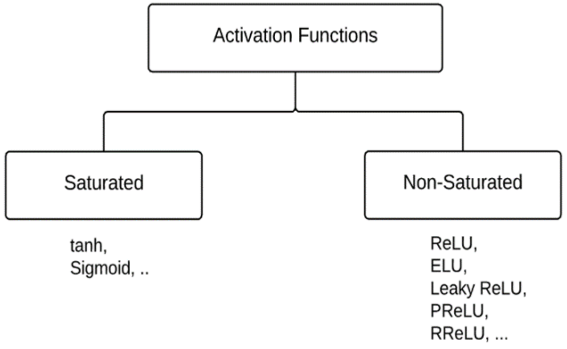


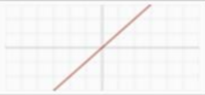
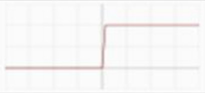
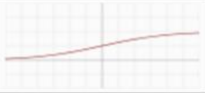
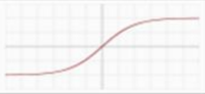
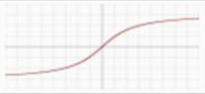
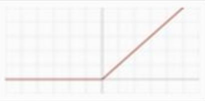
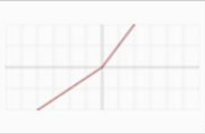
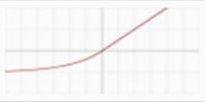
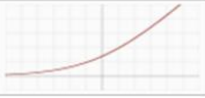


Universidade do Minho
Departamento de Informática

ISLab
Synthetic Intelligence Lab

Activation functions



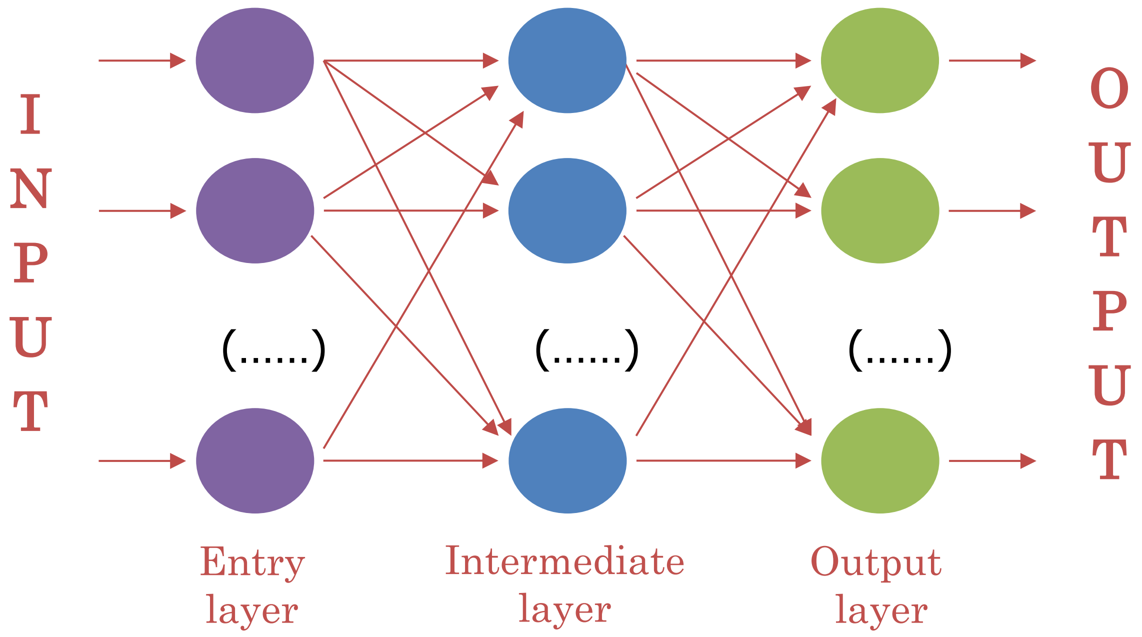
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Network Architectures (topologies)

Architecture (or topology) - the way nodes are interconnected in a network structure

There are numerous types of architectures each with its own potential, falling into two categories: **supervised** and **unsupervised**

Feedforward topology

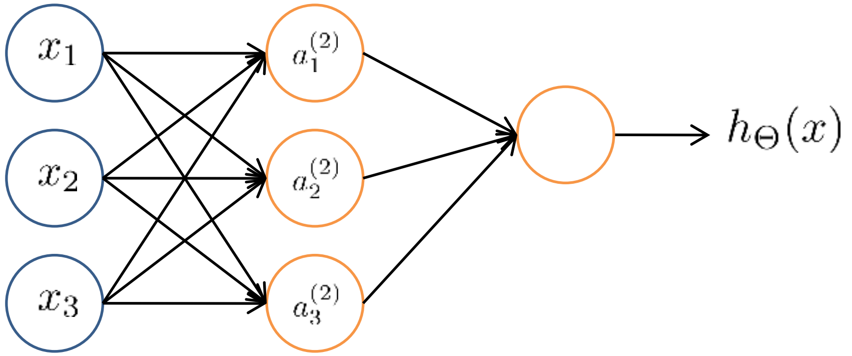


MultiLayer Perceptron (MLP) - Feedforward fully connected neural network with **several intermediate layers.**

With **one intermediate layer** - "*vanilla" neural network*

Universidade do Minho
Departamento de Informática
ISLab Synthetic Intelligence Lab

Network representation



$a_i^{(j)}$ = "activation" of neuron *i* in layer *j*

$\Theta^{(j)}$ = matrix of weights between neurons of layers *j* and *j+1* (rows – *destination*, columns - *source*)

sigmoid =1

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

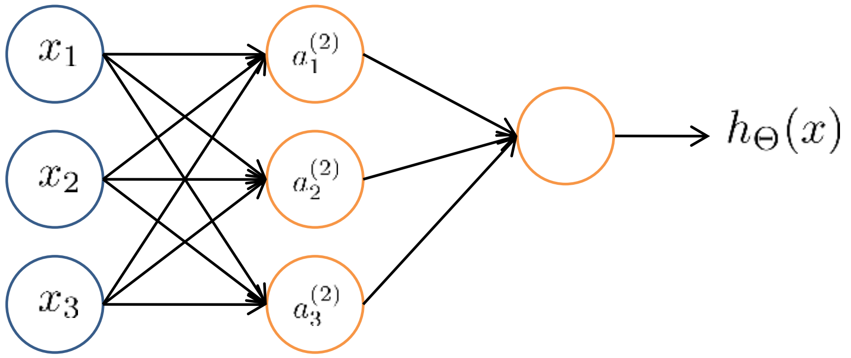
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

=1

Universidade do Minho
Departamento de Informática

ISLab
Synthetic Intelligence Lab

Computation of the output: vectorised



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x$$

$$a^{(2)} = g(z^{(2)})$$

Activation values of the intermediate layer

$$\text{Add } a_0^{(2)} = 1$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Network output values

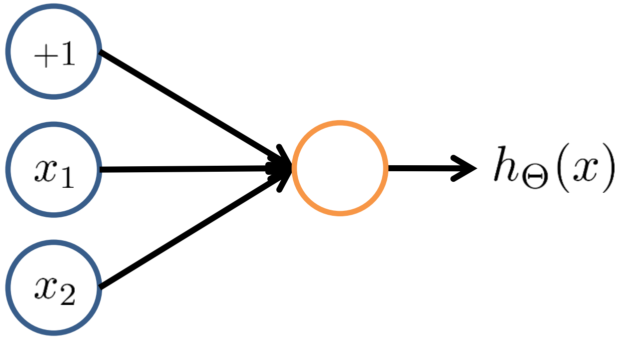
```
def __init__(self, dataset, hidden_nodes = 2, normalize = False):
        self.X, self.y = dataset.getXy()
        self.X = np.hstack ( (np.ones([self.X.shape[0],1]), self.X ) )
        self.h = hidden_nodes
        self.W1 = np.zeros([hidden_nodes, self.X.shape[1]])
        self.W2 = np.zeros([1, hidden_nodes+1])
```

```
def predict( self, instance):
        x = np.empty([self.X.shape[1]])
        x[0] = 1
        x[1:] = np.array(instance[:self.X.shape[1]-1])
        z2 = np.dot(self.W1, x)
        a2 = np.empty([z2.shape[0]+1])
        a2[0] = 1
        a2[1:] = sigmoid(z2)
        z3 = np.dot(self.W2, a2)
        return sigmoid ( z3 )
```

Universidade do Minho
Departamento de Informática

ISLab
Synthetic Intelligence Lab

Computation of the output: Examples

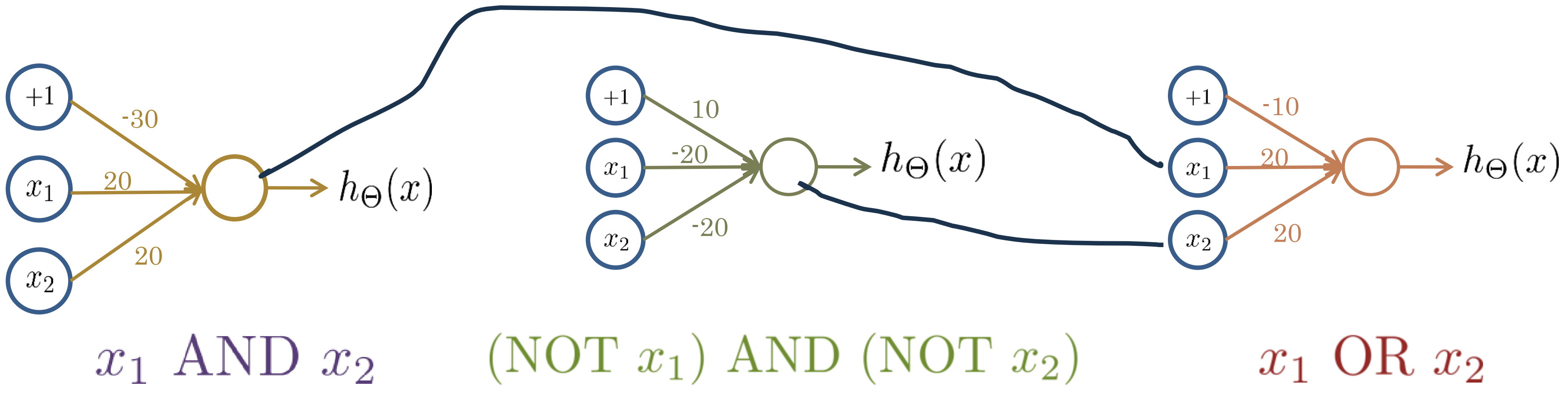


x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

Calculate the output value for the 2 weight cases:

- 30, 20, 20
-10, 20, 20
10, -20, -20

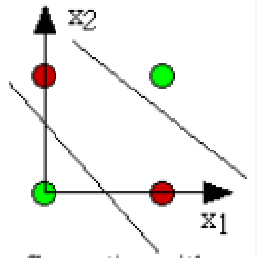
Computation of the output: Examples



-30 + 0x20 + 0x 20 = -30 ->0
-30 + 0x20 + 1x 20 = -10 ->0
-30 + 1x20 + 0x 20 = -10 ->0
-30 + 1x20 + 1x 20 = 10 ->1

How to make h(x1, x2) = x1 XNOR x2 with a network with an intermediate layer ?

Note that: x1 XNOR x2 = (x1 AND x2) OR (NOT x1 AND NOT x2)



Outputs: classification problems

If we use functional models for **classification problems**, we will have to convert the output of the model (**numeric value**) into a value of the desired output attribute (**nominal**), i.e. into the expected class.

We can opt for either of the two previously mentioned hypotheses: **one neuron dividing the domain** or **1-of-C / one-hot encoding**.

In the latter case, we will have M numeric outputs (1 per class), and the class corresponding to the highest value is usually chosen.

In this case we can also easily calculate probabilities for each class (***softmax*** function).

Training

Data: training examples consisting of inputs and their desired outputs

Objective: to fix the values of the weights of connections that minimise the lost function: in the case of ANNs, generalisation of the logistic regression cost function

Various **training algorithms** based on the descending gradient

- The most widely used is ***backpropagation***
- Other algorithms: ***Marquardt-Levenberg, Rprop, Quickprop***, etc.

Backpropagation algorithm

- It is based on the gradient vector of the error surface which defines the direction of maximum descent - *similar method to the gradient descent*
- Important parameter: **learning rate**, which defines the distance one walks
- A sequence of these moves leads to a (hopefully global) *minimum*
- Training runs for a given number of ***epochs***: defines the number of times each case is trained by the net, and examples are typically divided into ***batches*** (subsets of examples)
- Initial net configuration (link weights) is normally **randomly generated**
- **Stopping criteria:** fixed number of epochs, time, or convergence criteria based on a subset of validation examples

Phases of the backpropagation algorithm

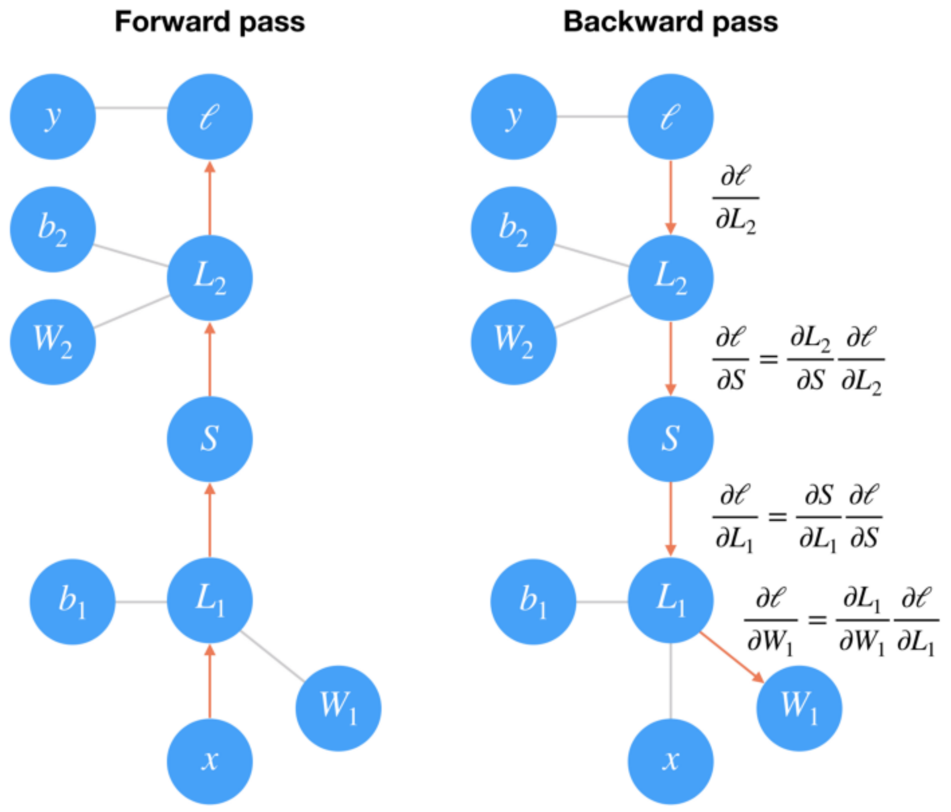
Forward propagation - calculated the output value for the input vector and the error committed

Backpropagation - given the error committed this is propagated backwards, adjusting the weights of the connections in the direction of its decrease. It's based on the calculation of the gradient using the chain rule for composed functions

Universidade do Minho
Departamento de Informática

ISLab
Synthetic Intelligence Lab

Phases of the backpropagation algorithm



Forward pass
Backward pass
y
ℓ
b_2
L_2
W_2
S
b_1
L_1
W_1
x
$\frac{\partial \ell}{\partial L_2}$
$\frac{\partial \ell}{\partial S} = \frac{\partial L_2}{\partial S}\frac{\partial \ell}{\partial L_2}$
$\frac{\partial \ell}{\partial L_1} = \frac{\partial S}{\partial L_1}\frac{\partial \ell}{\partial S}$
$\frac{\partial \ell}{\partial W_1} = \frac{\partial L_1}{\partial W_1}\frac{\partial \ell}{\partial L_1}$

Forwards pass – De baixo para cima na figura:

passo 1: o input x passa pela transformação linear $L1$ com peso $W1$ e bias $b1$
passo 2: o output passa depois pela função ativação sigmoid S e ainda outra transformação linear $L2$
passo 3: finalmente é calculado o erro através da função de loss ℓ, que será utilizado para medir a previsão. O objetivo posterior será minimizar este valor, ajustando os pesos e os bias.

Backward pass – de cima para baixo na figura:

- Para ajustar os pesos usando gradient descent o gradiente do erro é propagado para trás (fim para inicio) através da rede.
- Cada operação apresenta um gradiente entre os inputs e os outputs.
- Ao propagar os gradientes para trás, o gradiente anterior é multiplicado pelo gradiente da operação. Matematicamente, trata-se de calcular o gradiente do erro relativamente aos pesos usando a regra da cadeia (chain rule):

$$\frac{\partial \ell}{\partial W_1} = \frac{\partial L_1}{\partial W_1}\frac{\partial S}{\partial L_1}\frac{\partial L_2}{\partial S}\frac{\partial \ell}{\partial L_2}$$

- Os pesos são ajustados utilizando o gradient com uma taxa de aprendizagem (learning rate) α.

$$\boxed{W_1' = W_1 - \alpha \frac{\partial \ell}{\partial W_1}}$$

-A taxa de aprendizagem é ajustada de modo a que os ajustes dos pesos sejam suficientemente pequenos para estabilizar num mínimo.

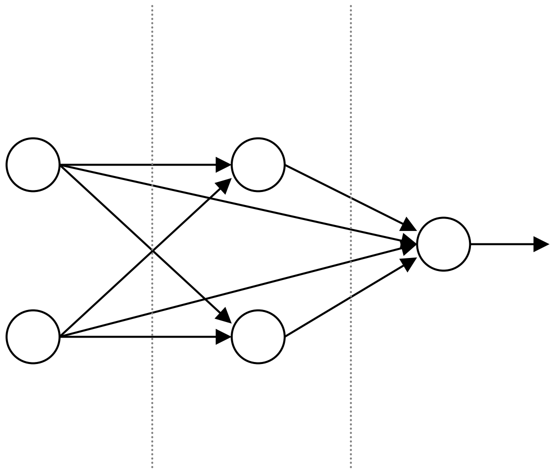
Exemplo prático:
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Summary

- Although there are numerous **variants of neural networks**, each can be defined in terms of:
  - An **activation function**, which transforms a node's net input signal into a single output signal to be broadcasted further in the network;
  - A **network architecture** (or topology), which describes the **number of nodes** in the model as well as the **number of layers** and manner in which they are connected;
  - The **training algorithm** that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal.

Choosing the topology for a feedforward ANN/hyperparameters

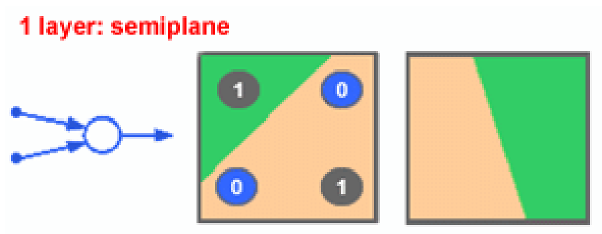
- How many **input** and **output** nodes?
- How many **layers** and **intermediate** nodes?
- How to connect the neurons?
- Shortcut connections?
- **Simplest model**: Feedforward Networks *with fully interconnected layers (MLP)*

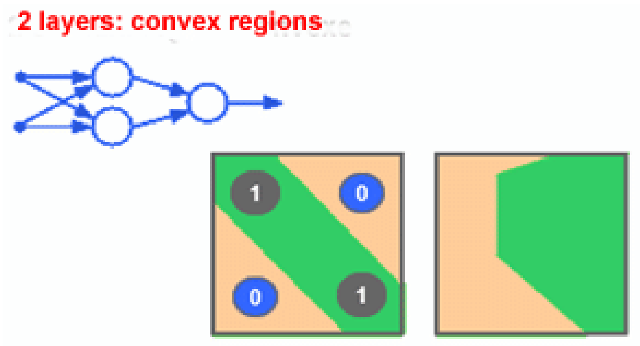


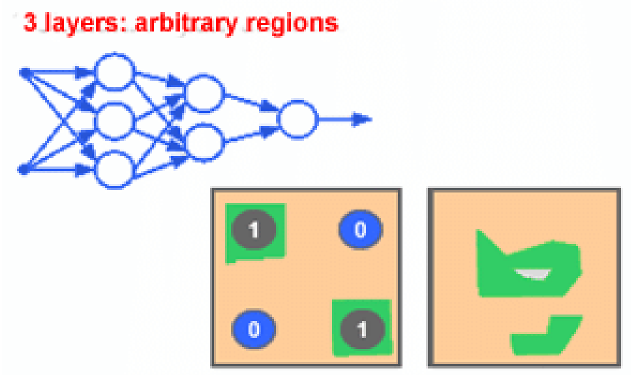
Network Architecture (Topology)

- The **capacity of a neural network to learn** is rooted in its architecture, or the patterns and structures of interconnected neurons;
- **Determines the complexity of tasks that can be learned** by the network;
  - generally, larger and more complex networks are capable of identifying more subtle patterns and complex decision boundaries
  - however, the power of a network is not only a function of the network size, but also the way units are arranged
    - The **number of layers**
    - The **direction of information flow**
    - The **number of nodes** within each layer of the network

- The number of hidden layers, typically:
  - 1 layer has the ability to approximate any linear decision area (**semiplane**);
  - 2 layers approximate any continuous decision area (**convex regions**);
  - 3 layers approximate any decision area (**arbitrary regions**).







Number of nodes on each layers

- The **number of input nodes** is predetermined by the **number of features** in the input data;
- The **number of output nodes** is predetermined by the **number of outcomes** to be modeled or the number of class levels in the outcome;
- The **number of hidden nodes** is left to the user to decide prior to training the model there is no reliable rule to determine the number of neurons in the hidden layer:

  - A **greater number of neurons will result in a model that more closely mirrors the training data**, but this runs a risk of overfitting -> it may generalize poorly to new data

  - Large neural networks can also be computationally expensive and slow **to train**

  - A **short number might not be enough** to model the decision area desired;

  - **Values between half and double** of the number of neurons in the input layer should be tested.

Use the fewest nodes that result in adequate performance on a validation dataset

Universidade do Minho
Departamento de Informática
Synthetic Intelligence Lab

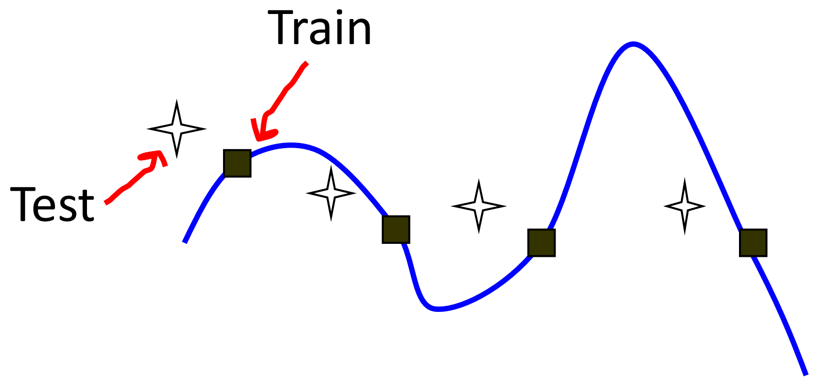
Generalization/overfitting

Overtraining an ANN *may prevent* generalization by overfitting – The ANN memorizes training cases and not generalization rules - training can be stopped early!

Regularization can be used in a similar way to linear/logistic regression (in ANNs it is called decay)

Probability of overfitting increases if:

- **Few training cases** (sample quality)
- **Too many connections** (net complexity)



Training process "a Better Model"

Underfitting

This is a model that lacks the necessary complexity to correctly capture the complexity inherent to the problem it is intended to solve. **You can recognize this situation when the error is too large, both in the training cases and in the test (validation) cases**.

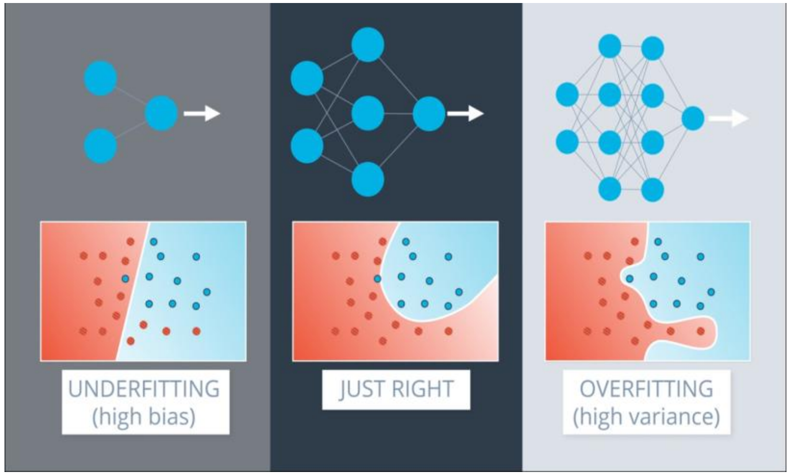
Overfitting

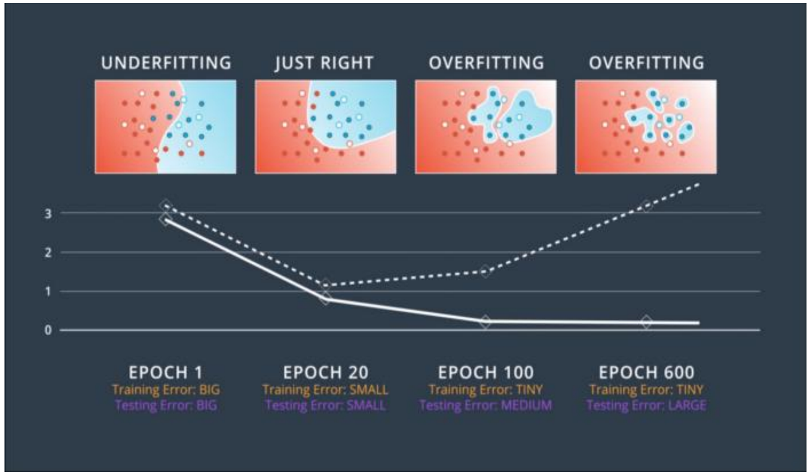
This is a model that is using too many parameters and has been trained too much.
Specifically, it has learned to identify exactly each case in the training set, becoming so specific that it cannot generalize to similar images. **You can recognize this situation when the error in the training cases is much smaller than in the test (validation) cases**.

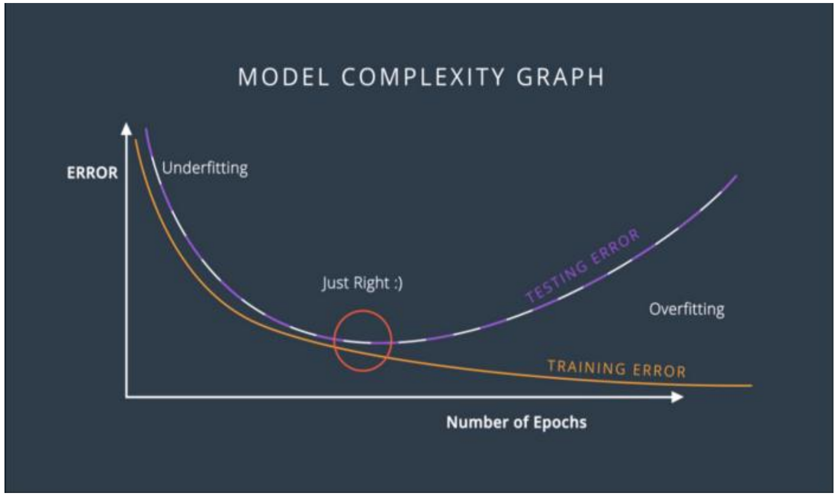
Measures you can take to reduce **overfitting**:

- Adding **more cases** to the training set
- Use architectures that have been shown to generalize well
- Reduce the complexity of the **network architecture**
- Use **data augmentation**
- Add normalization (**Batch Normalization layer**)
- Add Dropout (**Dropout layer**)

Training process "a Better Model"



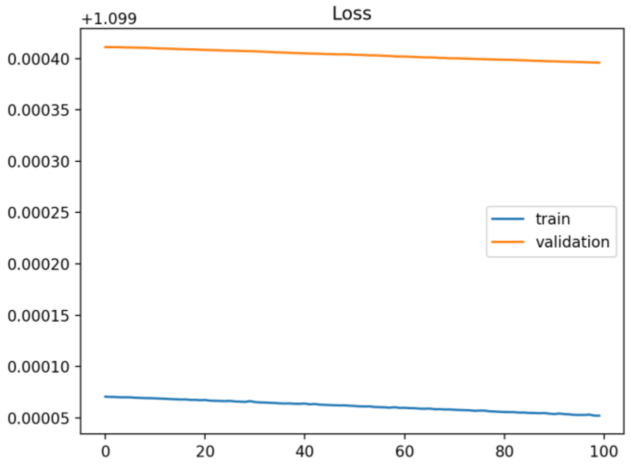




Universidade do Minho
Departamento de Informática

Synthetic Intelligence Lab

Training process "Learning Curves"



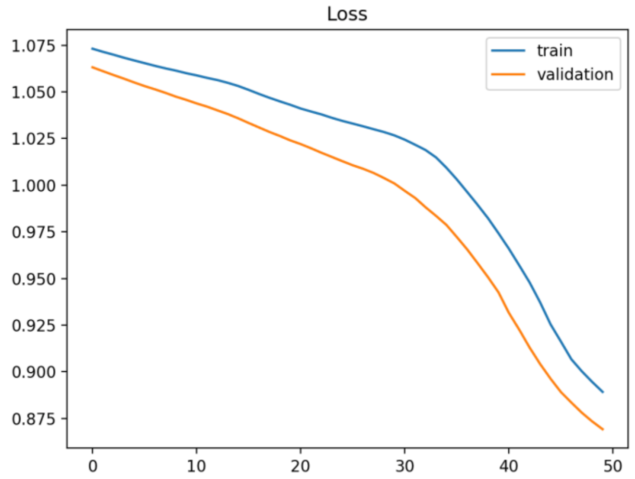
1: Example of an **Underfit Model** that **Does Not Have Sufficient Capacity**. It may show a **flat line** or **noisy values** of relatively **high loss**, indicating that the model was unable to learn the training dataset at all.

- **Add more observations**. You may not have enough data for the existing patterns to become strong signals.
- **Add more features**. Occasionally our model is under-fitting on the grounds that the feature items are insufficient.
- **Reduce any regularization on the model**. If you have explicit regularization parameters specified (i.e. dropout, weight regularization), remove or reduce these parameters.
- **Increase model capacity**. Your model capacity may not be large enough to capture and learn existing signals.

Universidade do Minho
Departamento de Informática

ISLab
Synthetic Intelligence Lab

Training process "Learning Curves"



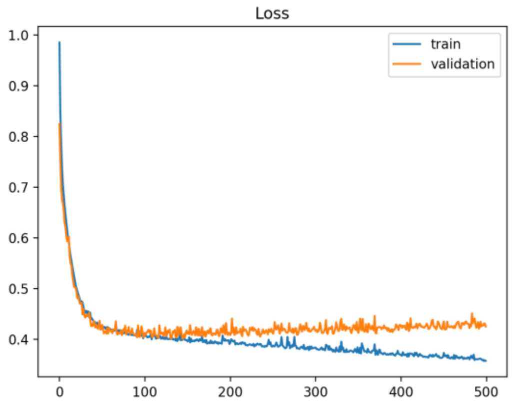
2: Example of an **Underfit Model** that **Requires Further Training**. An underfit model may also be identified by a training loss that is decreasing and continues to decrease at the end of the plot. This indicates that the model is capable of further learning and possible further improvements and that the training process was halted prematurely.

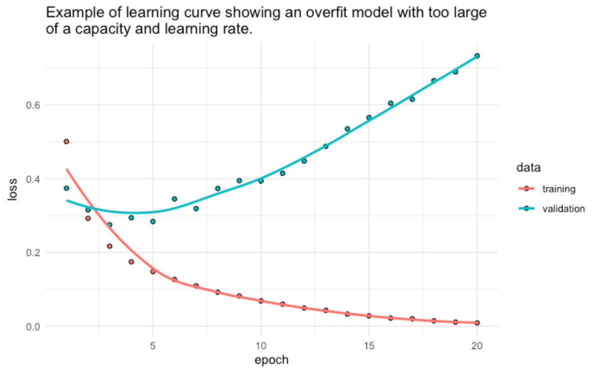
- **Increase the number of epochs** until the validation curve has stopped improving. This is a good time to crank up the epochs and add an early stopping callback to identify how many epochs are required.
- If it is taking a long time to reach a minimum for the validation curve, **increase the learning rate** to speed up the gradient traversal and also add a callback to automatically adjust the learning rate.

Universidade do Minho
Departamento de Informática

ISLab
Synthetic Intelligence Lab

Training process “Learning Curves”





3: Example of an **Overfit Model**. The inflection point in validation loss may be the point at which training could be halted as experience after that point shows the dynamics of overfitting.

- Regularize how quickly the model learns by **reducing the learning rate**. Add a callback to automatically reduce the learning rate as the validation loss plateaus.

- Regularize model capacity by **reducing the number and/or size of the hidden layers**.

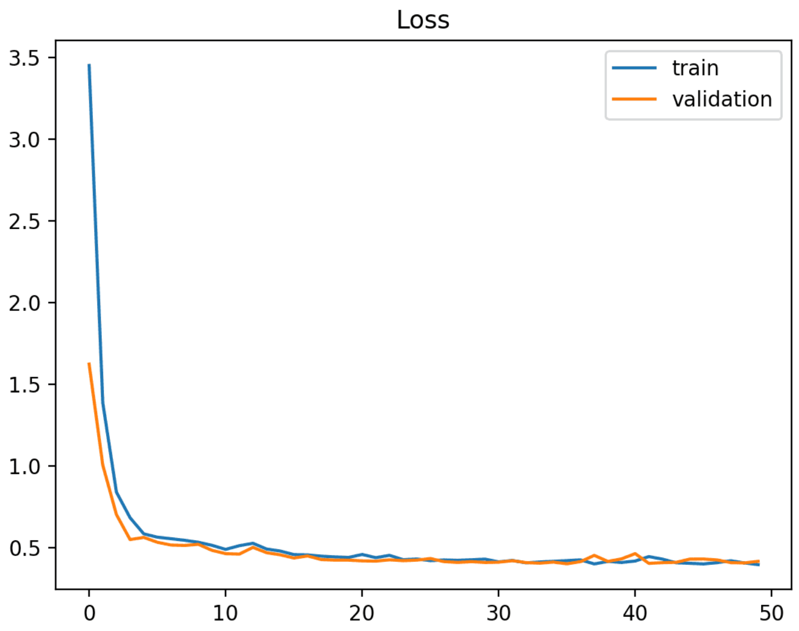
- Regularize the weights to constrain the complexity of the network.

- Regularize happenstance patterns by adding dropout to minimize the chance of fitting patterns to noise in the data.

Universidade do Minho
Departamento de Informática

ISLab
Synthetic Intelligence Lab

Training process “Learning Curves”

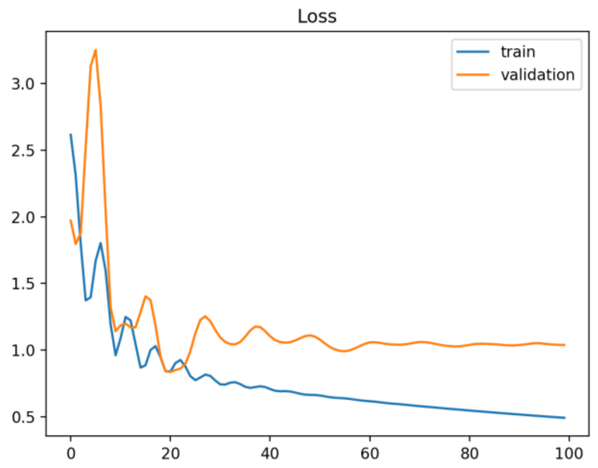


4: Example of Train and Validation Learning Curves showing a **Good Fit**

Universidade do Minho
Departamento de Informática

ISLab Synthetic Intelligence Lab

Training process "Learning Curves"



Diagnosing Unrepresentative Datasets

5: Example of train and validation learning curves showing **a training dataset that may be too small relative to the validation dataset**. This situation can be identified by a learning curve for training loss that shows improvement and similarly a learning curve for validation loss that shows improvement, but a large gap remains between both curves

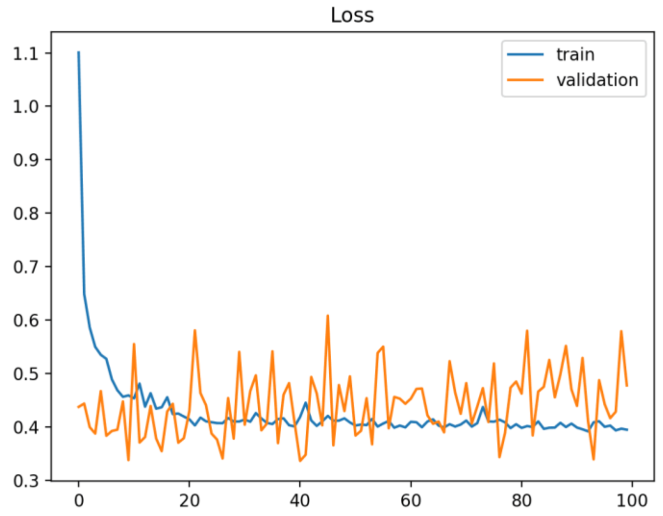
- **Add more observations**. You may not have enough data to capture patterns present in both the training and validation data.

- Make sure that you are **randomly sampling observations** to use in your training and validation sets. If your data is ordered by some feature (i.e. neighbourhood, class) then you validation data may have features not represented in your training data.

- Perform **cross-validation** so that all your data has the opportunity to be represented in both the training and validation sets.

Universidade do Minho
Departamento de Informática
ISLab
Synthetic Intelligence Lab

Training process "Learning Curves"

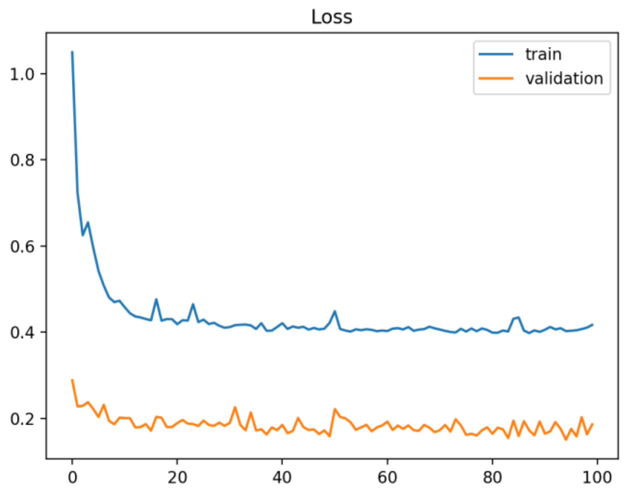


Diagnosing Unrepresentative Datasets

6: Example of train and validation learning curves showing a **validation dataset that may be too small relative to the training dataset**. This case can be identified by a learning curve for training loss that looks like a good fit (or other fits) and a learning curve for validation loss that shows noisy movements around the training loss.

- **Add more observations to your validation dataset**.
- If you are limited on the number of observations, **perform cross-validation** so that all your data has the opportunity to be represented in both the training and validation sets.

Training process "Learning Curves"



Diagnosing Unrepresentative Datasets

7: Example of train and validation learning curves showing a **validation dataset that is easier to predict than the training dataset**. It may also be identified by a validation loss that is lower than the training loss.

- Check to make sure **duplicate observations** do not exists across training and validation datasets.
- Check to make sure there is no **information leakage** across training and validation datasets.
- Make sure that you are **randomly sampling observations** to use in your training and validation sets so that feature variance is consistent across both sets.
- Perform cross-validation so that all your data has the opportunity to be represented in both the training and validation sets.

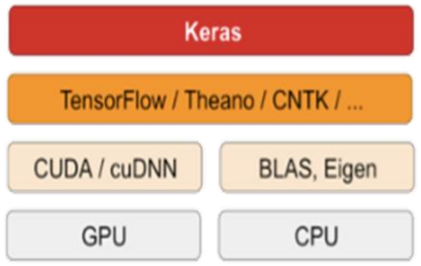
Artificial Neural Networks

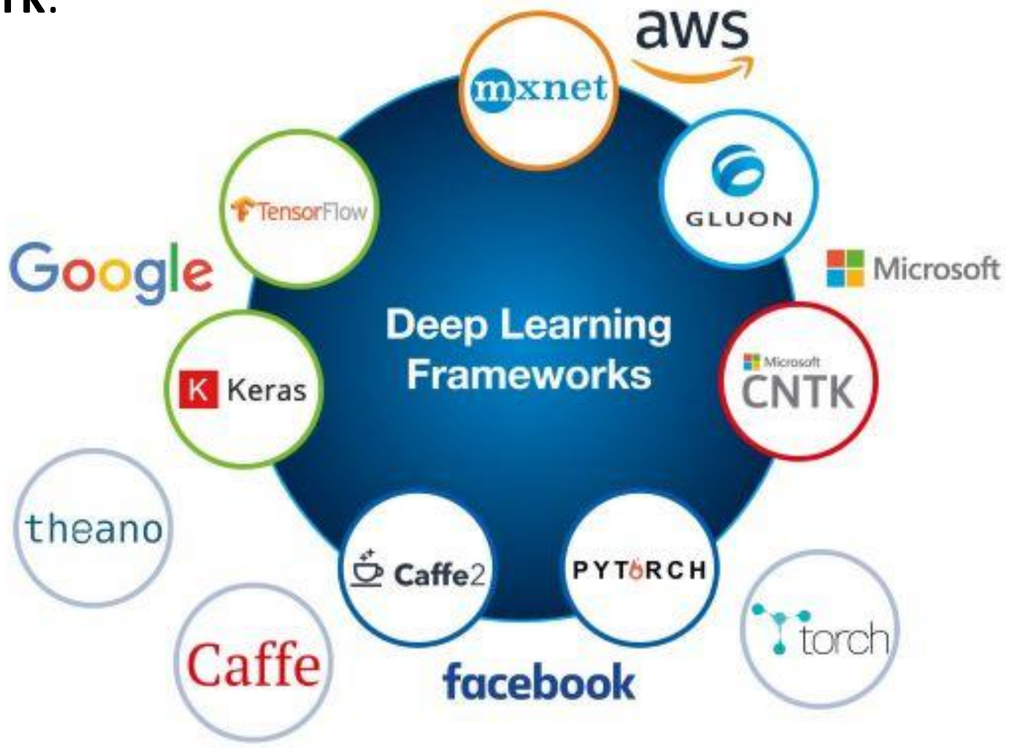
- **Strengths**
  - Classification **accuracy is usually high**, even for complex problems;
  - **Distributed processing**, knowledge is distributed through connection weights;
  - Robust in handling examples even if they contain errors;
  - **Handle redundant attributes well**, since the weights associated with them are usually very small;
  - Results can be discrete, real values, or a vector of values (discrete or real).
- **Weaknesses**
  - **Difficult to determine the optimal network** topology for a problem;
  - **Difficult to use** - have many parameters to define;
  - Requires specific data pre-processing;
  - Requires **long training time**;
  - **Difficult to understand** the learning function (weights);
  - **Discovered knowledge** is not readable;
  - Do not provide explanations of the results;
  - Domain knowledge incorporation is not easy.

Frameworks

Fundamental components of any **DL framework**:

- **The Tensor Object**
- **Operations on the Tensor object**
- **Graph computing and optimisation**
- Automatic **differentiation tools**
- Extensions **BLAS** / **cuBLAS** and **cuDNN**


Keras
TensorFlow / Theano / CNTK / ...
CUDA / cuDNN
BLAS, Eigen
GPU
CPU


Deep Learning Frameworks
aws
mxnet
GLUON
Microsoft
CNTK
PYTORCH
torch
facebook
Caffe2
Caffe
theano
Keras
TensorFlow
Google