Data Mining

Predictive Modelling

Artificial Neural Networks

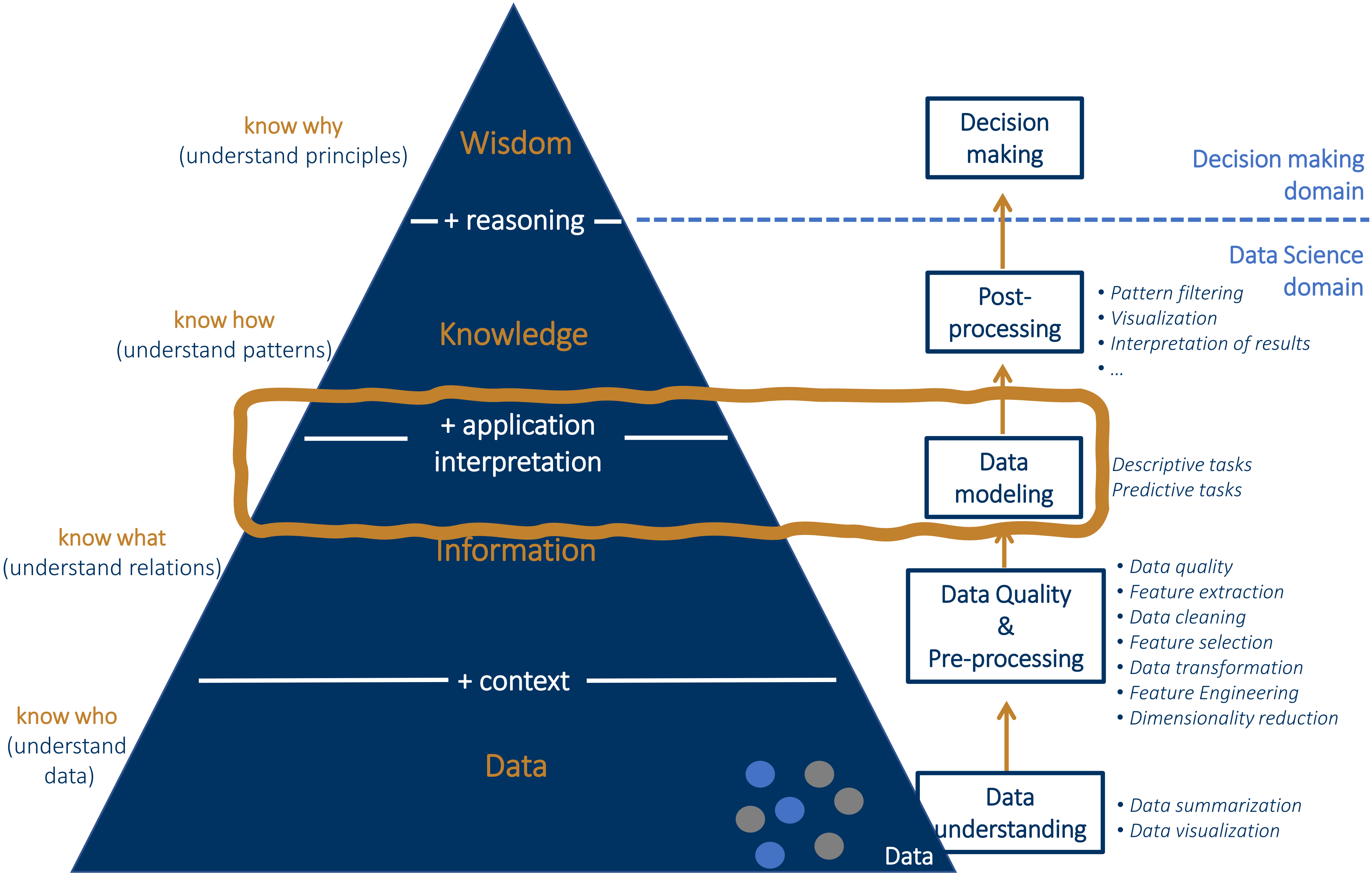
Raquel Sebastião

Departamento de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

raquel.sebastiao@ua.pt

2022/2023


know why
(understand principles)
Wisdom
+ reasoning
know how
(understand patterns)
Knowledge
+ application
interpretation
know what
(understand relations)
Information
+ context
know who
(understand data)
Data
Data
Decision making
Decision making domain
Data Science domain
Post-processing
• Pattern filtering
• Visualization
• Interpretation of results
• ...
Data modeling
Descriptive tasks
Predictive tasks
Data Quality & Pre-processing
• Data quality
• Feature extraction
• Data cleaning
• Feature selection
• Data transformation
• Feature Engineering
• Dimensionality reduction
Data understanding
• Data summarization
• Data visualization

Prediction Models – approaches

Geometric approaches

- **Distance-based:** kNN
- **Linear models:** Fisher's linear discriminant, perceptron, logistic regression, SVM (w. linear kernel)

Probabilistic approaches

- naive Bayes, logistic regression

Logical approaches

- classification or regression trees, rules

Optimization approaches

- neural networks, SVM

Sets of models (ensembles)

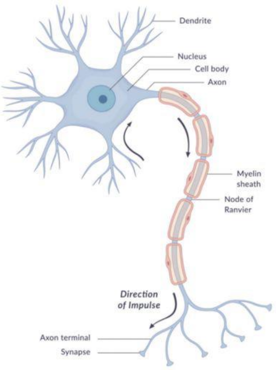
- random forests, adaBoost

Contents

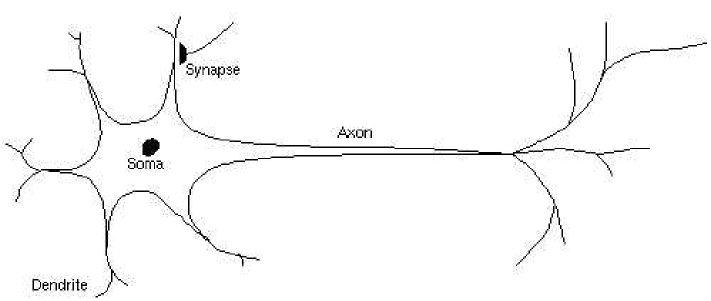
- Artificial Neural Networks
- Deep Learning (very short-introduction)

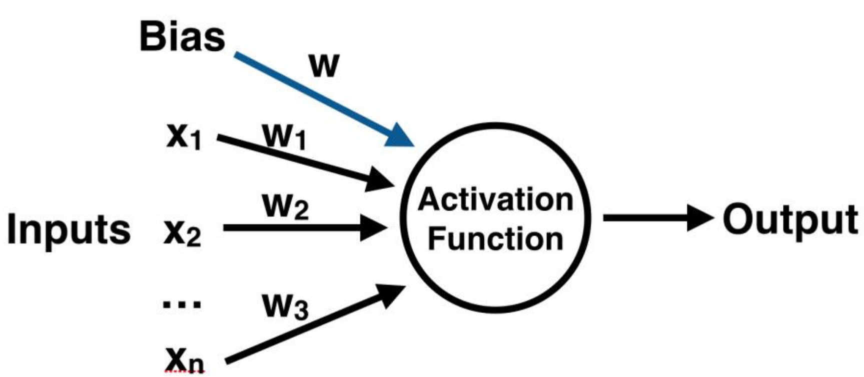
Artificial Neural Networks (ANN): Biological inspiration

- unit (neuron) has multiple inputs and one output
- network composed of highly interconnected processing units
- the weights of the connections are the adaptive elements
- inspired on brain structure



The computational model of the unit (neuron)





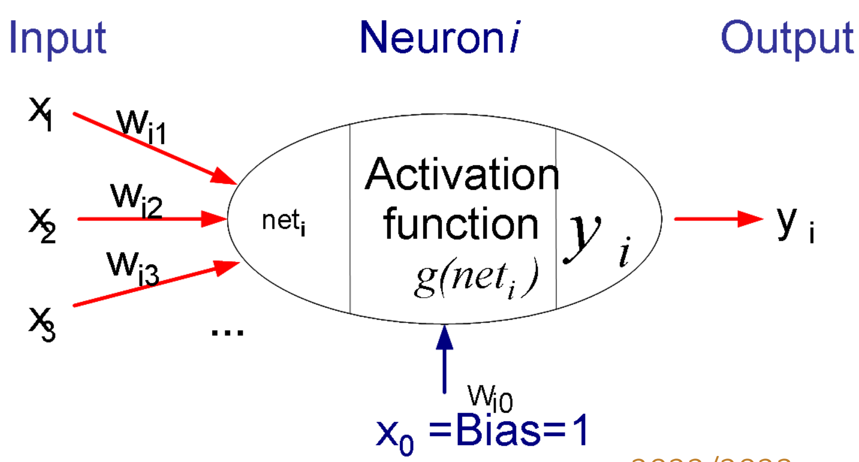
Artificial Neural Networks (ANN)

Each unit

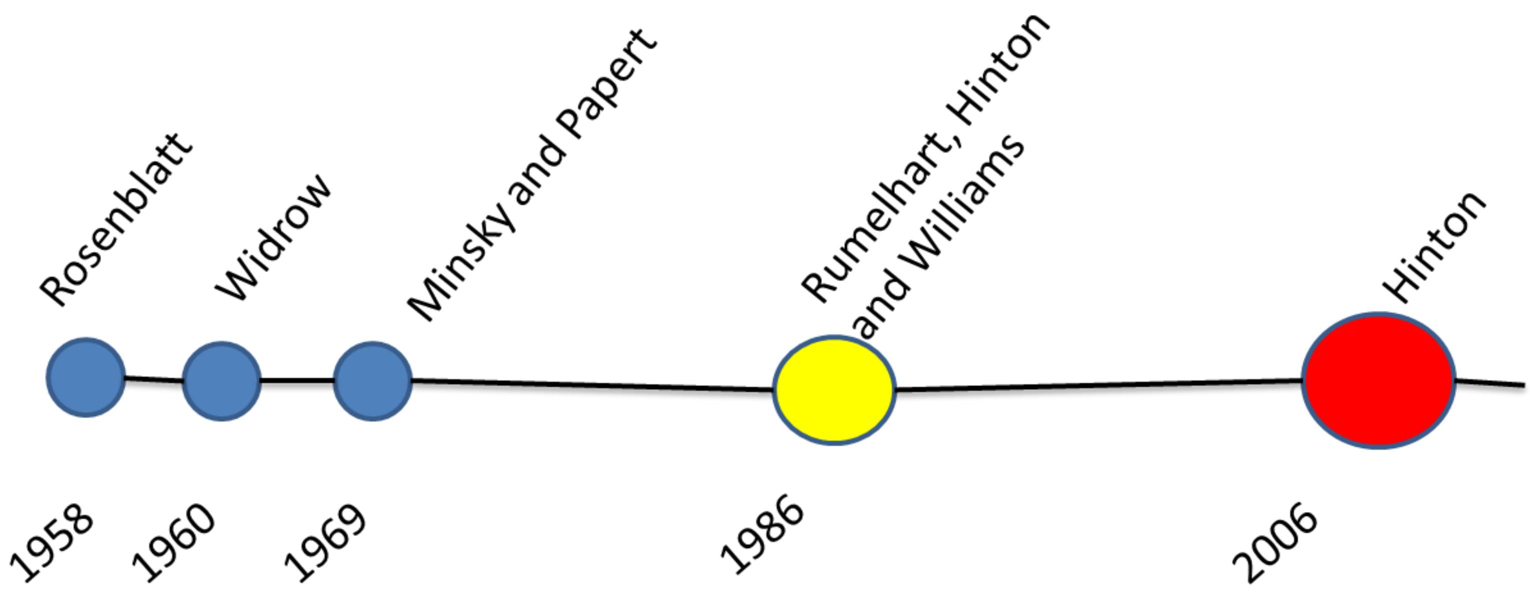
- receive the **input** impulses and calculate its **output** as a function of these impulses

This calculation is divided:

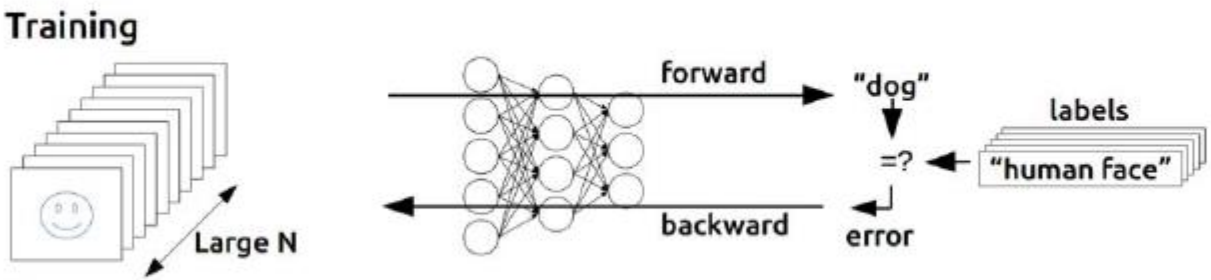
- linear combination of the inputs: $net_i = \sum_i w_{ij}^{(l)} x_i + b$
- application of activation function: $y_i = g(net_i)$ (typically non-linear)



Artificial Neural Networks (ANN): History



Artificial Neural Networks (ANN): History

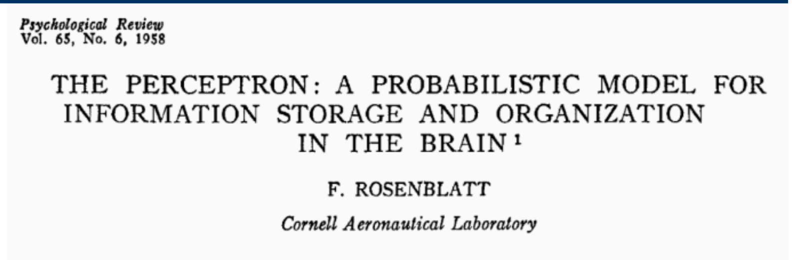


- Neural Networks need larger amounts of data to optimize
- Experimentally, training multi-layer feedforward networks was not useful
  - the accuracy didn’t improve with more layers

Around 1998 SVM and kernel based methods become popular.

- Once again Neural Networks were putted aside.

Artificial Neural Networks (ANN): Perceptron


Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

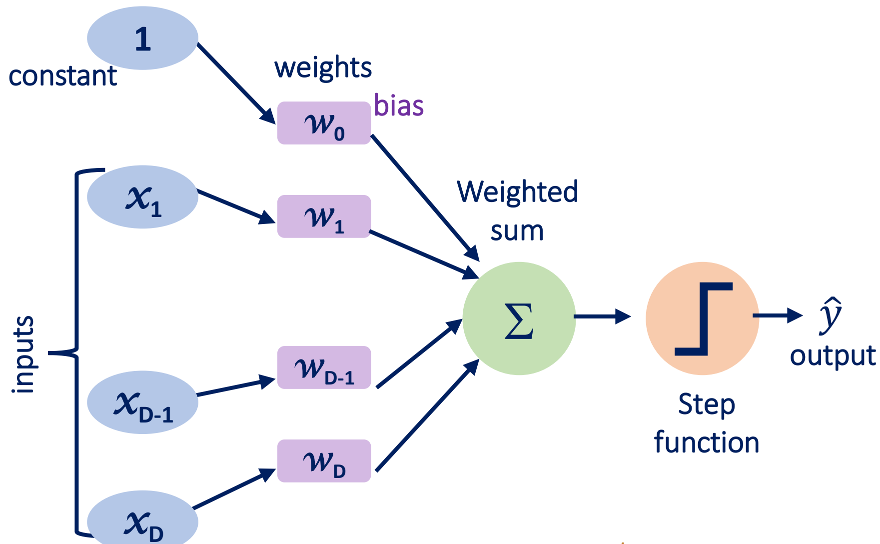
Cornell Aeronautical Laboratory

- Proposed by Rosenblatt (1958)

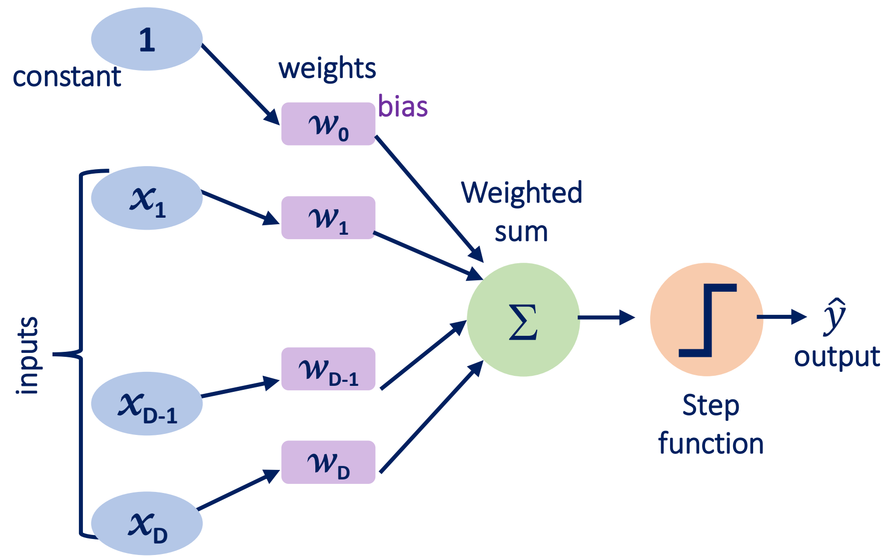
Perceptrons are the simplest ANN:

- Only **one input** layer
- Only **one output** layer

- Learn **linear** decision boundaries
- **Binary** problems



Artificial Neural Networks (ANN): Perceptron



- **Weighted** sum of the inputs:

$$a = \sum_d w_d x_d + w_0 = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$$

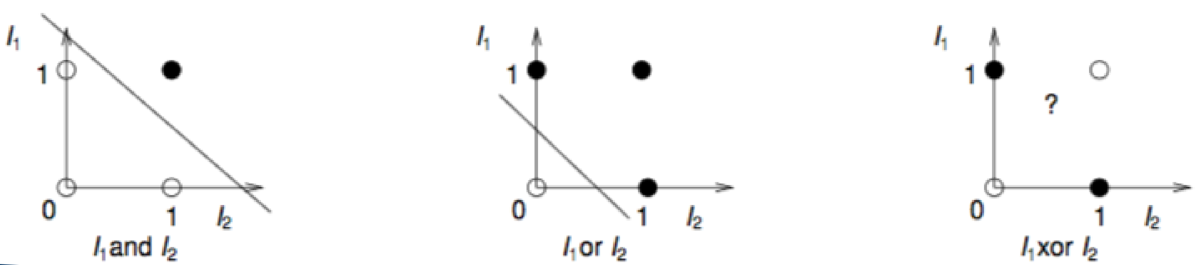
- $f(\cdot)$ **activation function**: step function

$$f(\mathbf{x}) = \begin{cases} +1, & \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 > \mathbf{0} \\ -1, & \text{otherwise} \end{cases}$$

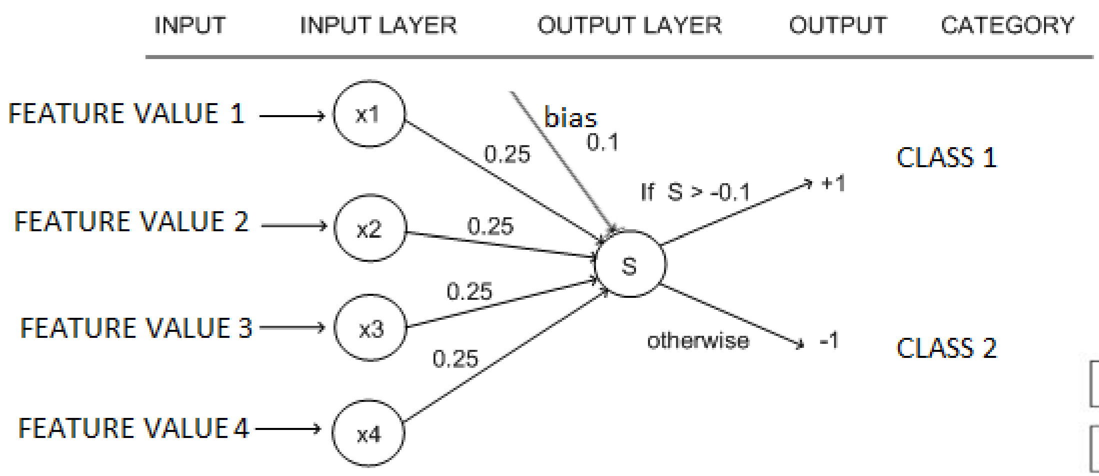
- It learns by **updating the weights** (only updates when misclassification occurs)

$$\boldsymbol{W}_i^{(k+1)} = \boldsymbol{W}_i^{(k)} + \lambda\left(y_i - \hat{y}_i^{(k)}\right)\boldsymbol{X}_i \qquad (\lambda \text{ is the learning rate})$$

- Perceptrons **are limited to linearly separable problems**

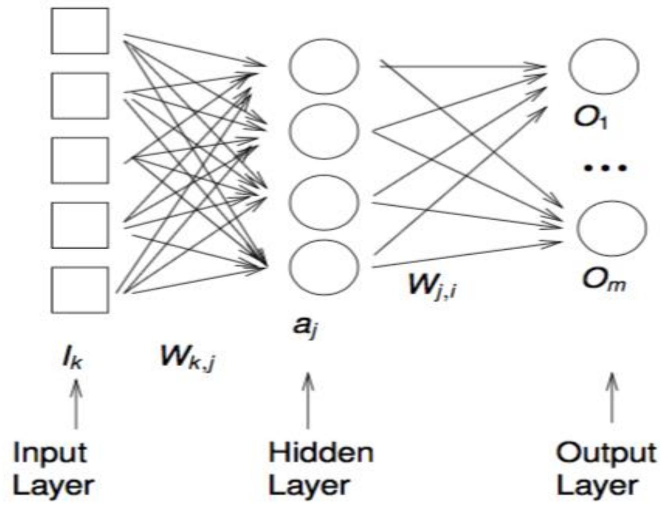


Artificial Neural Networks (ANN): Extending the Perceptron …



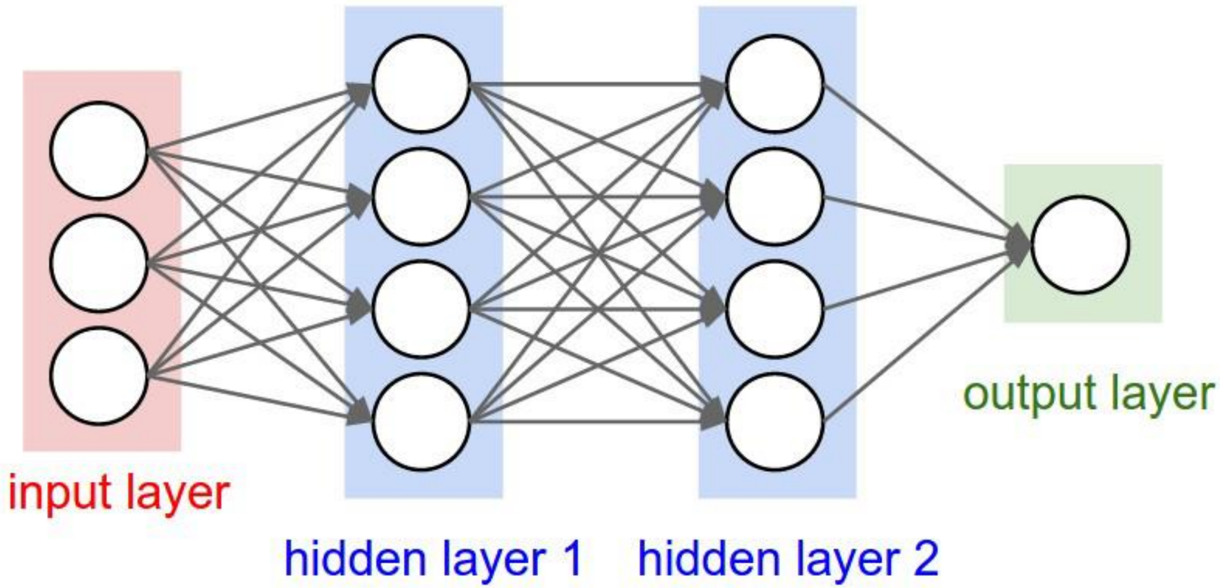
Perception: precursor for Neural Networks!

Extending Perceptron: connecting units together into multilayer Neural Networks!



Artificial Neural Networks (ANN)

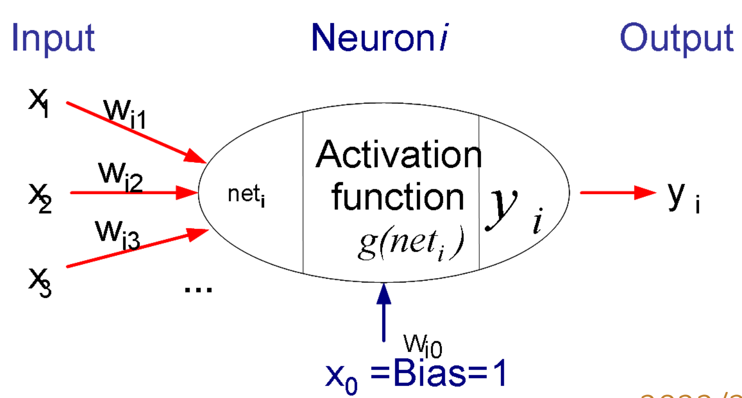
Feedforward Neural Networks (Multilayer perceptrons)



Each unit

$$net_i = \sum_i w_{ij}^{(l)} x_i + b$$

$$y_i = g(net_i)$$



Artificial Neural Networks (ANN)

Input and Output layers are the interface with the ”outside”

- Input: reads information. For instance, pixels of one image
- Output: writes information. Digit Classification: 10 units each identifying one digit

Configuration of the network (user’s choice)

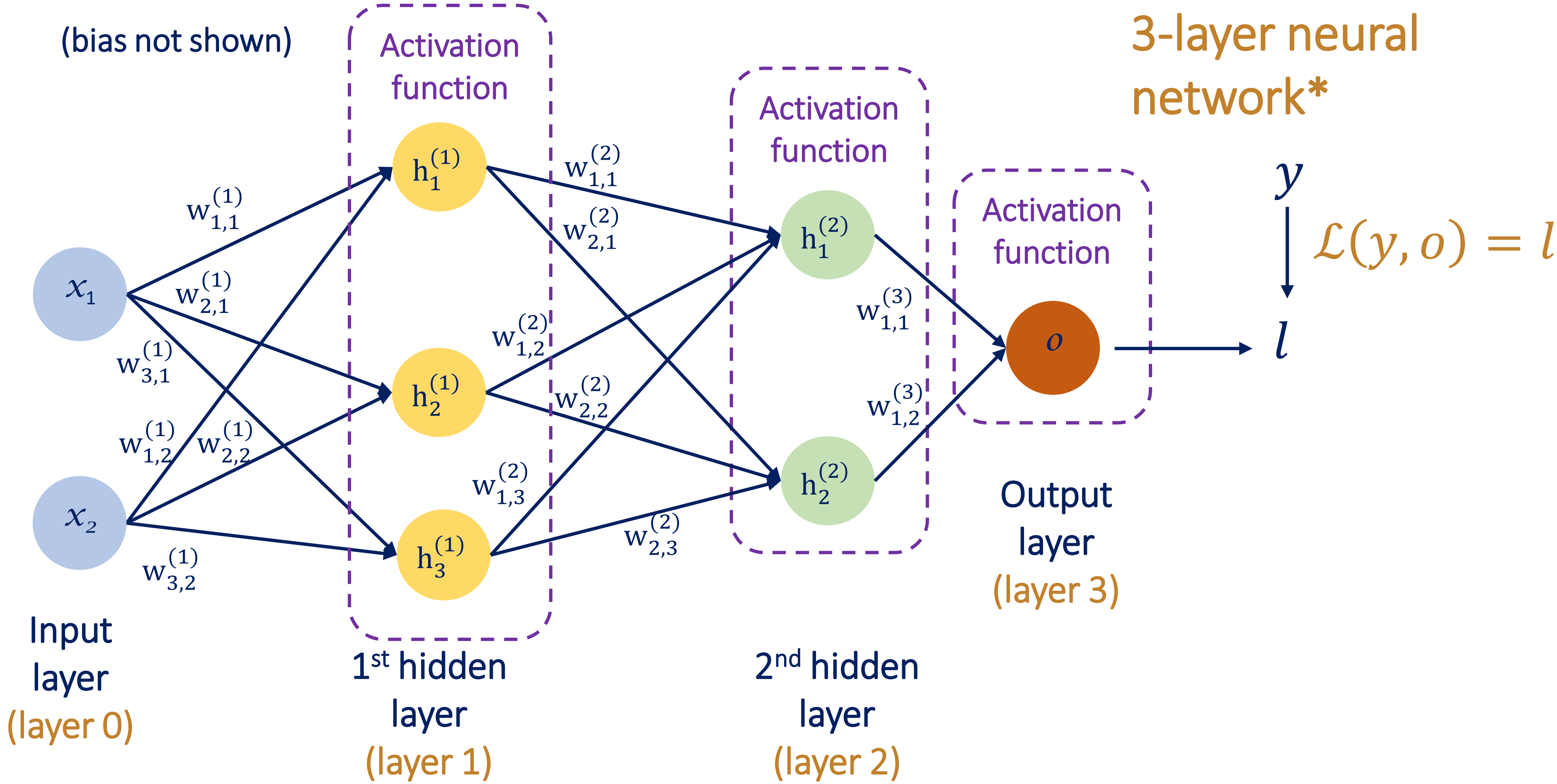
- number of hidden layers: groups of units with the same activation function
- number of units per layer
- connections between layers and units. **Fully connected**: all units of layer L are connected to the following layer $L+1$
- activation functions of each layer

Feedforward: the information flows from input to output

- Each layer is **feeding** the next layer: output of layer L is the input of layer $L+1$

The **Weights of all connections** are adapted during **learning phase**

Artificial Neural Networks (ANN): Feedforward NN – Binary classification



Perceptron and **logistic regression** model can be called a "**1-layer neural network**"

* number of layers = number of layer of adaptive weights

Artificial Neural Networks (ANN): Feedforward NN

- inputs are combined with the initial weights in a weighted sum

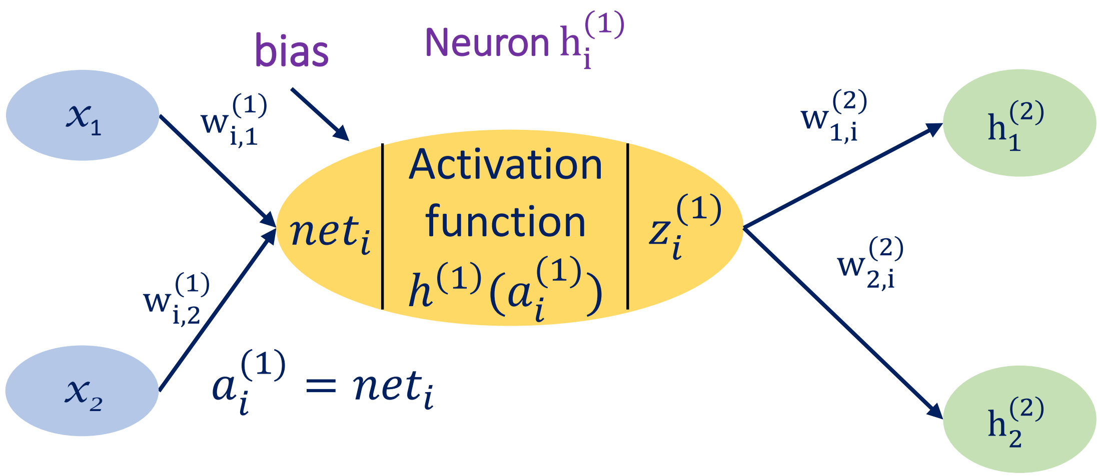
$$net_i = \sum_i \mathrm{w}_{ij}^{(l)} \mathrm{x}_i + b$$

- application of the activation function: $y_i = g(net_i)$ (typically non-linear)

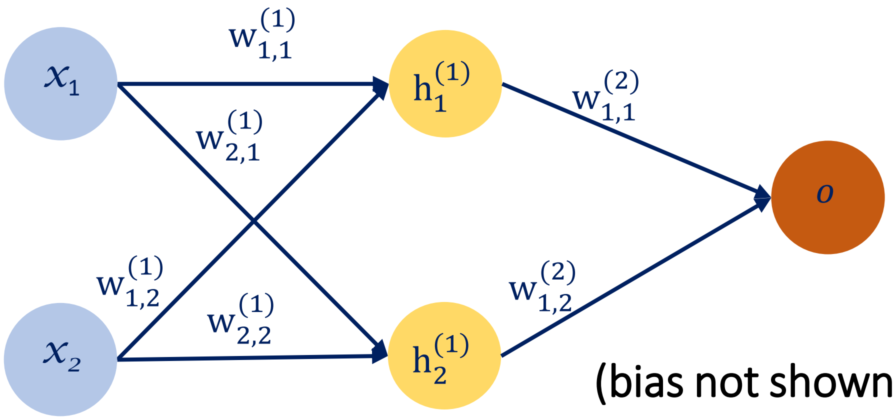
each linear combination is propagated to the next layer

Each layer is **feeding** the next layer

- The inputs of layer **L+1** are the outputs of layer **L**



Artificial Neural Networks (ANN): Feedforward NN



(bias not shown)

2-layer feedforward neural network:

- 1 input layer
- 1 hidden layer with 2 neurons
- 1 output layer (o) with one output variable:

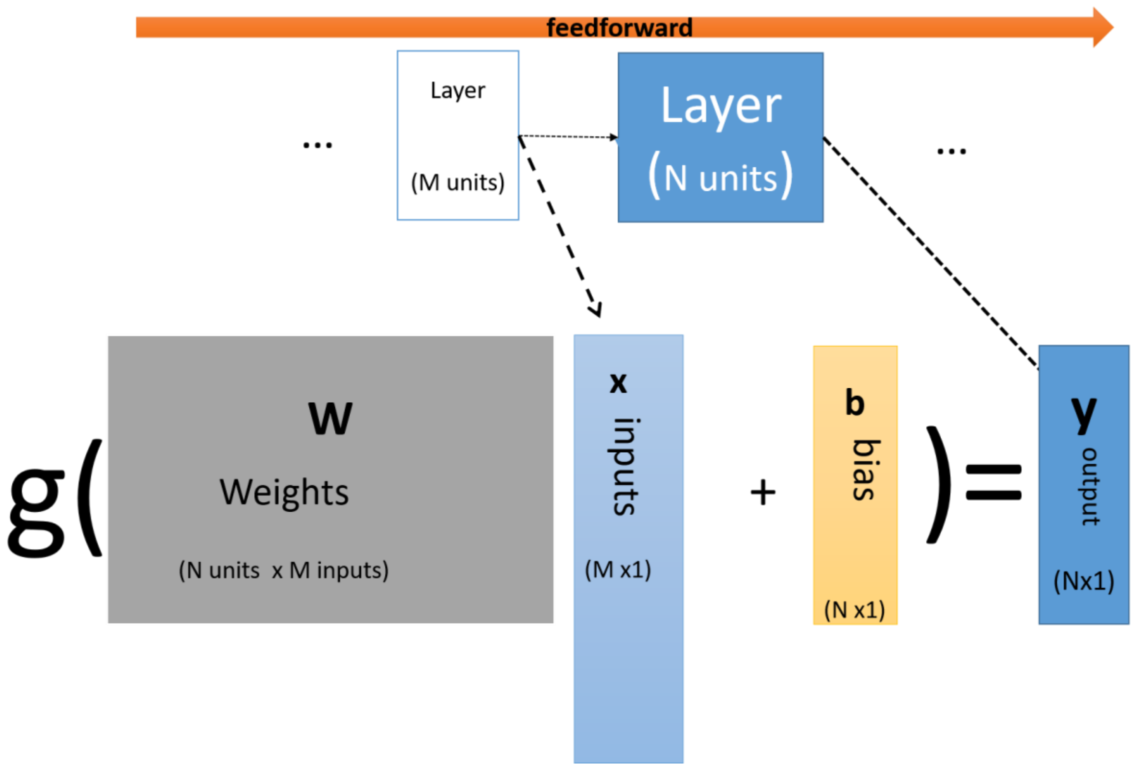
$$o = a_3 = g\left(w^{(2)}_{1,1} a_1 + w^{(2)}_{1,2} a_2\right) =$$

$$= g\left(w^{(2)}_{1,1} g\left(w^{(1)}_{1,1} x_1 + w^{(1)}_{1,2} x_2\right) + w^{(2)}_{1,2} g\left(w^{(1)}_{2,1} x_1 + w^{(1)}_{2,2} x_2\right)\right)$$

- $g(\,)$ is the activation function

Learning Phase: Calculation of the weights of the connections between units (layers)

Artificial Neural Networks (ANN): Feedforward NN



The inputs are propagated from input to output

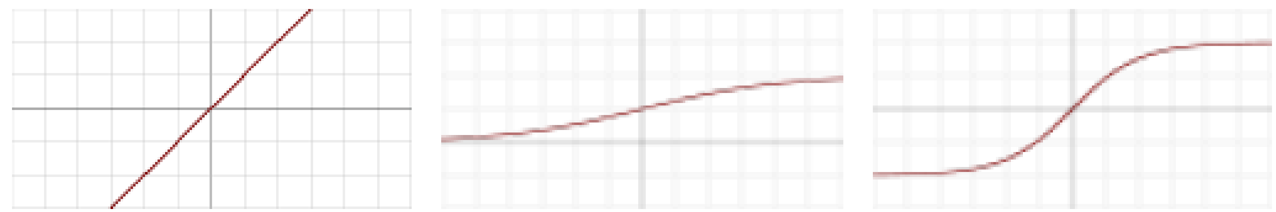
- The units of a layer process the outputs of the previous layer

Artificial Neural Networks (ANN): Activation functions

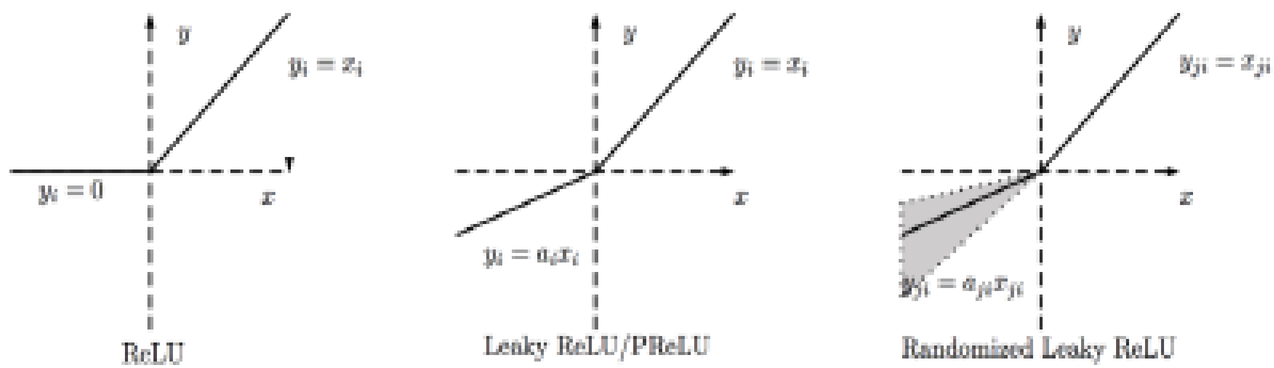
Activation functions are used to determine the output of each node of the neural network

- **linear**
- **non-linear**: most commonly used as it allows the model to generalize or adapt with variety of data

Linear | Sigmoid/logistic | Tangent Hyperbolic

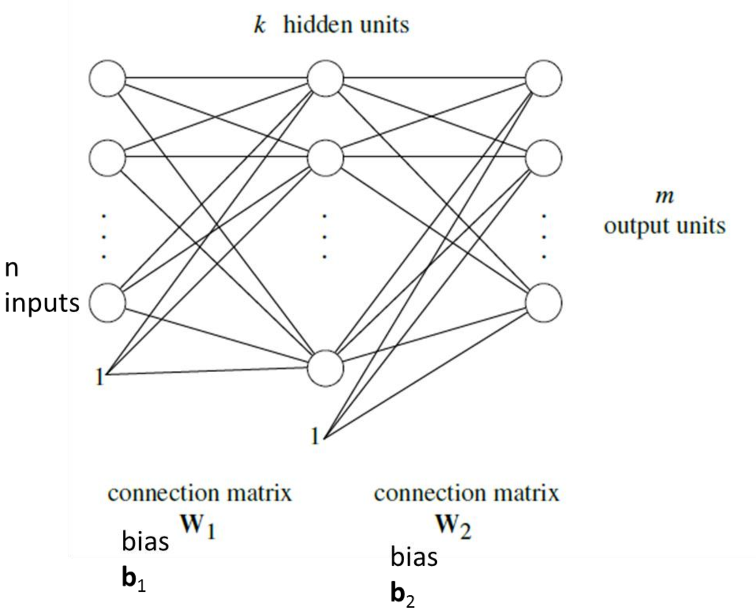


$y = g(a) = a$ | $y = g(a) = \frac{1}{1+e^{-a}}$ | $y = g(a) = tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$



Artificial Neural Networks (ANN): Non-linear activation functions

Non-linear activation functions in hidden layers are a must



- hidden layer activation function: sigmoid
- output layer activation function: linear
- The output of the network

$$o = W_2 g(W_1 x + b_1) + b_2$$

where g() is the activation function

With an activation function linear

$$W = W_2 W_1 \qquad b = W_2 b_1 + b_2$$

- **Possible** to find an **equivalent network without hidden layers**

Artificial Neural Networks (ANN): Learning multilayer NN

- Can we apply perceptron learning rule to each node, including hidden nodes?
  - **Perceptron learning rule** computes error term $e = y - \hat{y}$ and updates weights accordingly
    - Problem: how to determine the true value of y for hidden nodes?
  - Approximate error in hidden nodes by error in the output nodes
    - Problem:
      - Not clear how adjustment in the hidden nodes affect overall error
      - No guarantee of convergence to optimal solution

Artificial Neural Networks (ANN): Gradient-based learning

- Cost/loss function $E = J(\mathbf{w}) = \sum_{k=1}^{n} Loss(y_k, \hat{y}_k)$

- **Vector gradient:** indicate the maximum of the error

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_D}\right)$$

- each **weight** is updated according to

$$\Delta w_i = -\lambda \frac{\partial E}{\partial w_i}$$

Parameters are updated in the direction of "maximum descent" in the loss function across all points

$$w_{ij}^l \longleftarrow w_{ij}^l - \lambda \frac{\partial E}{\partial w_{ij}^l},$$

$$b_i^l \longleftarrow b_i^l - \lambda \frac{\partial E}{\partial b_i^l},$$

λ: learning rate

Stochastic gradient descent (SGD): update the weight for every instance

minibatch SGD: update over min-batches of instances

Artificial Neural Networks (ANN): Computing gradients

$$\frac{\partial E}{\partial w_j^l} = \sum_{k=1}^{n} \frac{\partial \text{ Loss } (y_k, \ \hat{y_k})}{\partial w_j^l}.$$

$$\hat{y} = a^L$$

$$a_i^l = f(z_i^l) = f\Big(\sum_j w_{ij}^l a_j^{l-1} + b_i^l\Big)$$

Using chain rule of differentiation (on a single instance):

$$\frac{\partial \text{ Loss}}{\partial w_{ij}^l} = \boxed{\frac{\partial \text{ Loss}}{\partial a_i^l}} \times \frac{\partial a_i^l}{\partial z_i^l} \times \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\delta_i^l = \frac{\partial \text{ Loss}}{\partial a_i^l}$$

How to compute δ_i^l for every layer???

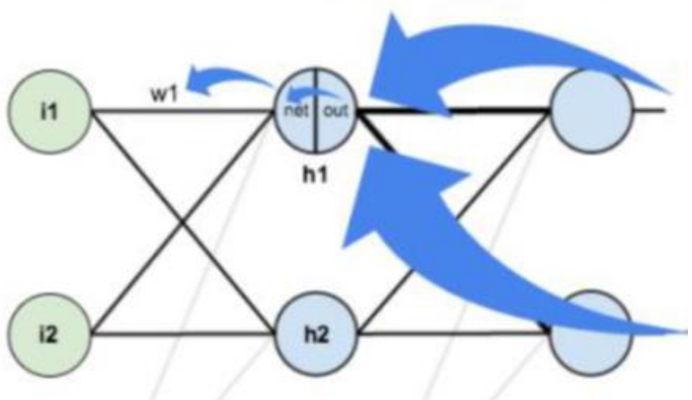
Artificial Neural Networks (ANN): Backpropagation algorithm

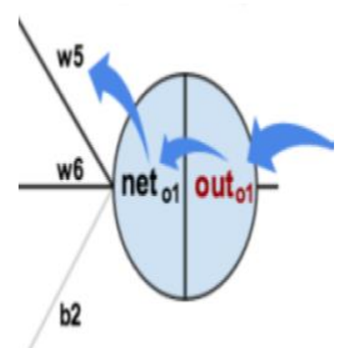
Intuition

- each unit is responsible for a certain fraction of the error in the output nodes to which it is connected
- thus, the error is divided according to the weight of the connection between the respective hidden and output units, thus propagating the errors backwards

Backpropagation computes the gradient in weight space of a feedforward neural network, with respect to a loss function

Chain rule





Artificial Neural Networks (ANN): Backpropagation algorithm

- At output layer L:

$$\delta^L = \frac{\partial \text{ Loss}}{\partial a^L} = \frac{\partial \ (y - a^L)^2}{\partial a^L} = 2(a^L - y)$$

- At a hidden layer L (using chain rule):

$$\delta_j^l \ = \ \sum_i (\delta_i^{l+1} \times a_i^{l+1}(1 - a_i^{l+1}) \times w_{ij}^{l+1})$$

  - Gradients at layer L can be computed using gradients at layer $L + 1$
  - Start from layer L and “backpropagate” gradients to all previous layers
- Use gradient descent to update weights at every epoch
- For next epoch, use updated weights to compute loss fn. and its gradient
- Iterate until convergence (loss does not change)

Artificial Neural Networks (ANN): Backpropagation algorithm

The algorithm **(for one hidden layer)**

- Initialize network weights (often small random values)
- Do
  - For each example in training set
    - predict the output
    - calculate the prediction error by a loss function
    - compute δ_h for all the weights from output layer to hidden layer
    - compute δ_i for all the weights from hidden layer to input layer
    - update network weights
- Until it converges
  - all examples are classified correctly or stopping criterion is satisfied
- Return the network

Artificial Neural Networks (ANN): Backpropagation algorithm

When to stop training?

- If stopping too early: risk of getting a network not yet trained
- If stopping too late: danger of overfitting (adjustment to noise in the data)
- Stopping criteria:
  - maximum number of iterations
  - error based on the training set
    - when the error in the training set is below a certain limit
  - error based on a validation set (independent from the training set)
    - when the error on the validation set has reached a minimum

Artificial Neural Networks (ANN): Some relevant hyperparameters

Network Structure

- number of layers
- number of neurons in each layer
- weights initialization
- activation function

Training Algorithm

- learning rate
- max. number of epochs
- early stopping criterion
- mini-batch size for mini-batch SGD

Artificial Neural Networks (ANN): Design issues

Network Structure

- Number of nodes in input layer:
  - One input node per **binary/continuous** attribute
  - **k** or **$\log_2$ k** nodes for each **categorical** attribute with **k** values
- Number of nodes in output layer:
  - **One output** for **binary** class problem
  - **k** or **$\log_2$ k** nodes for **k-class** problem
- Number of hidden layers

Artificial Neural Networks (ANN): Design issues

Network Structure

- Nodes per hidden layer:
  - few nodes: underfitting (network is unable to learn problem concept)
  - many nodes: overfitting (training set is memorized, thus making the network useless on new data sets)
  - there are no criteria for defining the number of nodes in the hidden layer
- Initial weights and biases

Artificial Neural Networks (ANN):
Design issues

Training Algorithm

- Learning rate (sets the size of the steps to obtain the direction of maximum descendent)
  - a small learning rate has the effect of learning times higher
  - a high learning rate may lead to non-convergence
  - Batch learning typical values are 0.001 to 0.1

η too small $\quad$ η too large

- max. number of epochs, mini-batch size for mini-batch SGD, …

Artificial Neural Networks (ANN): Pratical hints

- Preprocessing inputs of the network:
  - Data should be standardized
  - Features with very different distributions of values are not convenient, given the typical activation functions
- Missing values in input features may be represented as zeros, which do not influence the neural net training process
- Output in Multiclass Setting:
  - Use one-hot encoding, there are M output neurons (1 per class)
  - For each case, the class with the highest probability value
- Random initialization of the weights: zero mean and standard deviation equal to number $m^{-1/2}$ where m is the number of weights of the unit

Artificial Neural Networks (ANN)

Advantages

- Linear and Non-Linear in the same algorithm
- Good generalization (classification accuracy high)
- Robust, works when training examples contain errors
- Binary or multiclass problems
- Can handle redundant and irrelevant attributes because weights are automatically learnt for all attributes
- Ability to classify patterns on which they have not been trained

Disadvantages

- No criterium to choose the appropriate architecture
- Long training times (but testing if fast)
- The training can converge to a local minimum
- If the network is large, then a large training set is needed
- Not easy to interpret results (resulting models are essentially black boxes)

Artificial Neural Networks (ANN)

Use ANNs when

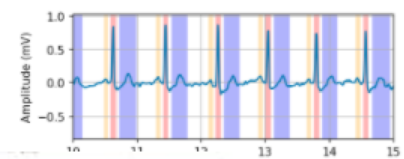
- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is a vector of values (classification or regression)
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

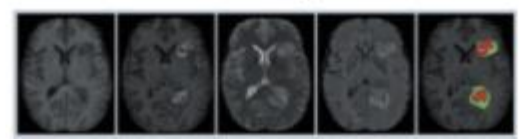
Contents

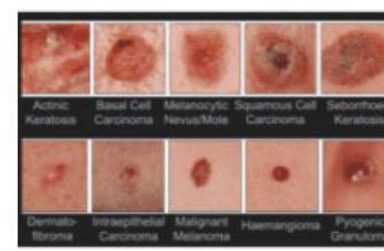
- Artificial Neural Networks
- Deep Learning (very short-introduction)

Deep Learning: where?

- Image recognition (e.g. Google, Facebook)
- Automatic text translation (e.g. Google Translator)
- Answers in natural language / digital assistants
- Games (e.g. DeepMind AlphaGo)
- Transcript of handwritten text
- Self-driving cars
- Image colorization, caption generation
- Classification of protein and DNA sequences
- Heart sound: classification and segmentation
- Tumor images detection from MRI, CT, X-rays
- Skin lesion classification from clinical and dermoscopic images
- Parkinson's disease detection from voice recording







Deep Learning

- Deep learning = Deep neural networks
  - Deep = high number of hidden layers
  - Learn a larger number of parameters!

- Training **deep neural networks** (more than **5-10 layers**) could only be possible in recent times with:
  - Faster computing resources (GPU)
  - Larger labeled training sets

Deep Neural Networks

- Algorithmic improvements in **Deep Learning**
  - Responsive activation functions (e.g., RELU)
  - Regularization (e.g., Dropout)
  - Supervised pre-training
  - Unsupervised pre-training (auto-encoders)
- **Specialized ANN Architectures**:
  - Convolutional Neural Networks (for image data)
  - Recurrent Neural Networks (for sequence data)
  - Residual Networks (with skip connections)
- Generative Models: Generative Adversarial Networks

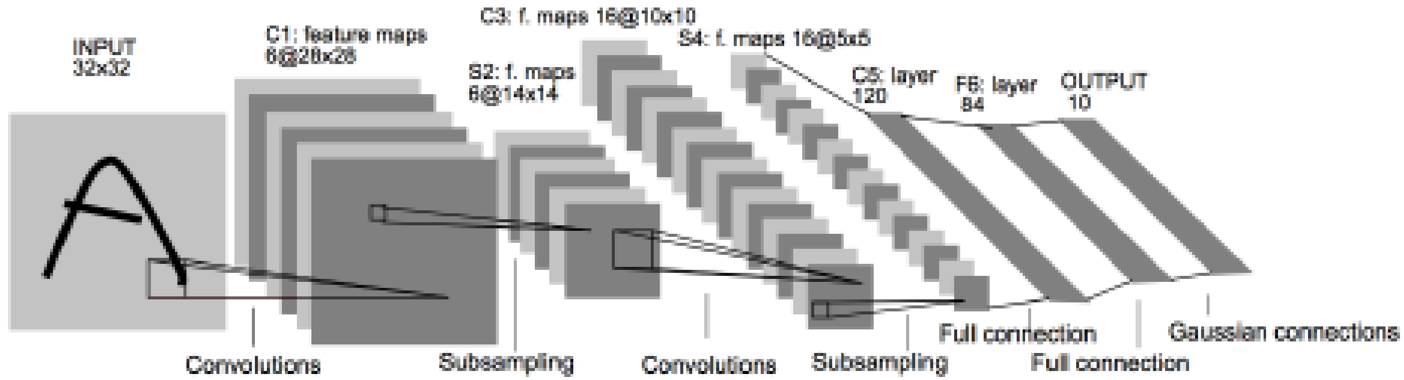
Deep Neural Networks

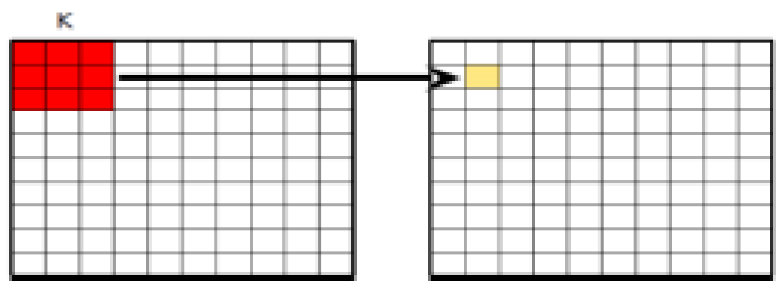
- Commonly used architectures are convolutional networks
- Almost all **CNN** architectures follow the same general design principles of
  - successively applying convolutional layers to the input,
  - periodically downsampling the spatial dimensions while
  - increasing the number of feature maps.
- Classic network architectures were comprised simply of stacked convolutional layers
- Modern architectures explore innovative ways for constructing convolutional layers for more efficient learning
- Almost all of these architectures are based on a repeatable unit

Convolution Neural Networks (CNNs)

- Feedforward neural networks
- Neurons typically use the ReLU or sigmoid activation functions
- Weight multiplications are replaced by convolutions (filters)
- Change of paradigm: can be directly applied to the raw signal, without computing first ad hoc features
- Features are learnt automatically!!

LeNet-5

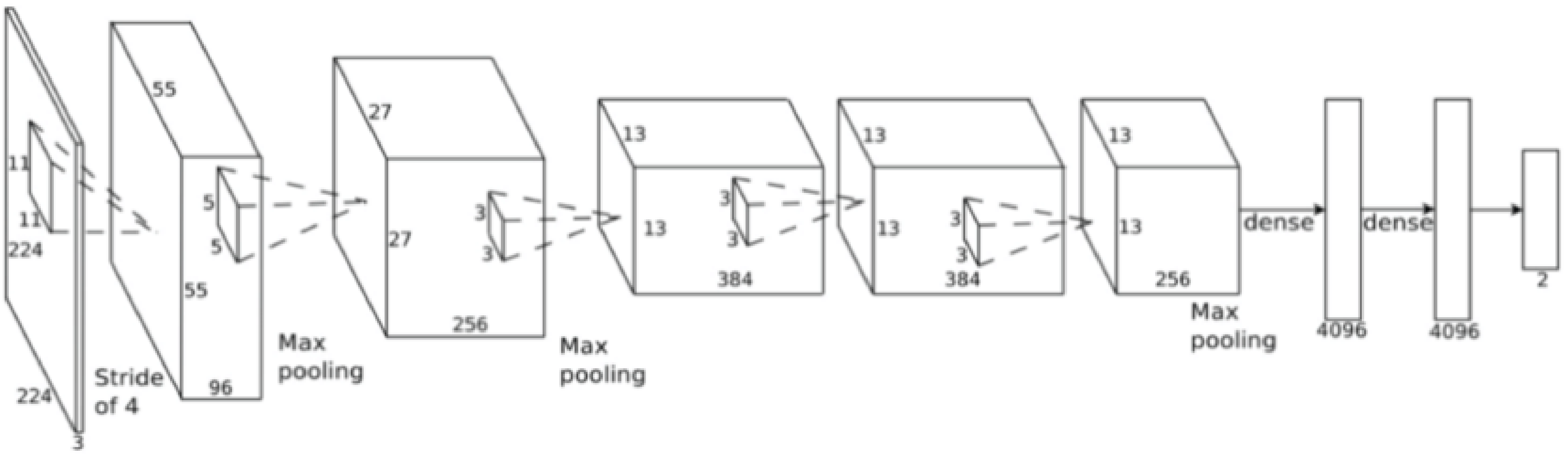




input f[m,n] $\Rightarrow$ filtered output $g[m,n]$

- Centering the mask on every pixel (m, n), $m = 1 \ldots M$; $n = 1 \ldots N$
- Convolution yields filtered output image
  $g[m, n] = \sum_{k,l} h[k, l] f[m - k, n - l]$

AlexNet



Conv Layer I: stride $S = 4$

Conv Layers II, III , IV and V: stride $S = 1$

Local Normalization: after Layer I and II

Max Pooling: after Layer I, II e V.

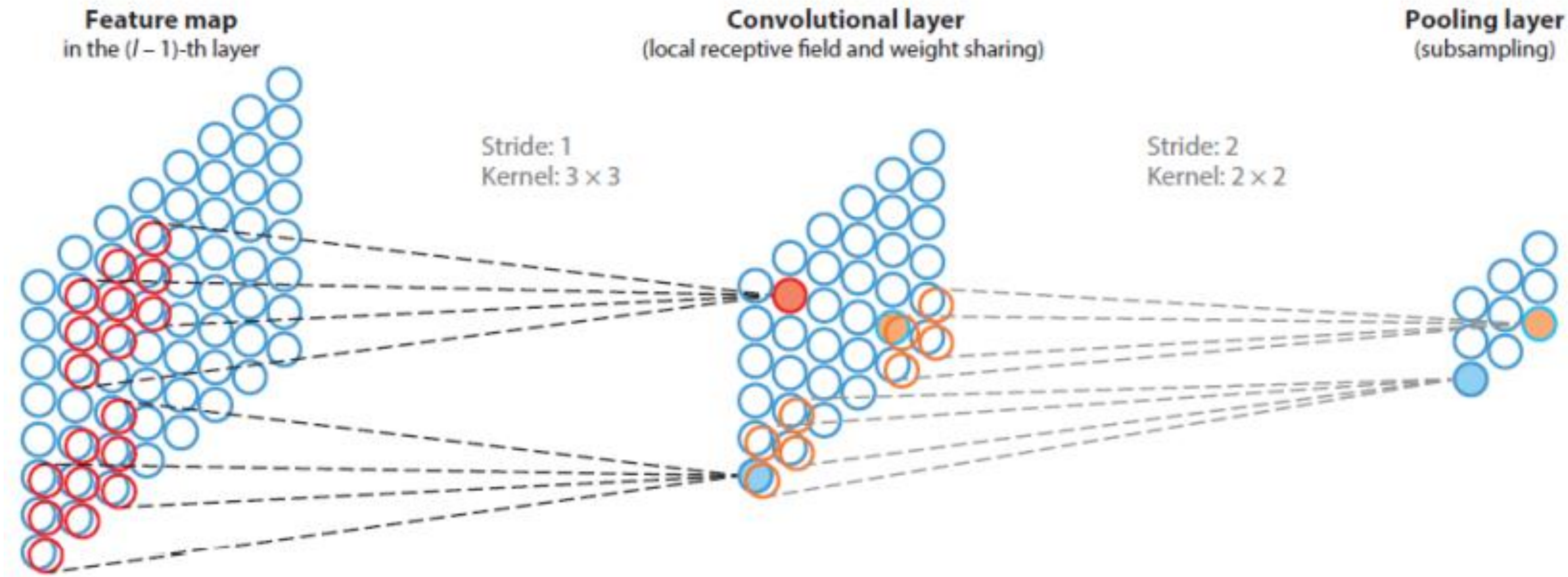
After last conv layer the outputs $13 \times 13 \times 128$ are **vectorized**

AlexNet

The parameters are

- Filter kernel K: The kernels have dimension K × K × D
- Stride S: The filter is centered in one out of S pixels
- Padding pad: The input feature maps are padded at the edges with pad pixels (zero-initialized)
- Pooling with overlap

Convolution and Pooling



- CNNs use Local Receptive Fields and scan input space
- Weight Sharing: All units of a feature map have the same weights
- Dimension reduction: pooling (max or average) or striding

Deep Learning: summarization

Great results! But...

- Like any other technique, DL does not solve all problems and will not always be the best option for any learning task
- Difficult to select best architecture for a problem
- Require new training for each task/configuration
- (Most commonly) require a large training dataset to generalize well
  - Data augmentation, weight regularization, dropout, transfer learning, etc
- Still not fully understood why it works so well
- Unstable against adversarial examples

Bibliography

Introduction to Data Mining, Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, Vipin Kumar, *Pearson*, 2019 (chap 6.7)

Data Mining, the Textbook, Charu C. Aggarwal, *Springer*, 2015 (chap 10.7)

Pattern Recognition and Machine Learning, C. Bishop, *Springer*, 2007 (ch 5)

Introduction to Machine Learning, Ethem Alpaydin, MIT Press

https://www.skynettoday.com/overviews/neural-net-history

https://sebastianraschka.com/pdf/lecture-notes/stat453ss21/L09_mlp__slides.pdf