Workshop on Deep Learning

University of Abderrahmane Mira - Bejaia

Mohand Saïd Allili

University of Quebec in Outaouais- Canada

27 November, 2018

Data analytics and machine learning

Big data

- We are entering a big data era, where masses of data are produced every day, with size beyond the ability of humans and commonly used software tools.

- For example, there are:

  ☞ ~one trillion (10^{12}) Web pages.

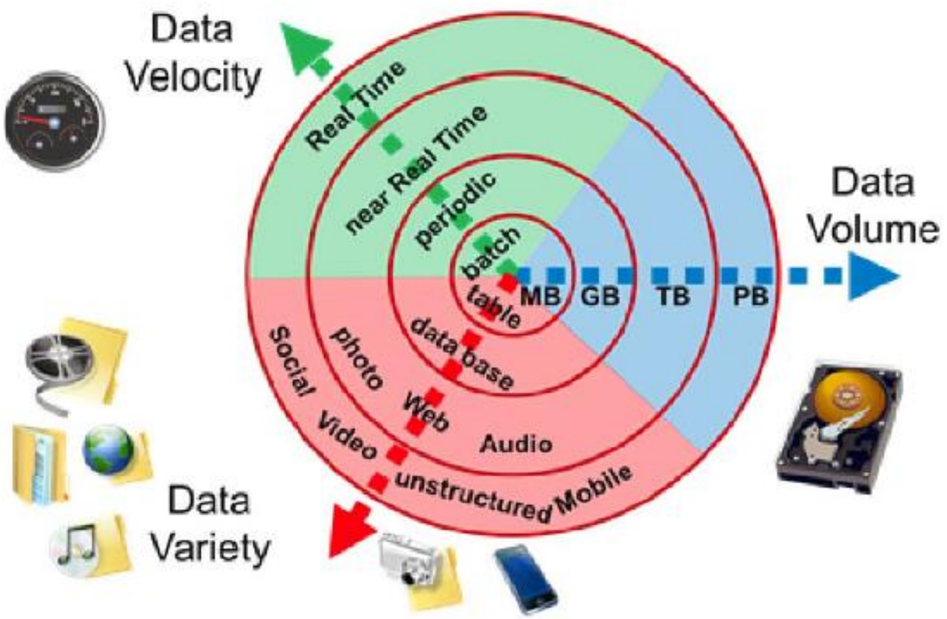
  ☞ ~one hour video uploaded each second to YouTube (2016).

  ☞ 10^6 transactions/h in **Walmart**, 2.5 petabytes (~10^{15}) of data.

  ☞ By 2025, there will be ~163 zettabytes (~10^{21}) of data.

Big data

- Big data was originally associated with three key V-concepts: *volume*, *variety*, and *velocity*.

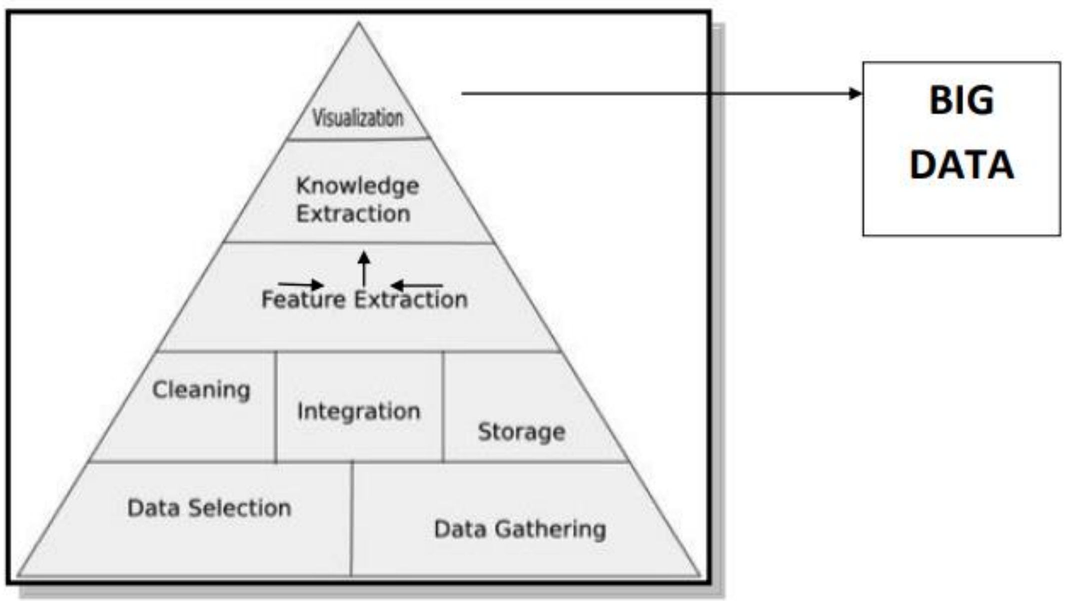


- A forth V has been added for *veracity* of extracted knowledge from data.

Big data

- Among the main Big data challenges, we can find:

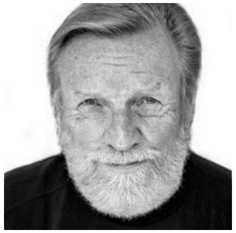
  ☞ Data acquisition & storage.
  ☞ Feature extraction.
  ☞ Data analysis for knowledge extraction.
  ☞ Visualization.



Knowledge extraction

- « We are drowning in information and starving in knowledge »

John Naisbitt,
(American author)



- The term "big data" tends to refer to the use of data analytics methods to extract value/knowledge from data in the form of: predictions, correlations, etc.

- Knowledge extraction from data can enable new strategies and practices to create efficient solutions in several domains:

Example: health, security, trade, manufacturing, etc.

Data analytics & machine leaning

- Machine learning (ML) is a growing field of artificial intelligence that uses statistical techniques to give computers the ability to learn from and make predictions on data.

- What does learning mean?

*A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P** if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.*

Tom M. Mitchell

Why study machine learning?

- ML tries basically to emulate human skills for learning from data, but can be deployed at a large scale.

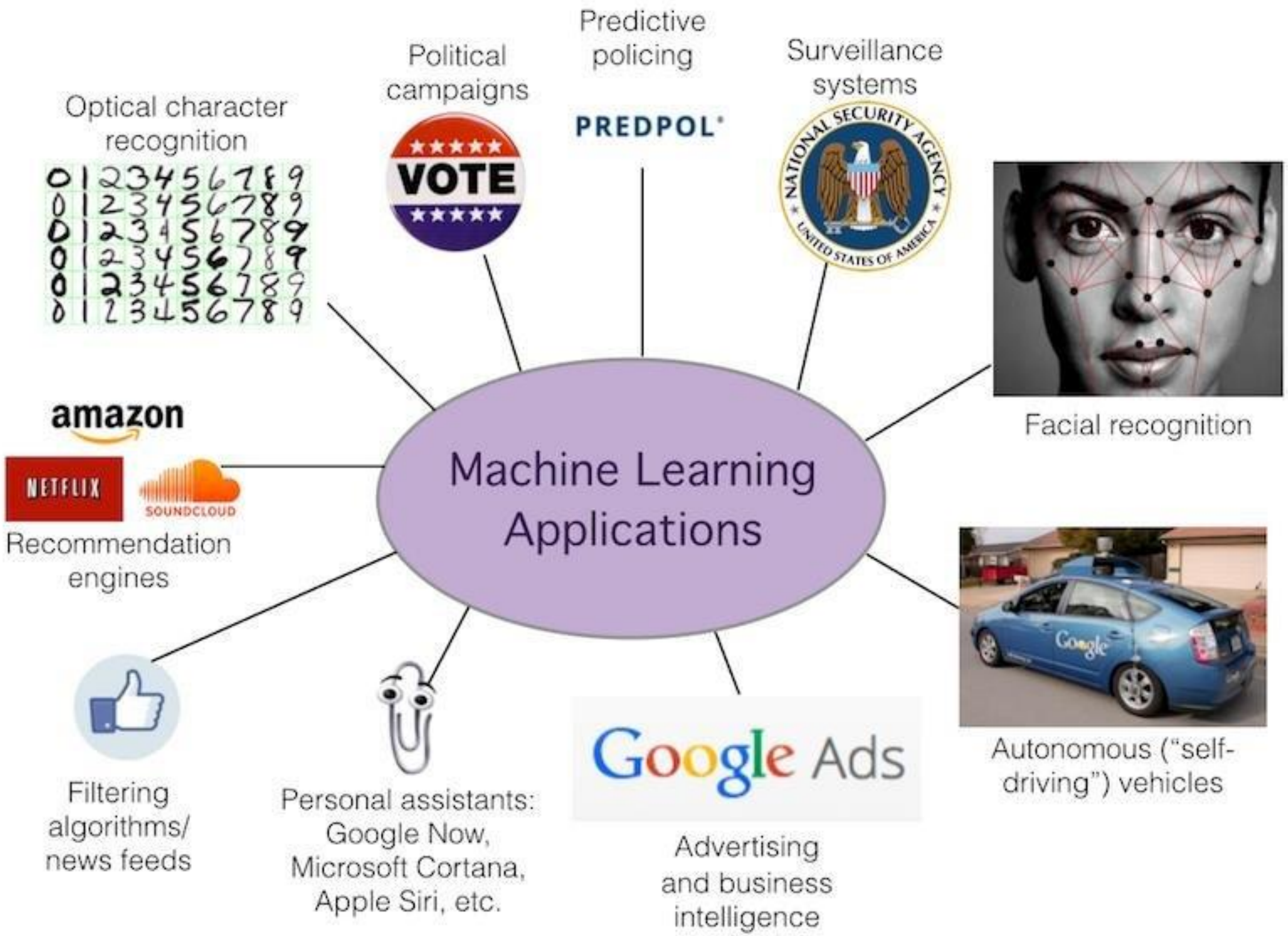
- Government agencies and industries are turning to ML:

  ☞ To create intelligent systems improving services (e.g., reduction of waiting times, personalized service, etc.)

  ☞ To analyse and predict outcomes from massive data (e.g. automatic decease diagnosis, risk management, etc.).

  ☞ Several technologies have been improved using ML.

Applications of machine learning



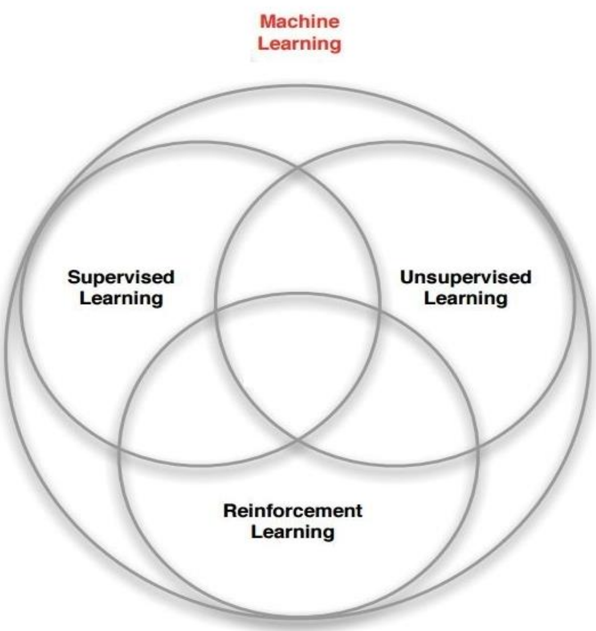
Applications of machine learning

- Machine learning is currently the preferred approach in the following domains:

    - **Speech analysis:** e.g., speech recognition, synthesis.
    - **Computer vision:** e.g., object recognition/detection.
    - **Robotics:** e.g., position/map estimation.
    - **Bioinformatics:** e.g., sequence alignment, genetic analysis.
    - **E-commerce:** e.g., automatic trading, fraud detection.
    - **Financial analysis:** e.g., portfolio allocation, credits.
    - **Medicine**: e.g., diagnosis, therapy conception.
    - **Web**: e.g., Content management, social networks, etc.

Learning paradigms

- There are three main learning paradigms in ML:
  - ☞ Supervised learning
  - ☞ Unsupervised learning
  - ☞ Reinforcement learning
- Possibility to combine these paradigms for ML models design.



Supervised learning (SL)

- In SL the computer is presented with training examples and their desired outputs.

$$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \cdots, (\mathbf{x}^{(n)}, y^{(n)})\}$$

- The goal is to learn a model/function that maps inputs to outputs for newly-observed data $f: \mathbf{x} \to y$:

  ☞ When y is real, we talk about regression.

  ☞ When y is discrete, we talk about classification.

Unsupervised learning (UL)

- The main goal of UL is discovering hidden patterns in data.

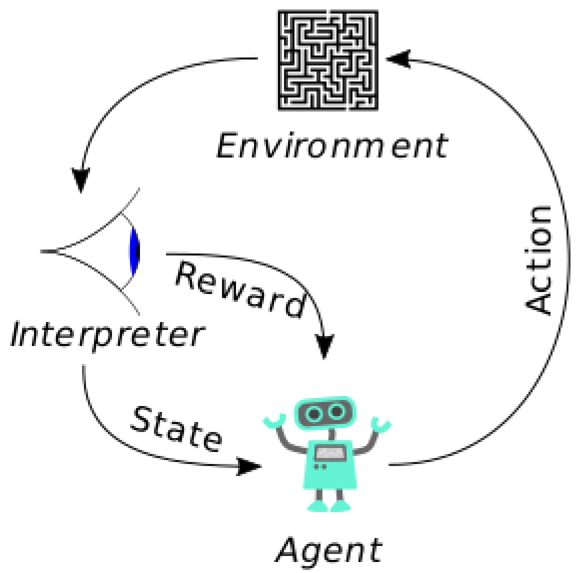
- No labels are given to the learning algorithm, leaving it on its own to explore or find structure in the data.

$$\mathcal{D} = \left\{x^{(1)}, x^{(2)}, \cdots, x^{(n)}\right\}$$

- UL problems include:
  - Clustering.
  - Density estimation.
  - Dimensionality reduction (e.g., ICA, PCA), etc.

Reinforcement learning (RL)

- The system learns by rewards and punishments given as a feedback to the program's actions in a dynamic environment;



Examples: Driving a vehicle, playing games, etc.

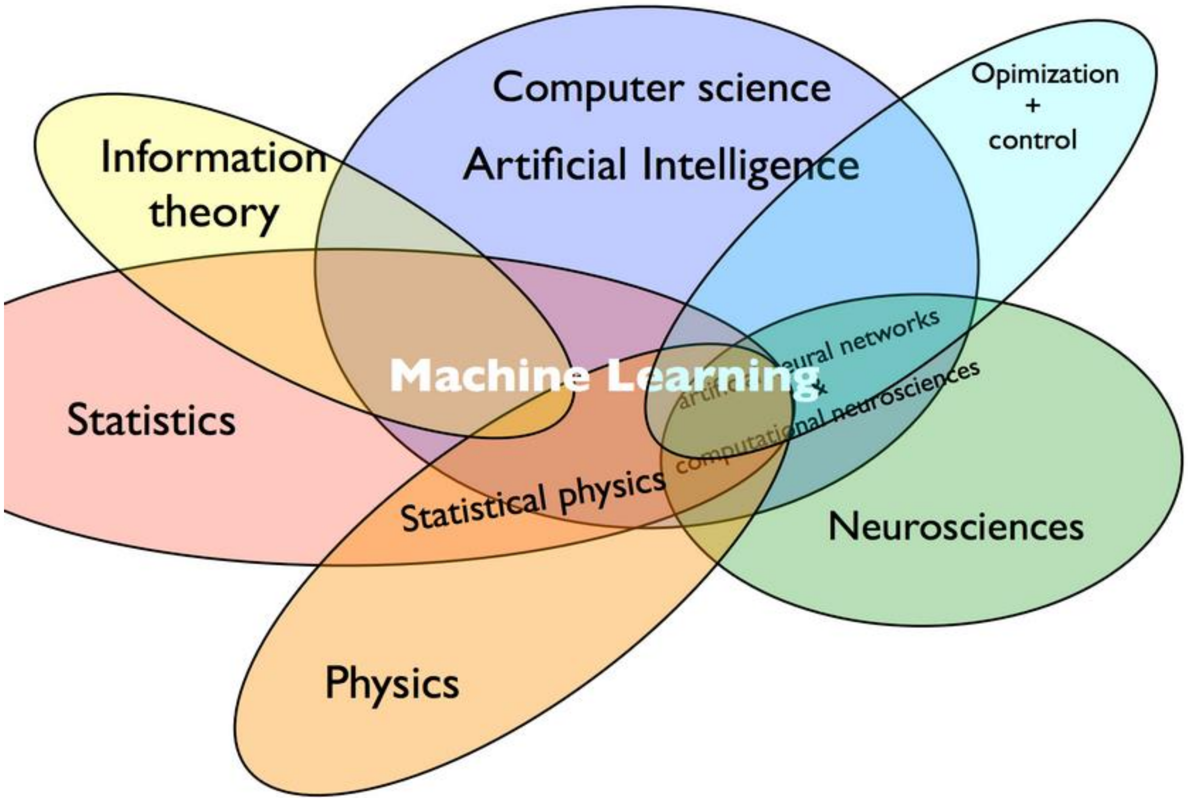
Learning paradigms

- There are several ML methods proposed in the literature

Supervised learning	Unsupervised learning	Reinforcement learning
Support vector machines, Decision trees, Naïve Bayes, Logistic regression, Neural networks, K-nearest neighbors, Bagging, Adaboost, etc.	*K-means, Mixture models, Hidden Markov models, Principal comp. analysis PCA Independent Comp. analysis ICA Non-negative matrix factorisation, Autoencoders, Deep Belief Nets, etc.*	*Temporal difference learning, Q-learning, Error-driven learning, Deep Q Networks, etc.*

Machine leaning field

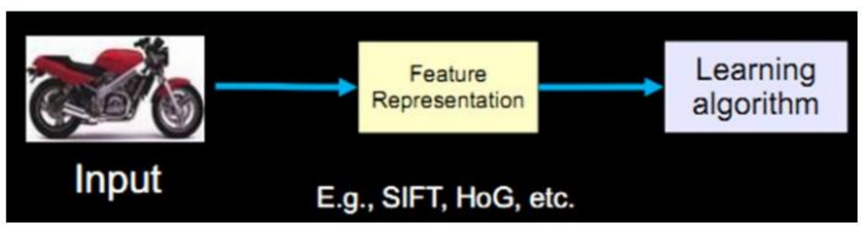
- ML algorithm design may involve several disciplines:



Conventional ML systems design

- Features are first extracted from raw data (e.g., text, image, sound, etc.) to provide a reduced representation.

- The features are then used for model learning.



- The produced features will be effective if they are non-redundant and informative about the problem at hand.

Deep learning (DL)

- DL is a subfield of ML research, introduced with the objective of moving ML closer to one of its original goals: AI.

- DL models learn automatically multiple levels of representations corresponding to different levels of abstraction; the levels form a hierarchy of concepts.

- DL is basically built on deep neural networks architectures. It uses a data-driven approach for their training.

- The success of DL is attributed mainly to the availability of huge amount of labeled data and computation power.

Focus of the tutorial

- Although DL can be applied to the tree types of learning, this tutorial will focus mainly on supervised learning.

- We will also introduce some methods for unsupervised learning, especially when autoencoders will be presented.

- This tutorial is divided into two main parts:

  - **Part I:** Supervised learning using neural networks.

  - **Part II:** Deep learning methods.

Part I:
Supervised learning using Neural networks

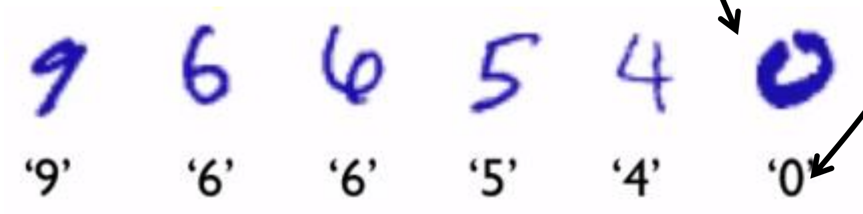
Notions about supervised learning

Supervised learning (SL) methods

- SL is about learning models (or functions) mapping input data $\mathbf{x}$ to defined targets y.

- A model is trained using a set of labeled data :

$$\mathcal{D}_{\text{train}} = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \left(\mathbf{x}^{(2)}, y^{(2)}\right), \cdots, \left(\mathbf{x}^{(n)}, y^{(n)}\right)\right\}$$



- We call $\mathbf{x}^{(i)}$ the entry and $y^{(i)}$ the target of the ith example.

- An element of $\mathcal{D}_{\text{train}}$ is called a learning example or an instance.

Supervised learning (SL) methods

- The input vector $\mathbf{x}^{(i)}$ can be of several dimensions.
- Let d be the number of these dimensions. Then, we have:

$$\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \dots, x_d^{(i)}\right)^T$$

- Each dimension of $\mathbf{x}^{(i)}$ expresses a feature (attribute) of the data to be predicted.
- In general, the features are of real type ($x_j^{(i)} \in \mathbb{R}, j = 1, \dots, d$).

Supervised learning (SL) methods

- Most often, the training data in $\mathcal{D}_{\text{train}}$ are arranged in a matrix $\mathbf{X}$, called a design matrix, and a target vector $\mathbf{y}$:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_j^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_j^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^{(i)} & x_2^{(i)} & x_3^{(i)} & \cdots & x_j^{(i)} & \cdots & x_d^{(i)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \cdots & x_j^{(n)} & \cdots & x_d^{(n)} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(i)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

ith data point $\mathbf{x}^{(i)}$

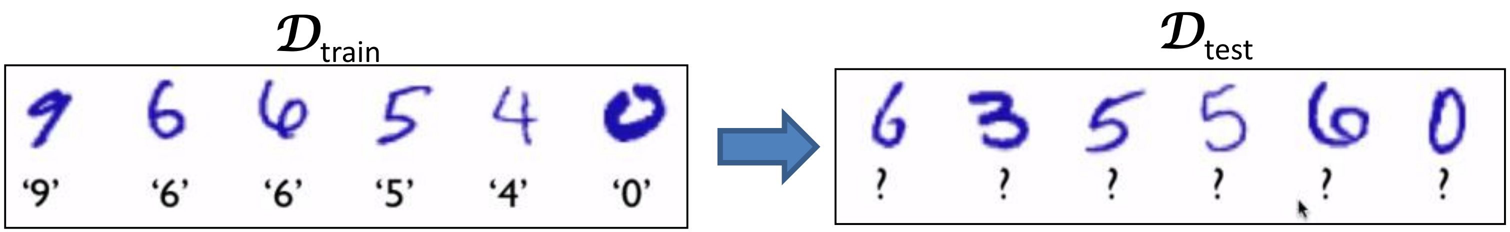
jth feature

Supervised learning (SL) methods

- A SL algorithm returns a function (model/hypothesis) able to give the correct answer and to generalize itself to new data:

- Let us denote by $f(\mathbf{x})$ the function generated by the learning algorithm using training data $\mathcal{D}_{\text{train}}$.

- We can measure the generalization performance of $f(\mathbf{x})$ on a separate data set $\mathcal{D}_{\text{test}}$.

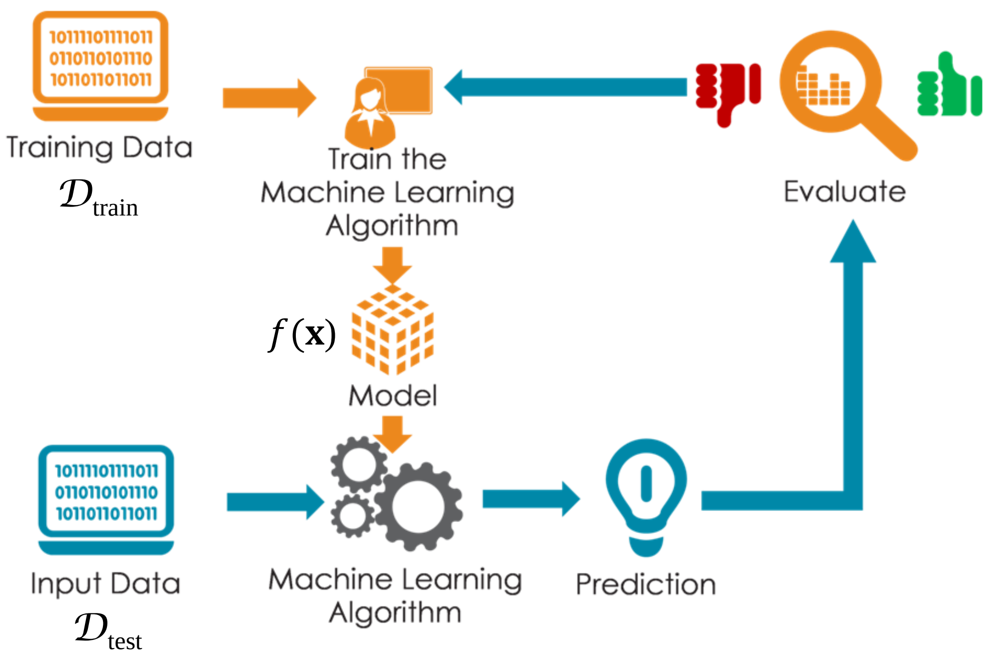


Generalization?

Supervised learning (SL) methods

- Having a training data set $\mathcal{D}_{\text{train}}$, find a function f such that f is a good predictor for the target value y:

- The general scheme for constructing a supervised machine learning model is described as follows:



Performance evaluation

- The validation error is measured differently for regression and classification problems.

☞ Regression error (E_r): is defined by the average of differences between real and predicted targets by the model:

$$\boxed{E_r = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{\mathbf{x}^{(i)} \in \mathcal{D}_{\text{test}}} \left(f\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

☞ Classification error (E_c): is defined by the % of misclassified data. Let I be the indicator function where $\mathrm{I}(a) = 1$ if a = true:

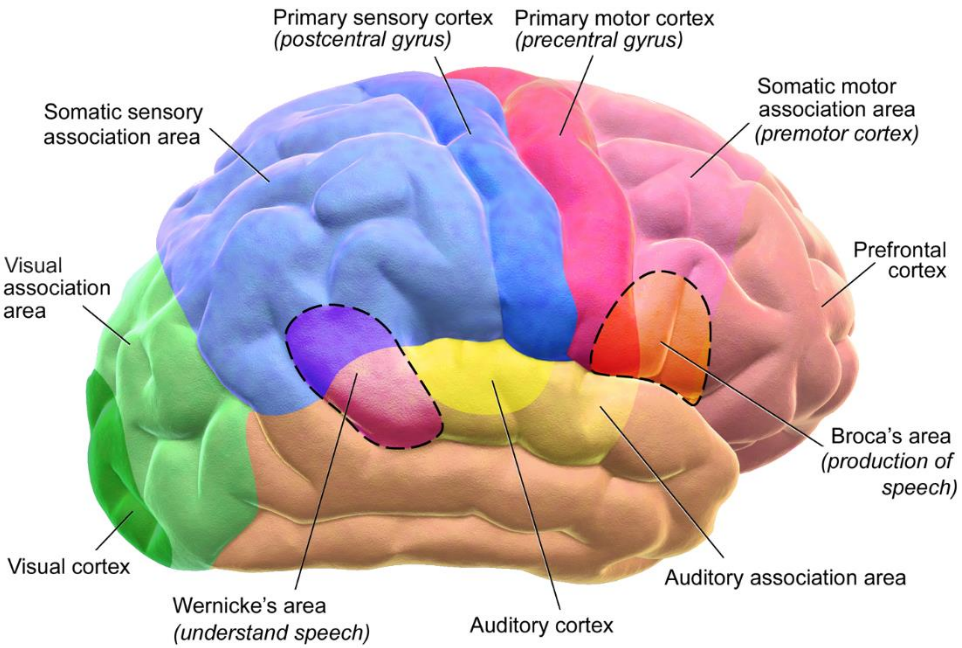
$$\boxed{E_c = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{\mathbf{x}^{(i)} \in \mathcal{D}_{\text{test}}} \mathrm{I}\left(f\left(\mathbf{x}^{(i)}\right) \neq y^{(i)}\right)}$$

Introduction to neural networks

Artificial intelligence

- For a long time, scientists have been interested to understand how the human brain works.

- The brain has huge capabilities to perform complex tasks (e.g., vision & speech recognition, language processing, etc.).

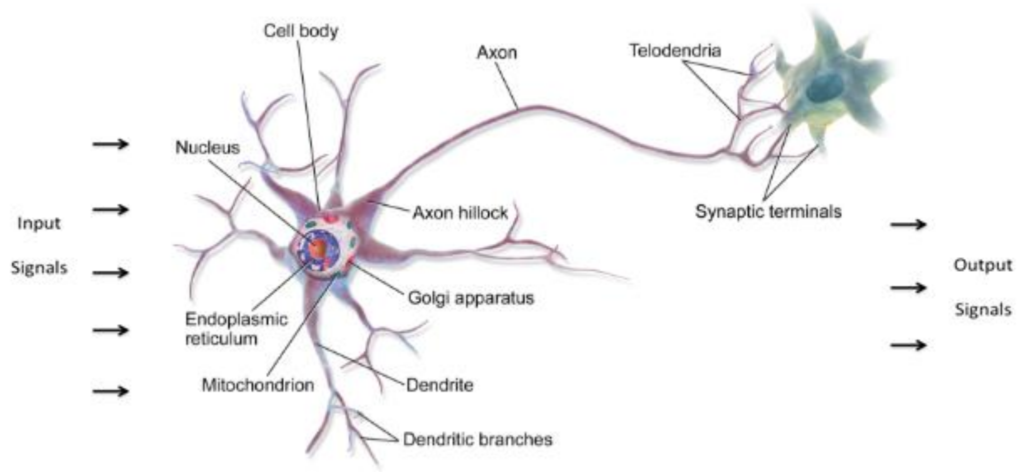


Computer versus Brain

Computer	Brain
• Generally has one single active processor. • The memory is separate from the processor and is passive.	• Has $\sim 10^{11}$ parallel processing units: the neurons. • Processing is done by neurons and memory is encoded in the synapses,
• Can organize computers in network configurations. • Not significant connectivity between computers.	• Units and memory are distributed over the brain network. • Significant connectivity between neurons in the brain.

Human brain network

- The brain is composed of neurons linked together by synapses.

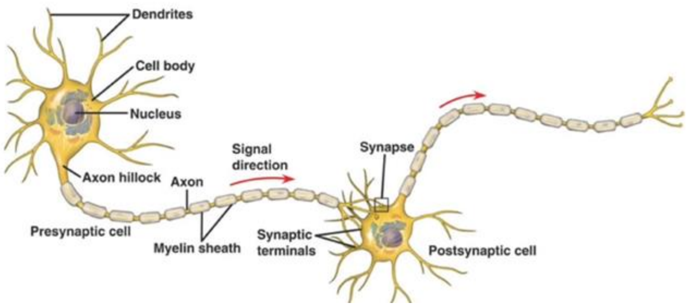


- Human brain has $\sim 10^{11}$ neurons $\approx$ number of trees in the amazon forest.

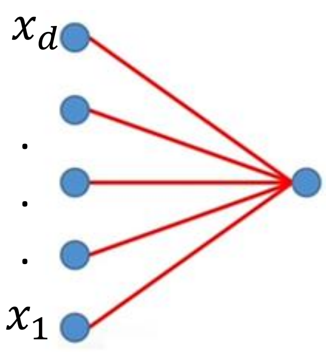
- The number of synapses $\approx$ number of tree leaves in the amazon forest.

From biological to artificial neurons

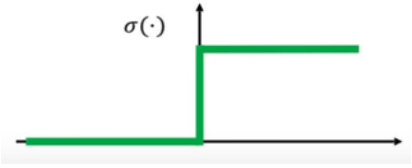
- Biological neurons are fired based on the intensity of the entering signals:



- Can be simulated by activation functions:

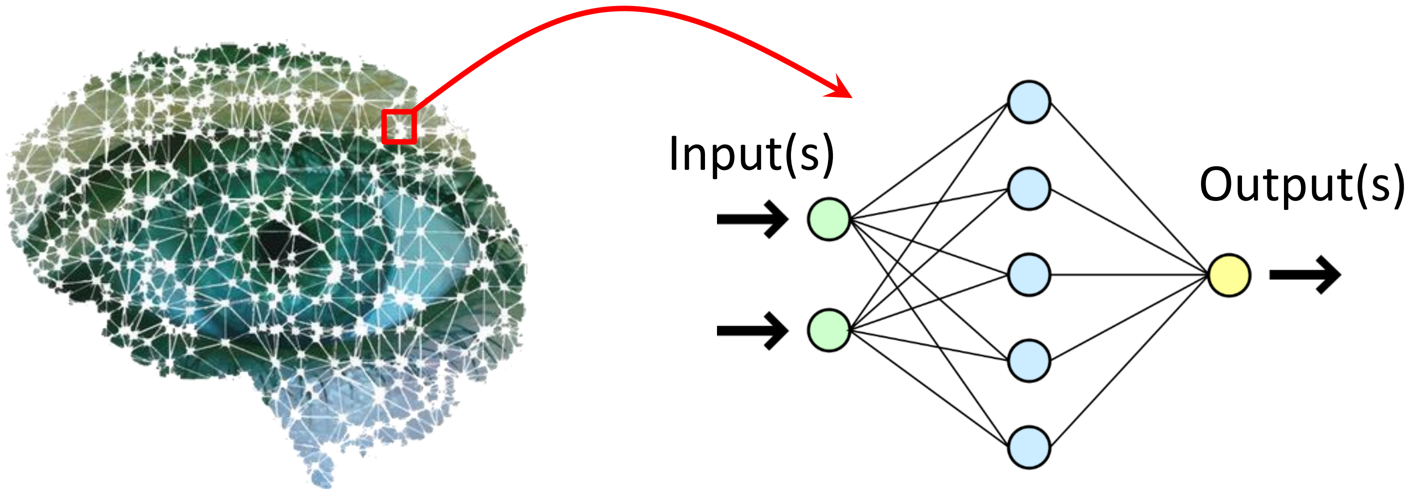


$$y = \sigma\left(\sum_i w_i x_i\right)$$



Artificial neural networks (ANN)

- Are a set of algorithms with schematic design inspired from the biological neurons composing the brain.



- Artificial neurons are arranged in layers and linked by weights in a similar way to synapses linking biological neurones.

Perceptron

Perceptron

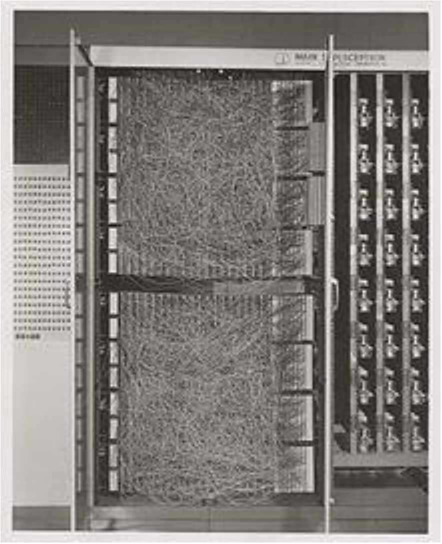
- Has been invented in 1957 par Frank Rosenblatt at the aero-spatial labs at Cornell university.

- Used analog hardware to ensure connection between neurons.

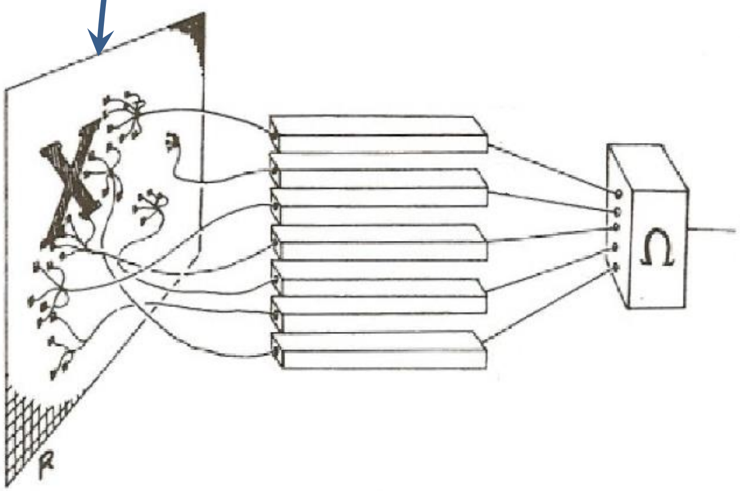
Perceptron

- The perceptron was first designed to learn simple shapes and characters on images.

Image de 20 x20 cellules



perceptron Mark 1



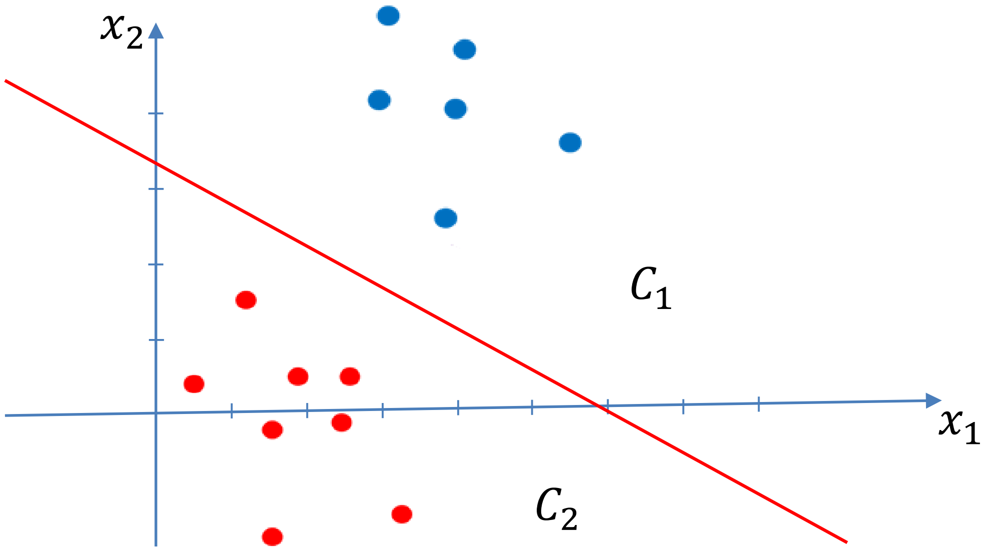
Perceptron

- More formally, let us have an input vector $\mathbf{x}$ with d entries:

$$\mathbf{x} = (x_1, x_2, \dots, x_d)^T,\ x_j \in \mathbb{R}, j = 1, \dots, d.$$

- Suppose we have 2-dimensional classification problem where data belong to one of two linearly separable classes $\{C_1,\ C_2\}$:

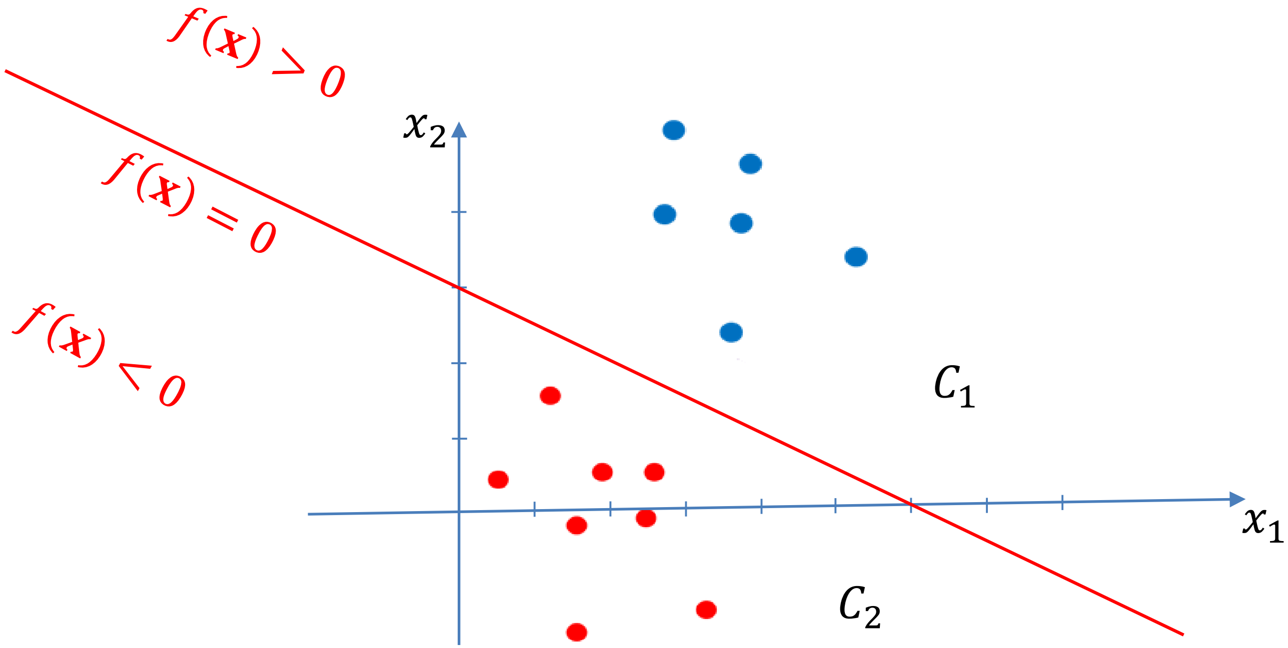
$d = 2$



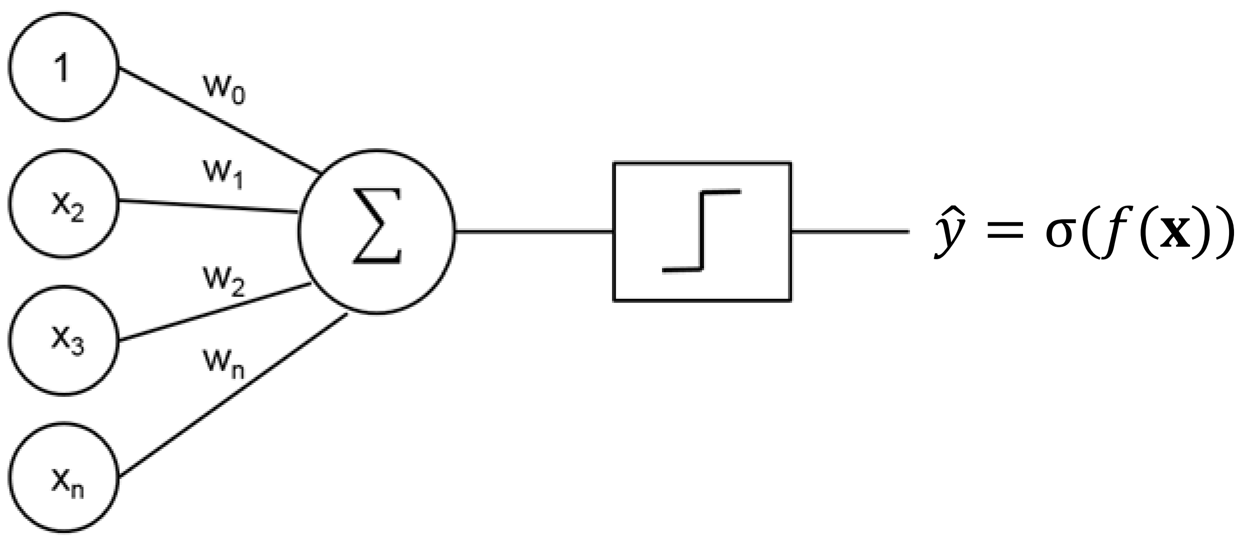
Perceptron

- The classification of points can be achieved by applying a threshold σ to a linear function $f(\mathbf{x})$:

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j$$



Perceptron



$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}\begin{cases} \geq 0, \text{then } \hat{y} = +1 \text{ and } \mathbf{x} \in C_1 \\ < 0, \text{then } \hat{y} = -1 \text{ and } \mathbf{x} \in C_2 \end{cases}$$

How to estimate the vector $\mathbf{w} = (w_0, w_1, \ldots, w_d)^T$ from data?

Training perceptrons

- Let us have n training data with their labels:

$$\mathcal{D} = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \left(\mathbf{x}^{(2)}, y^{(2)}\right), \cdots, (\mathbf{x}^{(n)}, y^{(n)})\right\}$$

- The perception can learn $\mathbf{w}$ iteratively using all data at once (batch) or one data point at a time (stochastic).

- Let us have an initial value for the weight vector $\mathbf{w}^{(0)}$.

- Let us have a learning factor $\alpha > 0$.

Algorithm for training a perceptron

Input :($\mathcal{D}$, $\mathbf{w}^{(\mathbf{0})}$).

Output :$\mathbf{w}$.

for each data point $\mathbf{x}^{(j)}, j = 1, \dots n$, **do**

  if $\left(\mathrm{y}^{(j)} = +1 \text{ and } f(\mathbf{x}^{(j)}) \leq 0\right)$ **then**

    $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \alpha\, \mathbf{x}^{(j)}$

  else **if** $\left(\mathrm{y}^{(j)} = -1 \text{ and } f(\mathbf{x}^{(j)}) \geq 0\right)$ **then**

    $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha\, \mathbf{x}^{(j)}$

  else

    $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}$.

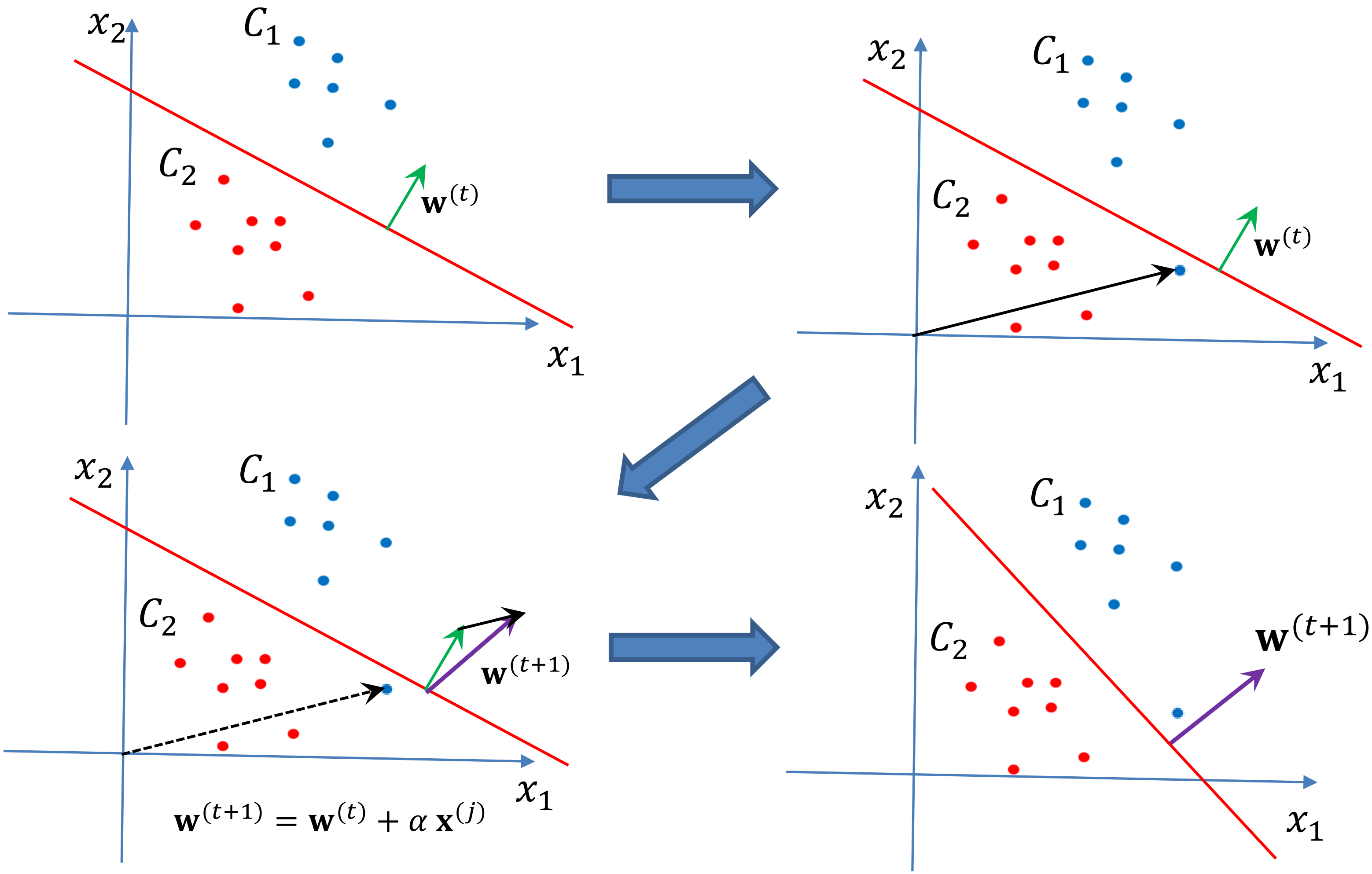
  end

  end

end

Algorithm for training a perceptron

Example:



From perceptrons to neural nets

- When classes are linearly separable, the training will always converge. Otherwise, it will iterate indefinitely.

- Use the perceptron as the basic element to develop neural networks (RNs) for recognition and prediction.

- It will have an input vector $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$, where $x_j \in \mathbb{R}$, which can come from the environment or other perceptrons.

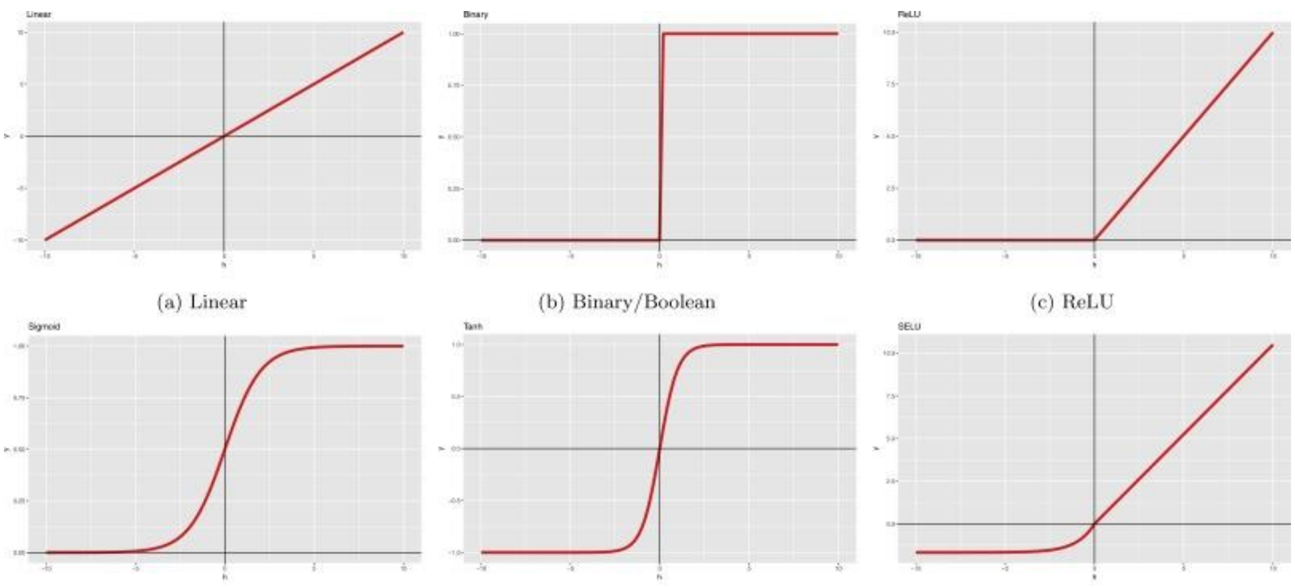
- For each entry x_j, associate a (synaptic) weight w_j. Add a weight w_0 with virtual input $x_0 = 1$.

From perceptrons to neural nets

- Before thresholding, the output f can vary from $-\infty$ à $+\infty$.

  ☞ Very small values of f can create instabilities.

- Instead of thresholds, approximate the value of $y \in \{0,1\}$ using soft activation functions.



(a) Linear (b) Binary/Boolean (c) ReLU

(d) Logistic (e) Tanh (f) SELU

Activation functions: number of classes K=2

- The sigmoid function has been widely used in the literature. It is given as follows:

$$h(\mathbf{x}) = \frac{1}{1 + \exp(-f(\mathbf{x}))}$$

$$= \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}$$

- Having two classes $\{C_1, \ C_2\}$, $h(\mathbf{x})$ is an approximation of the posterior probability of the class C_1.

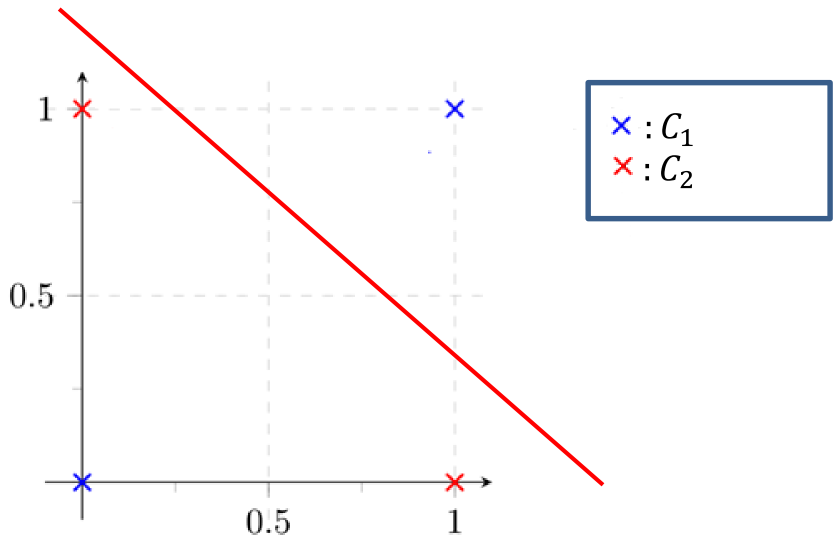
- The ReLU function has been more successful than the sigmoid in deep ANN architectures.

Non-linearly separable classes: K=2

- A perceptron can only separate linearly separable classes, but it is unable to separate non-linear class boundaries.

Example: Let the following problem of binary classification (problem of the XOR).

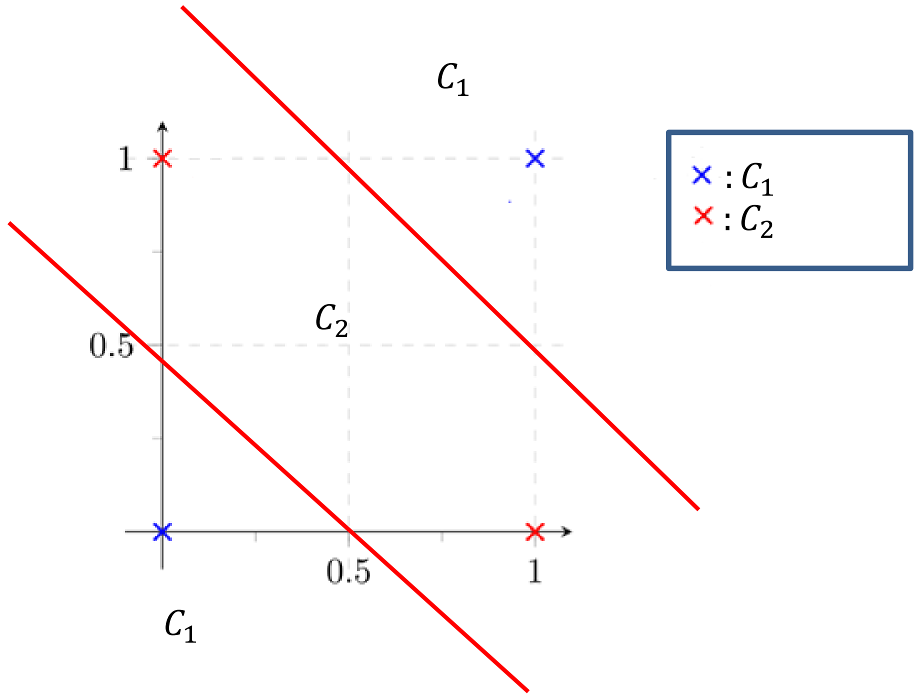
☞ Clearly, no line can separate the two classes!



Non-linearly separable classes: K=2

Solution :

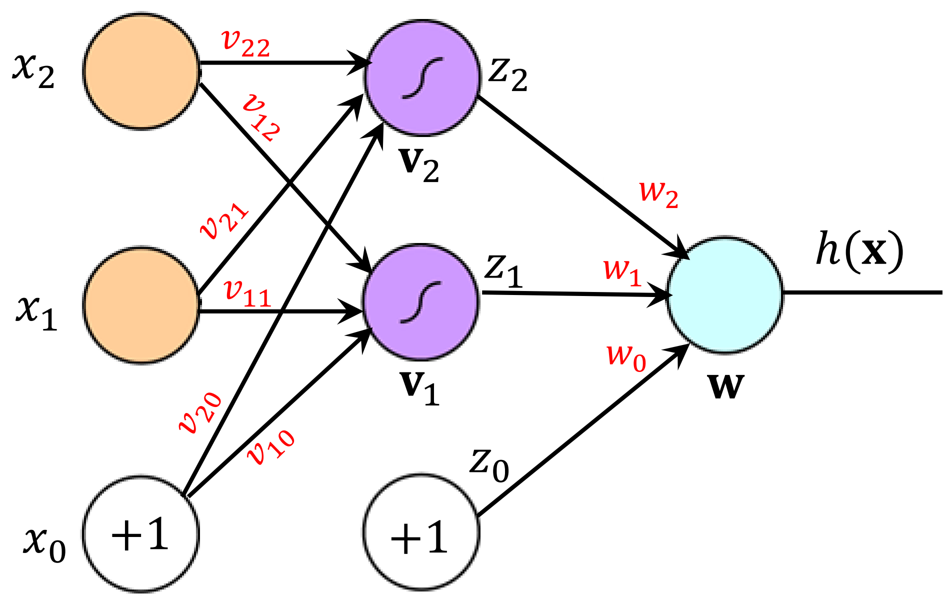
☞ Use two lines instead of one!



☞ Use an intermediary layer of neurons in the NN.

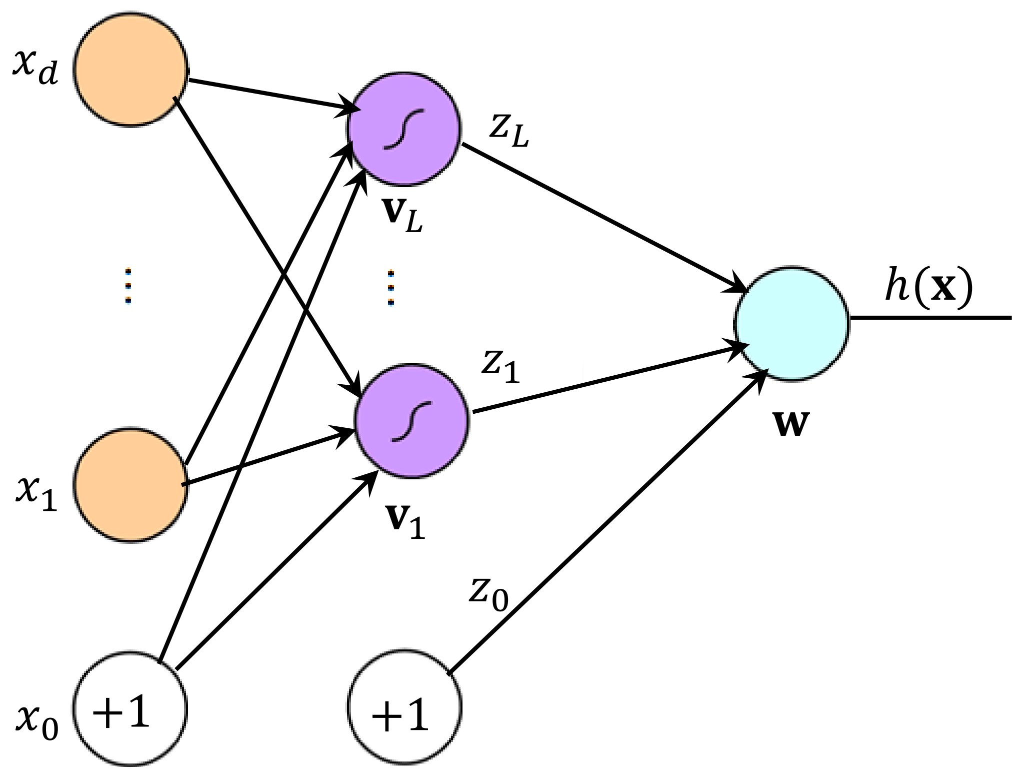
Multi-layer networks

- The intermediary (hidden) layer will create two intermediary decisions z_1 and z_2, which are combined into a final decision function $h(\mathbf{z})$ with weights w_1 et w_2.



Multi-layer networks

- One can add as many nodes as desired in the hidden layer (i.e., extra lines in the decision boundary).
- Associate a weight vector $\mathbf{v}_l$ for each hidden node $l \in \{1, \ldots, L\}$.



Multi-layer networks

- Each hidden node $l \in \{1, \dots, L\}$ will have an output z_l activated by a sigmoid function:

$$z_l = \frac{1}{1 + \exp(-\mathbf{v}_l{}^T \mathbf{x})}$$

- The final output of the NN h will combine all neuron outputs of the hidden layer $\mathbf{z} = (z_0, z_1, \dots, z_L)^T$, weighted by the weight vector $\mathbf{w} = (w_0, w_1, \dots, w_L)^T$:

$$h(\mathbf{z}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{z})}$$

Multi-class classification with ANNs

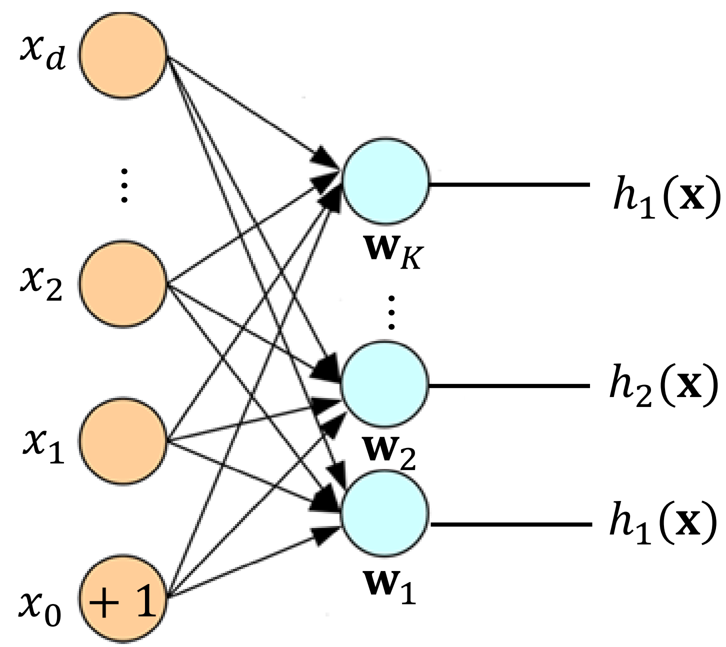
Shallow networks: number of classes K>2

- When the number of classes $K > 2$, we define K outputs (perceptrons), each one with a vector:

$$\mathbf{w}_k = (w_{k0}, w_{k1}, \dots, w_{kd})^T, k \in \{1, \dots, K\}:$$

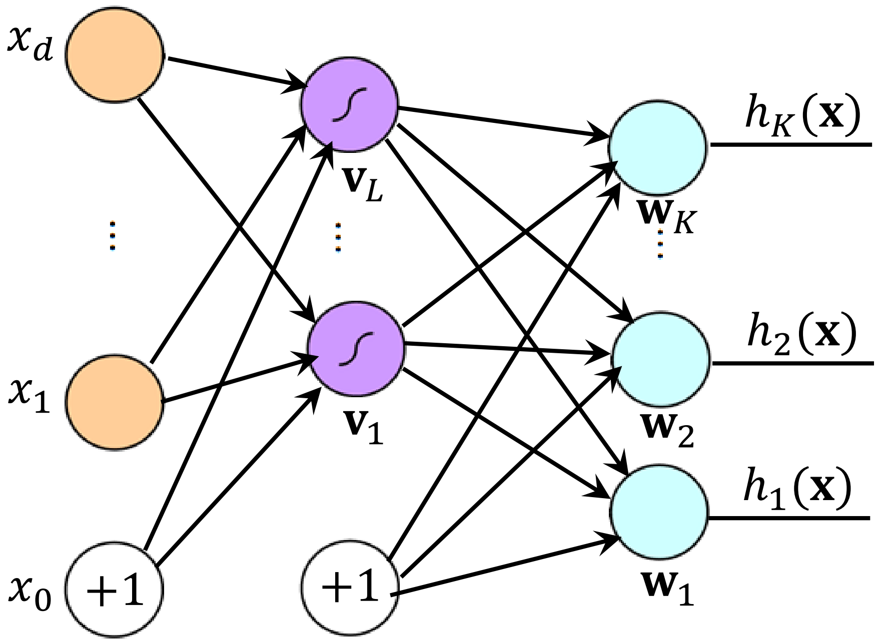
- The posterior probability of the class C_k is defined by the softmax function :

$$h_k(\mathbf{x}) = \frac{\exp(-\mathbf{w}_k{}^T\mathbf{x})}{\sum_l \exp(-\mathbf{w}_l{}^T\mathbf{x})}$$



Deep networks: number of classes K>2

- We can have non-linear class boundaries by adding an intermediary (hidden) layer:



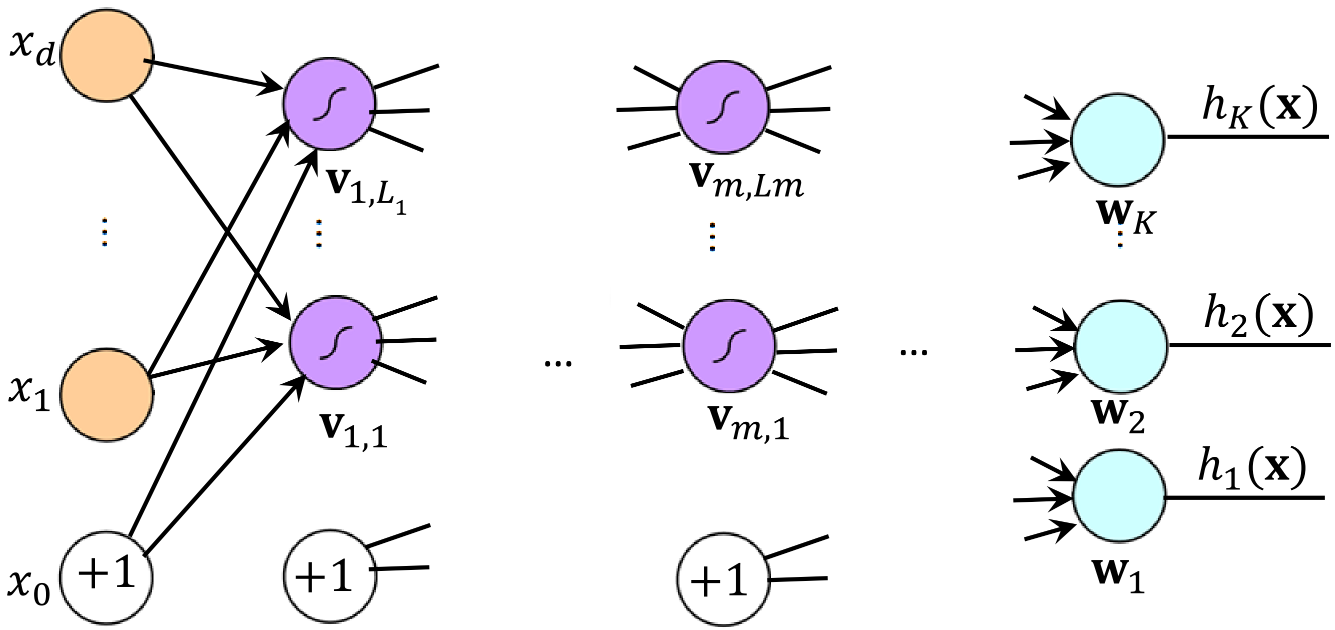
- The probability of class C_k is given by the softmax function:

$$h_k(\mathbf{x}) = \frac{\exp(-\mathbf{w}_k{}^T\mathbf{z})}{\sum_l \exp(-\mathbf{w}_l{}^T\mathbf{z})}$$

Deep networks: number of classes K>2

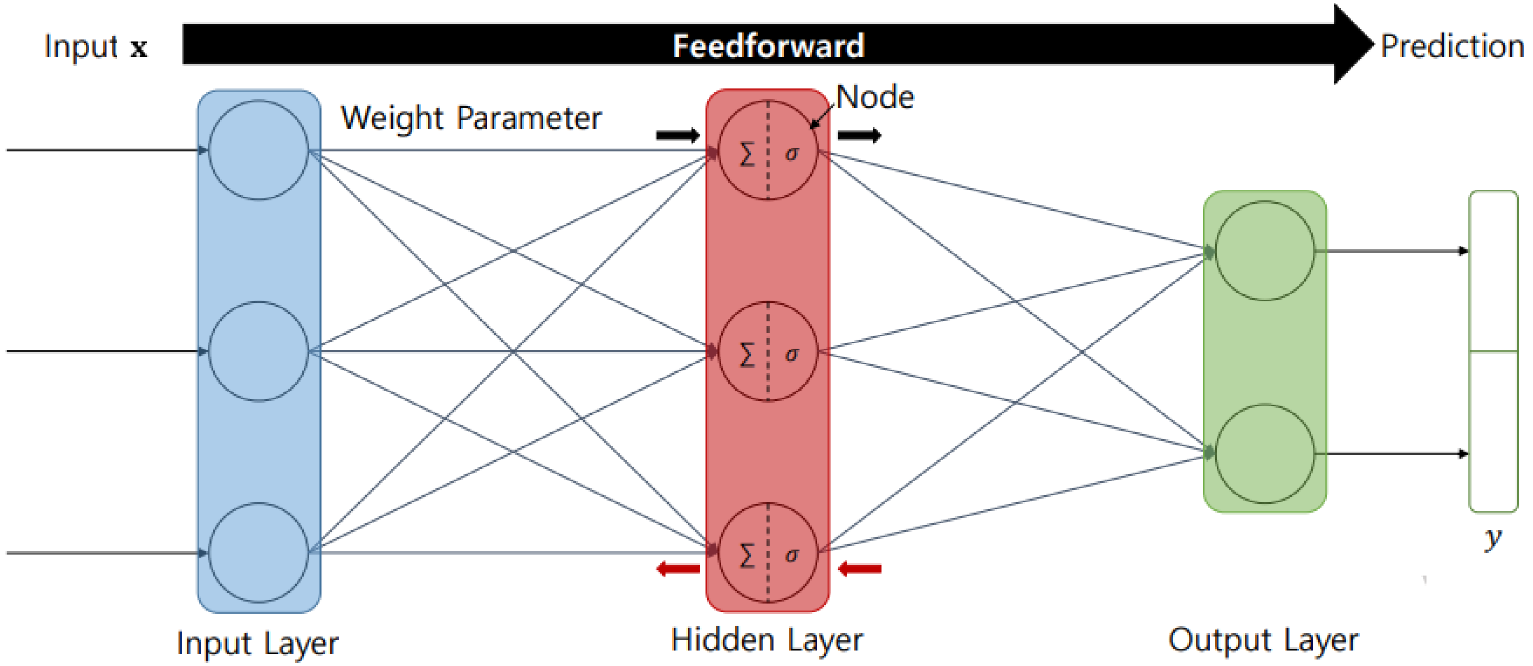
- Deep neural networks (number of hidden layers $M \geq 1$) can approximate very complex functions:

- Each hidden layer $m \in \{1, \ldots, M\}$ will have L_m nodes, connected by weight vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_{L_m}\}$ to the previous layer.



Feedforward ANN

- As ANNs propagate information in one direction, it is commonly referred to as a feed forward networks.



Parameter estimation for neural networks

Parameter estimation in ANN

- Consists of automatically determining node weights from data.
- Parameter estimation is formulated as an optimization problem seeking the minimum (or maximum) of an objective function.
- This objective function is built on training data by minimizing the error (or maximizing the precision) of their class prediction.
- Since the objective function in often nonlinear, the optimization is done by the algorithms of gradient descent.

ANN parameter estimation : case K=2

- Let us have a set $\mathcal{D}_{train}$ of n training data:

$$\mathcal{D}_{train} = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \left(\mathbf{x}^{(2)}, y^{(2)}\right), \cdots, (\mathbf{x}^{(n)}, y^{(n)})\right\}$$

- For binary classification, we have two classes $\{C_1,\ C_2\}$. We use the following encoding for the target variable y:

$$y = \begin{cases} 1 \quad si \;\; \mathbf{x} \in C_1 \\ \\ 0 \;\; si \;\; \mathbf{x} \in C_2 \end{cases}$$

- The optimal network weights are obtained by minimizing the classification error for predicting classes in $\mathcal{D}_{train}$.

ANN parameter estimation : case K=2

- Having a network output $h(\mathbf{x}^{(i)}, \mathbf{w})$ for the ith training example $(\mathbf{x}^{(i)}, y^{(i)})$, the classification precision can be measured by the logarithm of the likelihood function:

$$\boxed{\begin{aligned}\ell^{(i)}(\mathbf{w}) &= \log\left(\left[h(\mathbf{x}^{(i)}, \mathbf{w})\right]^{y^{(i)}} \times \left[1 - h(\mathbf{x}^{(i)}, \mathbf{w})\right]^{1-y^{(i)}}\right) \\ &= y^{(i)} \log\left(h(\mathbf{x}^{(i)}, \mathbf{w})\right) + (1 - y^{(i)}) \log\left(1 - h(\mathbf{x}^{(i)}, \mathbf{w})\right)\end{aligned}}$$

- The negative of this function $-\ell(\mathbf{w})$, called cross entropy, measures the classification error :

$$\boxed{E^{(i)}(\mathbf{w}) = -y^{(i)} \log\left(h(\mathbf{x}^{(i)}, \mathbf{w})\right) - (1 - y^{(i)}) \log\left(1 - h(\mathbf{x}^{(i)}, \mathbf{w})\right)}$$

ANN parameter estimation : case K=2

- Given the output $h\left(\mathbf{x}^{(i)}, \mathbf{w}\right)$, we can have two possibilities:

  ☞Good classification:

$$y^{(i)} = 1 \text{ et } h\left(\mathbf{x}^{(i)}, \mathbf{w}\right) = 1 \;\rightarrow E^{(i)}(\mathbf{w}) = 0.$$
$$y^{(i)} = 0 \text{ et } h\left(\mathbf{x}^{(i)}, \mathbf{w}\right) = 0 \;\rightarrow E^{(i)}(\mathbf{w}) = 0.$$

  ☞ Bad classification:

$$y^{(i)} = 1 \text{ et } h\left(\mathbf{x}^{(i)}, \mathbf{w}\right) = 0 \;\rightarrow E^{(i)}(\mathbf{w}) \gg 0.$$
$$y^{(i)} = 0 \text{ et } h\left(\mathbf{x}^{(i)}, \mathbf{w}\right) = 1 \;\rightarrow E^{(i)}(\mathbf{w}) \gg 0.$$

- Update $\mathbf{w}$ to minimize the error function $E^{(i)}(\mathbf{w})$.

ANN parameter estimation : case K>2

- When $K > 2$, the target variable of each example $\mathbf{x}^{(i)}$ will be encoded by a target vector $y^{(i)}$ defined as follows:

$$y^{(i)} = \left(y_1{}^{(i)}, y_2{}^{(i)}, \dots, y_K{}^{(i)}\right), \text{ with } \begin{cases} y_k{}^{(i)} = 1 \text{ si } \mathbf{x}^{(i)} \in C_k \\ y_k{}^{(i)} = 0 \text{ si } \mathbf{x}^{(i)} \notin C_k \end{cases}.$$

- We have a set of K vectors to estimate $\mathbf{W} = \{\mathbf{w}_1, \dots, \mathbf{w}_K\}$. Classification error for the point $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ is given by :

$$\boxed{E^{(i)}(\mathbf{W}) = -\sum_{k=1}^{K} y_k{}^{(i)} \log\left(h_k\left(\mathbf{x}^{(i)}, \mathbf{W}\right)\right)}$$

ANN parameter estimation : case K>2

- The ANN weights can be calculated in two ways:

  ☞ Offline (batch): Use all data in $\mathcal{D}_{train}$ at once.

  ☞ Online (stochastic): Use one data point at time.

- Using gradient descent, we iteratively adjust the weight vectors to minimize the error $E_k^{(i)}$ for each data point :

$$\boxed{\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha\, \Delta E_k^{(i)}}$$

- The constant $\alpha \in \mathbb{R}$ is the training factor.

ANN parameter estimation : case K>2

- The symbol $\Delta E_k^{(i)}$ designates the gradient vector of the error function $E^{(i)}$ with respect to the entries of $\mathbf{w}_k$.

- It can be easily demonstrated that the value of each entry of $\Delta E_k^{(i)}$ is given by:

$$\boxed{\Delta w_{kj} = \left({y_k}^{(i)} - h_k\left(\mathbf{x}^{(i)}, \mathbf{W}\right)\right) \ {x_j}^{(i)},} \quad j = 0, \ldots, d.$$

$$\boxed{\textbf{update} = (\text{real target} - \text{predicted target}) \times \text{data}}$$

Parameter estimation algorithm

- The final algorithm for estimating the parameters of one-layer ANN for a multi-class setting is given by:

$w_{kj} = \leftarrow rand(-0.01, +0.01);\;\; k = 1, \dots, K; j = 1, \dots, d.$

Repeat

  For each data point $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ and class C_k

    Calculate $h_k\left(\mathbf{x}^{(i)}, \mathbf{W}\right)$;

    For each dimension $j = 1, \dots, d$:

$$w_{kj}(t+1) = w_{kj}(t) + \alpha\left(y_k{}^{(i)} - h_k\left(\mathbf{x}^{(i)}, \mathbf{W}\right)\right) x_j{}^{(i)}$$

    end

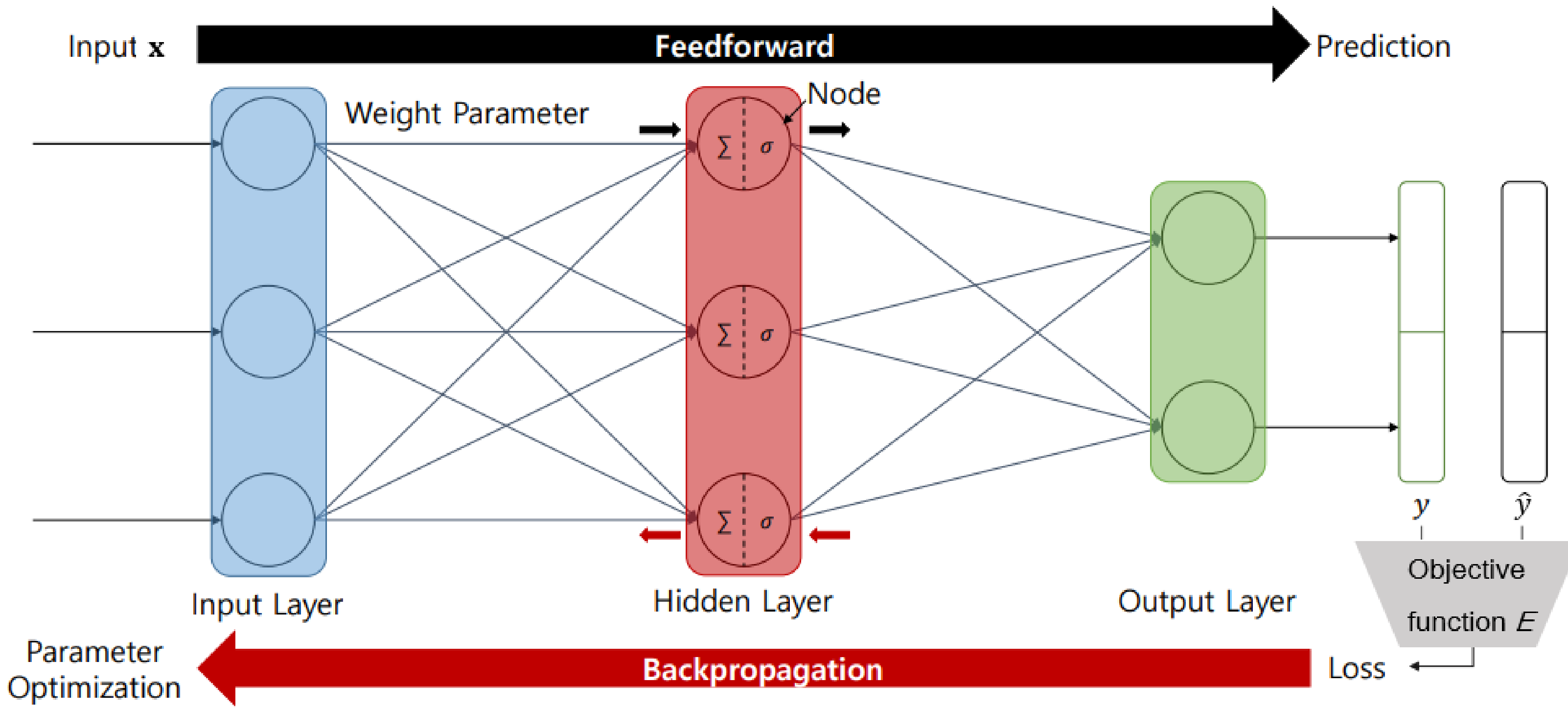
  End

Until convergence

Parameter estimation for multi-layer ANNs

Backpropagation method

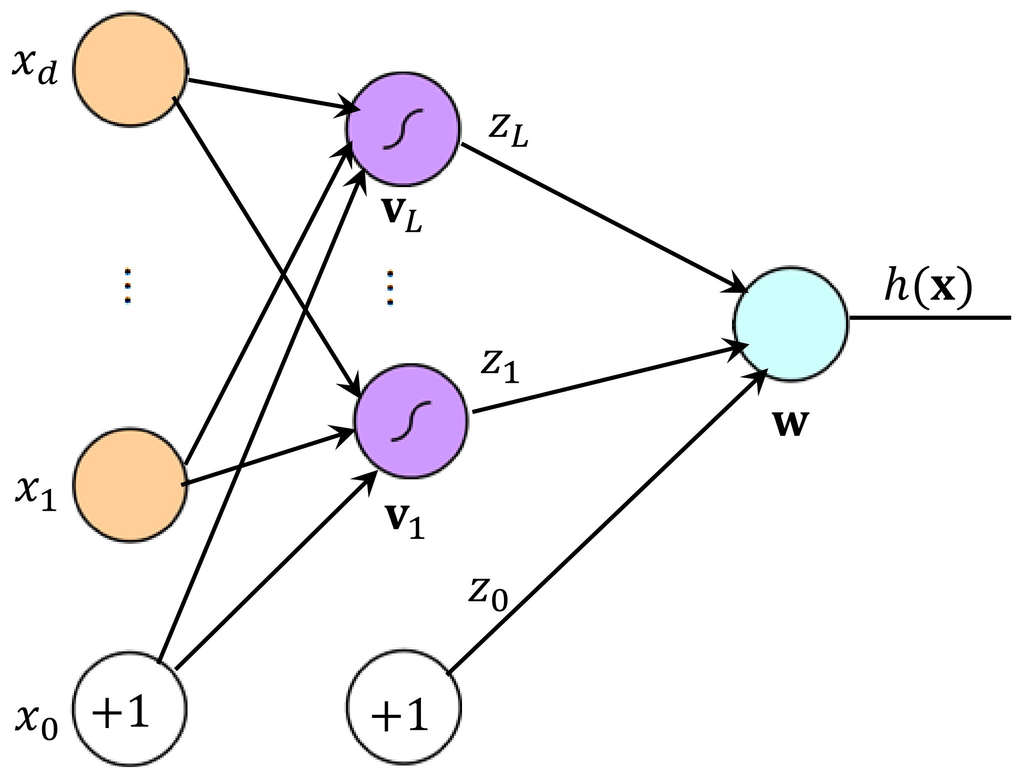
- The ANN weights are estimated by the backpropagation method, using the chain rule for calculating error derivatives :



ANN with one hidden layer: case K=2

- Suppose that we have L neurons in the hidden layer.

- We have to estimate L intermediate weight vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_L\}$ and a vector $\mathbf{w}$ for the final output.



Weight update using backpropagation: case K=2

- The derivative of he error function $E^{(i)}$ with respect to the entries of a vector $\mathbf{v}_l$ are given by:

$$\frac{\partial E^{(i)}}{\partial v_{lj}} = \frac{\partial E^{(i)}}{\partial h} \times \frac{\partial h^{(i)}}{\partial z_l} \times \frac{\partial {z_l}^{(i)}}{\partial v_{lj}}; j = 1, \dots, d; l = 1, \dots, L.$$

- The update of v_{lj} is given by:

$$\Delta v_{lj} = \frac{\partial E^{(i)}}{\partial v_{lj}}$$

$$= \underbrace{\frac{\partial E^{(i)}}{\partial h}} \times \underbrace{\frac{\partial h^{(i)}}{\partial z_l}} \times \underbrace{\frac{\partial {z_l}^{(i)}}{\partial v_{lj}}}$$

$$= \left(y^{(i)} - h^{(i)}\right) \times w_l \times {z_l}^{(i)}(1 - {z_l}^{(i)}){x_j}^{(i)}$$

Weight update using backpropagation: case K=2

- To update of the weights in the final and hidden layers can be performed online by alternating the following equations:

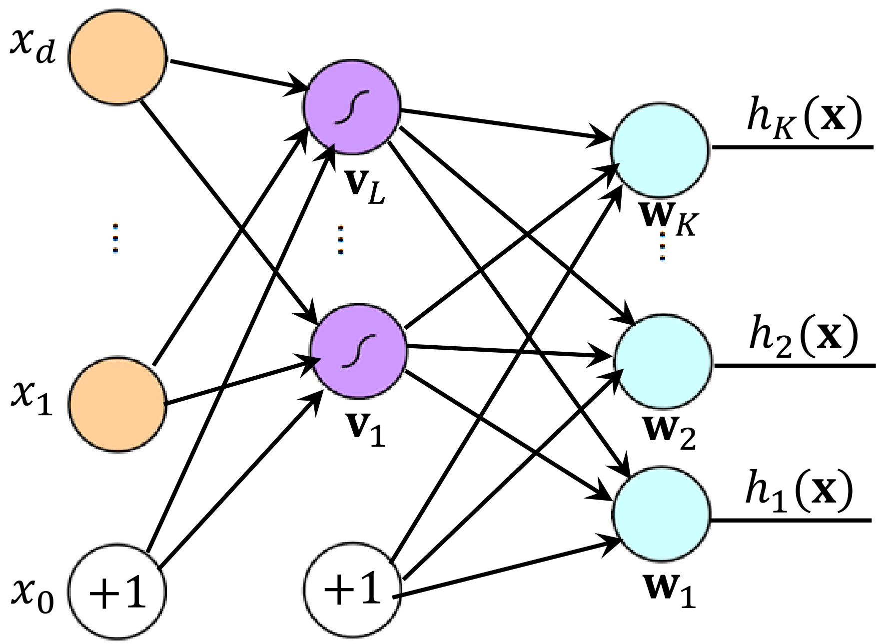
$$\boxed{\begin{aligned} \Delta w_l \; &= \left(y^{(i)} - h^{(i)}\right) {z_l}^{(i)} \\ \Delta v_{lj} &= \left(y^{(i)} - h^{(i)}\right) w_l \times {z_l}^{(i)} \left(1 - {z_l}^{(i)}\right) {x_j}^{(i)} \end{aligned}}$$

- The product $\left(y^{(i)} - h^{(i)}\right) w_l$ back-propagates the error to the l-th hidden node $\mathbf{z}_l$.

- The term ${z_l}^{(i)}\left(1 - {z_l}^{(i)}\right){x_j}^{(i)}$ is the derivative of the sigmoid output ${z_l}^{(i)}$ with respect to v_{lj}.

Backpropagation for K>2

- We have to estimate L vector $\{\mathbf{v}_1, \dots, \mathbf{v}_L\}$ for the hidden layer and K vectors $\{\mathbf{w}_1, \dots, \mathbf{w}_K\}$ for the final layer:



Backpropagation for K>2

- Recall that the encoding of the target variable $y^{(i)}$ for the ith example is the following:

$$y^{(i)} = \left({y_1}^{(i)}, {y_2}^{(i)}, \dots, {y_K}^{(i)}\right), \text{ where: } \begin{cases} {y_k}^{(i)} = 1 \text{ if } \mathbf{x}^{(i)} \in C_k \\ {y_k}^{(i)} = 0 \text{ if } \mathbf{x}^{(i)} \notin C_k \end{cases}.$$

- The update equations are as follows:

$$\boxed{\begin{aligned} \Delta w_{kl} &= \left({y_k}^{(i)} - {h_k}^{(i)}\right){z_l}^{(i)} \\ \Delta v_{lj} &= \sum_{k=1}^{K} \left({y_k}^{(i)} - {h_k}^{(i)}\right) w_{kl} \times {z_l}^{(i)}\left(1 - {z_l}^{(i)}\right){x_j}^{(i)} \end{aligned}}$$

Estimation algorithm

$w_{kl} \leftarrow rand(-0.01, +0.01);$
$v_{lj} \leftarrow rand(-0.01, +0.01);$

Repeat

For each training example $\left(\mathbf{x}^{(i)}, y^{(i)}\right), i \in \{1, \dots, n\};$

Calculate $z_l\left(\mathbf{x}^{(i)}\right)$, for $l \in \{1, \dots, L\};$

Calculate $h_k\left(\mathbf{x}^{(i)}\right)$, for $k \in \{1, \dots, K\};$

Calculate Δw_{kl}, for $k \in \{1, \dots, K\}, l \in \{1, \dots, L\};$

$$w_{kl}(t+1) = w_{kl}(t) + \alpha\, \Delta w_{kl};$$

Calculate Δv_{lj}, for $l \in \{1, \dots, L\}, j \in \{1, \dots, d\};$

$$v_{lj}(t+1) = v_{lj}(t) + \alpha \Delta v_{lj};$$

End

Until Convergence

Some remarks

- The iterations required to estimate the parameters of ANNs are called epochs.

  1 epoch = a complete pass over the data.

- Adding multiple hidden layers can approximate complex functions. However, the ANN will become:

  ☞ More complex to train.

  ☞ Less interpretable.

  ☞ Slow to reach convergence.

Enhancing convergence?

- For better convergence:

  ☞ For each epoch, shuffle randomly the data.

  ☞ Increase / decrease the learning factor according to the rate of decay of the error function.

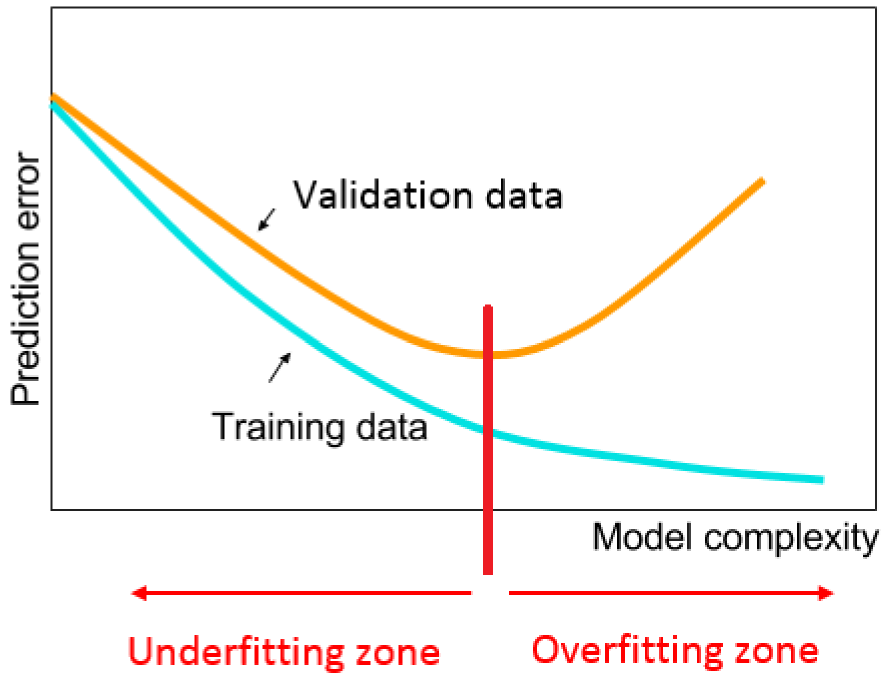
  ☞ Initialize weights to values close to 0.

  ☞ Use other weight initialization techniques (e.g., autoencoders).

Reduce overfitting in ANN

- Overfitting can occur when complex networks are used for resolving simple problems.

- To mitigate the overfitting, use a separate validation data to stop network training.

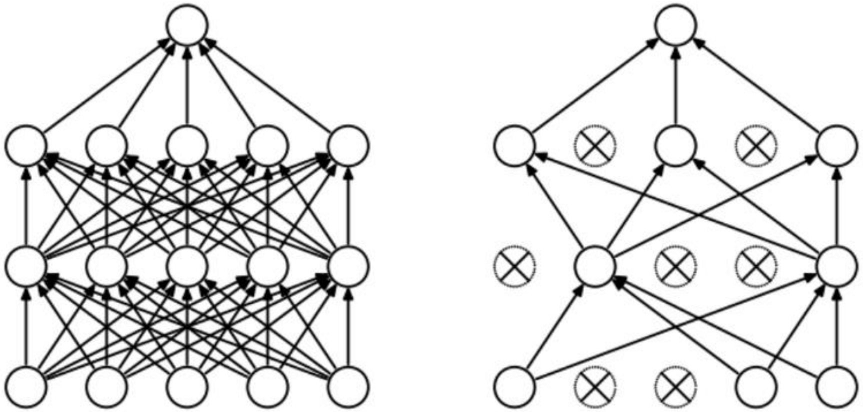


Reduce overfitting in ANN

- Another technique to reduce overfitting is using dropout :

Training phase: For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, *p*, of nodes (and corresponding activations).

Testing phase: Use all activations, but reduce them by a factor *p* (to account for the missing activations during training).

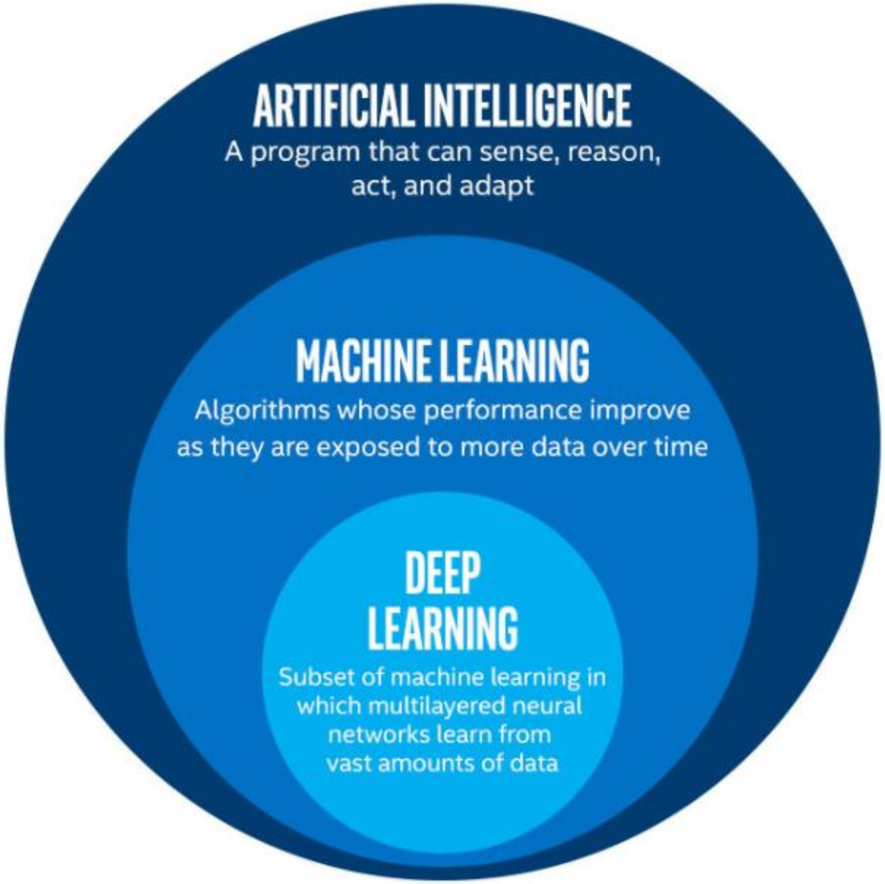


(a) Standard Neural Net (b) After applying dropout.

Part II:
Deep learning

Deep learning (DL)

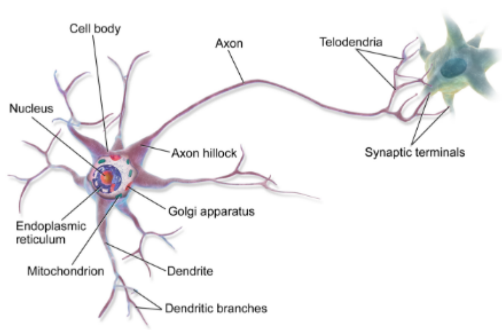
- DL is a subfield of ML, developed by several researchers : G. Hinton, Y. Bengio, Y. Lecun, etc.



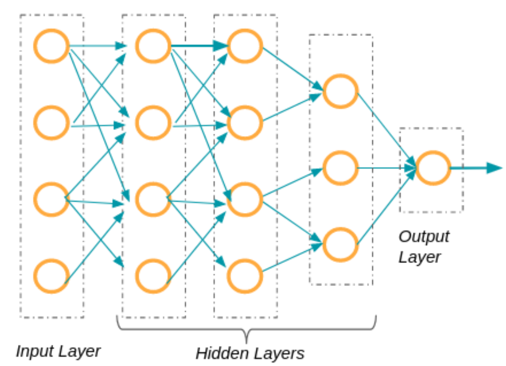
Deep learning (DL)

- DL is a set of deep neural networks architectures for learning data representations with several levels of abstraction.

- DL models are vaguely inspired by information processing in biological nervous systems.



A Multipolar Neuron



A Neural Network

- They use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation.

Conventional machine learning

- ML algorithms (e.g., classification) are limited in their capacity to process natural data in their raw form.

- Efforts have been dedicated for decades to engineer features facilitating prediction in different classification domains:

Example:

Visual recognition: *edges, corners, SIFT, etc.*

Text classification : *Word frequency, punctuations , etc.*

Speech recognition: *MFCC, DCT, etc.*

Conventional machine learning

- Features are engineered, and then extracted from raw data (e.g., text, image, sound, etc.) in a separate process to model learning for classification.

- The features are then passed through data to train a classifier that will be capable of making predictions.

Feature extraction

- Engineering of features is , however, a tedious process for several reasons:

    ☞ Takes a lot of time.
    ☞ Requires expert knowledge.

- For learning-based applications, a lot of time is spent to adjust the features.

- Extracted features often lack a structural representation reflecting abstraction levels in the problem at hand.

Representation learning

- DL aims at learning automatically representations from large sets of labeled data:

  ☞ The machine is powered with raw data.

  ☞ Automatic discovery of representations.

- Representations at different levels of abstraction are learned by simple non-linear transformations.

- The composition of several transformations can approximate very complex functions.

Representation learning

Example:

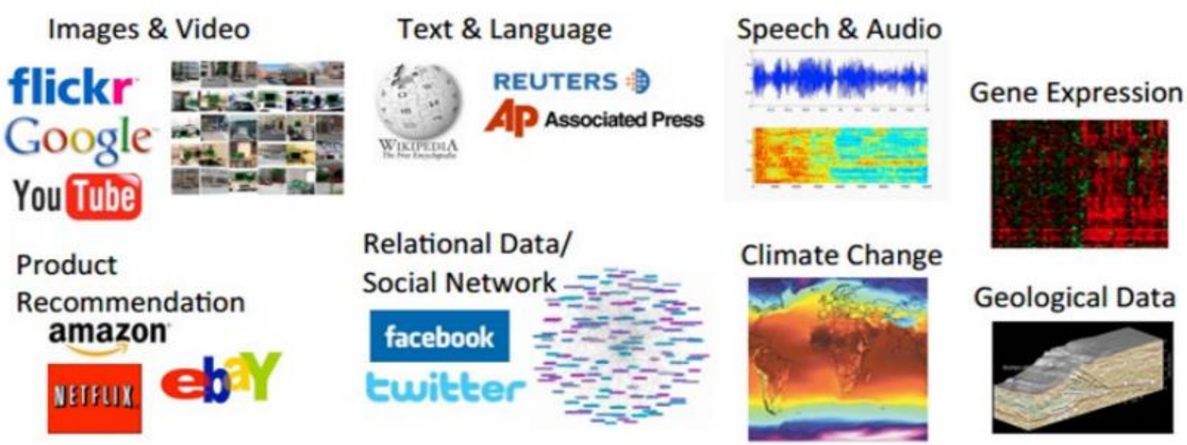
- In text processing:
  - ☞ Sentences are made of words
  - ☞ Words are made of letters
  - ☞ Letters are made of edges.

- In image processing:
  - ☞ Objects are made of local parts (e.g., head, torso, etc.)
  - ☞ Local parts are made of lines and corners.
  - ☞ Corners and lines are made of edges.

Deep Learning (DL) applications

- Recently, DL has enabled a huge progress in several domains :

  ☞ Speech recognition (> 20% of accuracy).
  ☞ Objet recognition in images (> 20% of accuracy).
  ☞ Machine translation.
  ☞ Bioinformatics and drug design, etc.



Deep learning (DL) models

- Several DL models have been proposed :

    ☞ Autoencoders (Aes)

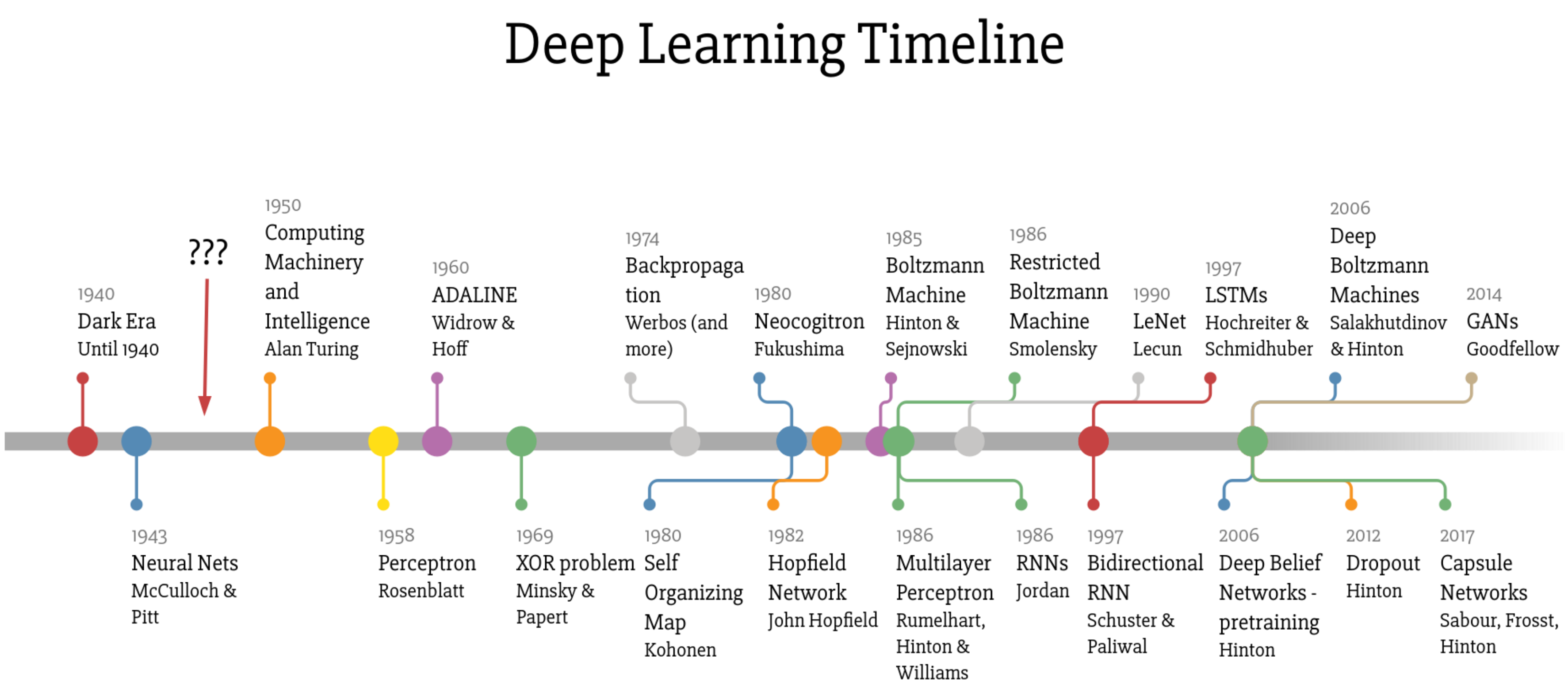
    ☞ Deep belief networks (DBNs)

    ☞ Convolutional neural networks (CNNs).

    ☞ Recurrent neural networks (RNNs).

    ☞ Generative adversial networks (GANs), etc.

Deep learning (DL) models



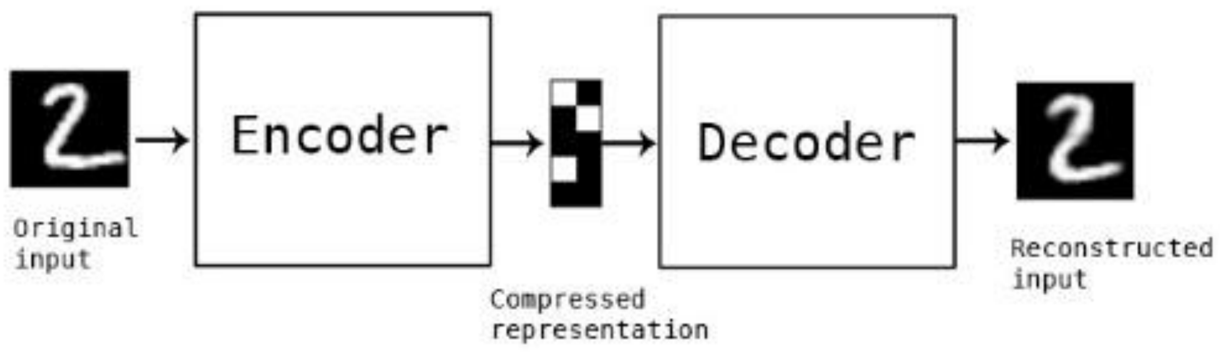
Made by Favio Vázquez

Autoencoders

Autoencoders (AEs)

- An AE is an ANN with a symmetric structure, where a hidden layer $\mathbf{h}$ represents an encoding of the input data $\mathbf{x}$.

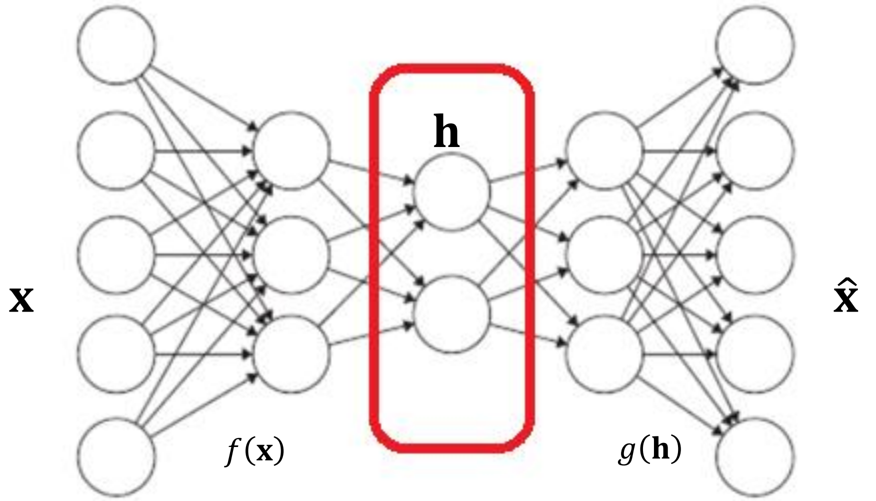
- AEs can have as many layers as needed, usually placed symmetrically in the encoder and decoder parts.



Autoencoders

- AE can learn to compress data from the input layer $\mathbf{x}$ into a short code $\mathbf{h}$, and then decompress the code into something that closely matches the original data $\hat{\mathbf{x}}$.

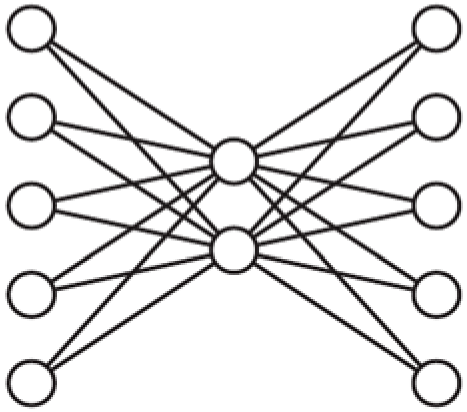
- Let $\mathbf{h} = f(\mathbf{x})$ be the coding function and $\hat{\mathbf{x}} = g(\mathbf{h})$ the decoding (reconstruction) function. If the AE succeeds : $\mathbf{x} \approx g(f(\mathbf{x}))$.



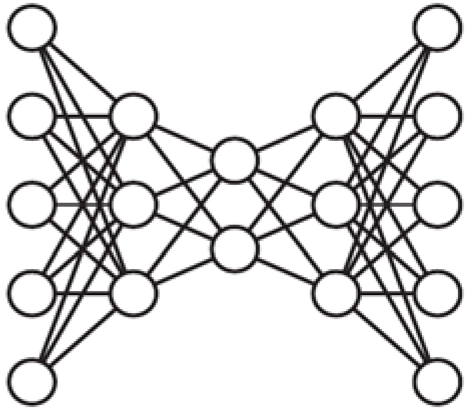
Under-complete AE

- An AE is under-complete if the encoding layer has a lower dimensionality than the input.

  ☞ The AE is forced to learn a more compact representation.



(a) Shallow undercomplete

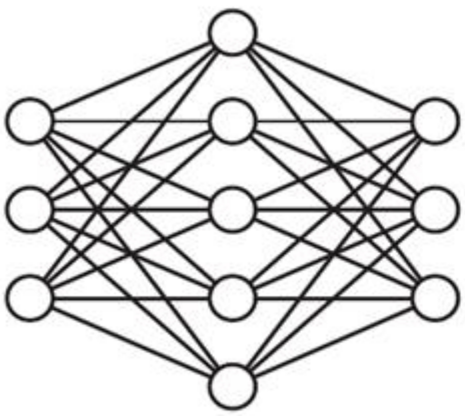


(c) Deep undercomplete

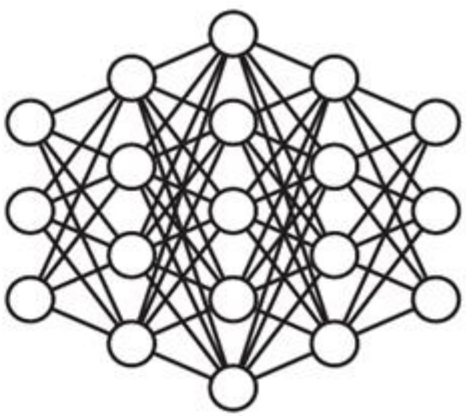
Over-complete AE

- An AE is over-complete if the encoding layer has a higher dimensionality than the input.

☞ To avoid learning the identity function (not interesting), some restrictions (i.e., regularization) can be applied.



(b) Shallow overcomplete



(d) Deep overcomplete

Learning autoencoders

- Since AEs are restricted to copy approximately the input data, they are forced to prioritize which aspect of the data should be copied (i.e., learn useful properties of data).

- The learning procedure for an AE is usually based on minimizing a loss function $L: \mathbb{R}^d \to \mathbb{R}$:

$$L\left(\mathbf{x}, g\left(f(\mathbf{x})\right)\right)$$

- A typical loss function is the mean squared error (MSE):

$$L\left(\mathbf{x}, g\left(f(\mathbf{x})\right)\right) = \left\|\mathbf{x} - g\left(f(\mathbf{x})\right)\right\|^{\mathbf{2}}$$

Learning autoencoders

- AE parameters are learned by unsupervised learning using the principle of backpropagation.

- Suppose that we have a set of training examples:

$$\mathcal{D}_{train} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}\}$$

- In the simple case of one hidden layer, the encoder maps the input $\mathbf{x}$ to the code $\mathbf{h}$ as follows :

$$h_j = \sigma\left(\mathbf{w}_j{}^T \mathbf{x} + b_j\right)$$

- Where σ is an activation function (e.g., Sigmoid, ReLu).

Learning autoencoders

- The decoder maps the code $\mathbf{h}$ for reconstructing the input data $\mathbf{x}$ as follows:

$$x_i = \sigma'\left(\mathbf{w}_i'^{T}\mathbf{h} + b_i'\right)$$

- Where σ' is an activation function that can be different from σ.

- Let $(\mathbf{W}, \mathbf{b})$ and $(\mathbf{W}', \mathbf{b}')$ denote all the parameters of the encoder and decoder, respectively. The MSE function will be given by:

$$L\left(\mathbf{x}, g\left(f(\mathbf{x})\right)\right) = ||\mathbf{x} - \sigma'(\mathbf{W}'^{T}\sigma(\mathbf{W}^{T}\mathbf{x} + \mathbf{b}) + \mathbf{b}')||^{\mathbf{2}}$$

Regularizing autoencoders

- Regularization allows the AE to have other properties beside copying the input to the output (e.g., sparsity, denoising, etc.).

  ☞ Sparsity in AE means that most of the values in a given code representation $\mathbf{h}$ are zero or close to zero. It can be achieved by using penalty term $\Omega(\mathbf{h})$ to the loss function:

$$\tilde{L}\left(\mathbf{x}, g\left(f(\mathbf{x})\right)\right) = L\left(\mathbf{x}, g\left(f(\mathbf{x})\right)\right) + \Omega(\mathbf{h})$$

  ☞ Denoising aims at minimizing the loss function for a copy of the input data corrupted by noise $\tilde{\mathbf{x}}$:

$$L\left(\mathbf{x}, g\left(f(\tilde{\mathbf{x}})\right)\right)$$

Applications of autoencoders

- Classification: reduce or transform the training data in order to achieve better performance in classification.

- Data compression: train AEs for specific types of data to learn efficient compressions.

- Detection of abnormal patterns: identify discordant/abnormal instances by analyzing generated encodings.

- Hashing: summarize input data onto binary vectors for faster search in large databases.

- Visualization: project data onto 2 or 3 dimensions with an AE for graphical representation.

Restricted Boltzmann machines

Restricted Boltzmann machines (RBMs)

- RBMs are generative stochastic ANNs that can learn a probability distribution over their sets of inputs. They can be used for several applications:

    ☞ dimensionality reduction,

    ☞ classification,

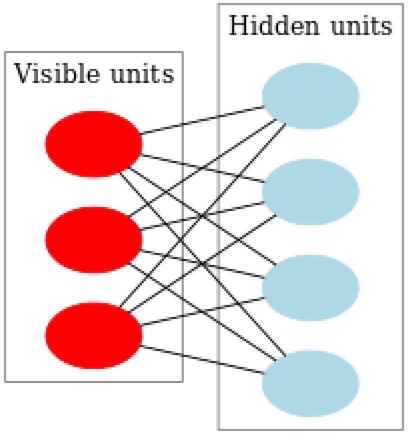
    ☞ collaborative filtering,

    ☞ feature learning

    ☞ topic modelling, etc.

Restricted Boltzmann machines (RBMs)

- More formally, an RBM network is a bipartite graph with two groups of (binary) nodes: visible $\mathbf{x}$ and hidden $\mathbf{h}$.



- The groups have a symmetric connection between them, but there are no connections between nodes within a group.

- RBMs can be stacked to build DBNs. The resulting network is optionally fine-tuned by gradient descent / backpropagation.

RBM energy function

- A standard RBM with n hidden and m visible units consists of:

  ☞ A matrix $\mathbf{W}$ of weights w_{ij} associated with the connection between a visible unit x_i and a hidden unit h_j.

  ☞ Bias weights (offsets) a_i and b_j associated with visible unit x_i and hidden unit h_j, respectively.

- An energy function is defined for the RBM as follows :

$$E(\mathbf{x}, \mathbf{h}) = \sum_i a_i x_i + \sum_j b_j h_j + \sum_{i,j} w_{ij} x_i h_j$$

Probability distributions in RBMs

- The joint probability of all variables is given by the distribution :

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{x},\mathbf{h})}$$

- With Z is a normalization constant (partition function).

- The marginal probability of a visible (input) vector $\mathbf{x}$ is the sum of $p(\mathbf{x}, \mathbf{h})$ over all possible hidden layer configurations:

$$p(\mathbf{x}) = \sum\nolimits_{\boldsymbol{h}} p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} \sum\nolimits_{\boldsymbol{h}} e^{-E(\mathbf{x},\mathbf{h})}$$

Conditional probabilities in RBM

- We have also the following conditional distributions:

$$p(\mathbf{x}|\mathbf{h}) = \prod_{i=1}^{m} p(x_i|\mathbf{h}) \qquad p(\mathbf{h}|\mathbf{x}) = \prod_{j=1}^{n} p(h_j|\mathbf{x})$$

- Elements of $\mathbf{h}$ and $\mathbf{v}$ are activated as follows:

$$p(h_j = 1|\mathbf{x}) = \sigma\left(\sum_{i=1}^{m} w_{ij}x_i + b_j\right)$$

$$p(x_i = 1|\mathbf{h}) = \sigma\left(\sum_{j=1}^{n} w_{ij}h_j + a_i\right)$$

Learning RBM parameters

- The RBM parameters $\mathbf{W}, \mathbf{a}$ and $\mathbf{b}$ are learned by maximizing the probabilities assigned to a set of training data $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$.

$$(\mathbf{W}^*, \mathbf{a}^*, \mathbf{b}^*) = \text{argmax}_{\mathbf{W},\mathbf{a},\mathbf{b}} \log \left[\prod_{k=1}^{N} p\left(\mathbf{x}^{(k)}\right) \right]$$

- Use methods such as contrastive divergence: Repeat the following steps until convergence:

  ☞ Take a training sample $\mathbf{v}$, sample $\mathbf{h} \sim p(\mathbf{h}|\mathbf{x})$.

  ☞ From $\mathbf{h}$, sample a reconstruction $\mathbf{x}' \sim p(\mathbf{x}|\mathbf{h})$, then resample $\mathbf{h}' \sim p(\mathbf{h}|\mathbf{x}')$.

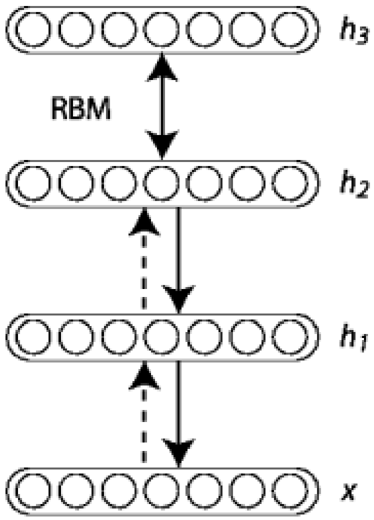
  ☞ Update the matrix $\mathbf{W}$ by $\Delta\mathbf{W} = \epsilon\left(\mathbf{x}\mathbf{h}^{\mathbf{T}} - \mathbf{x}'\mathbf{h}'^{\mathbf{T}}\right)$.

  ☞ Update $\mathbf{a}$ and $\mathbf{b}$ by $\Delta\mathbf{a} = \boldsymbol{\epsilon}(\mathbf{x} - \mathbf{x}')$ and $\Delta\mathbf{a} = \epsilon(\mathbf{h} - \mathbf{h}')$.

Deep belief networks (DBN)

- A DBN is a generative graphical model composed of multiple layers of latent (hidden) variables.

- DBNs can be viewed as a composition of simple, unsupervised networks such as autoencoders (AEs) or restricted Boltzmann machines (RBMs).



Convolutional neural networks

Visual recognition

- Visual recognition is one of the complex tasks humans are capable to perform naturally and effortlessly.
- When a computer sees an image (takes an image as input), it will see an array of pixel values.

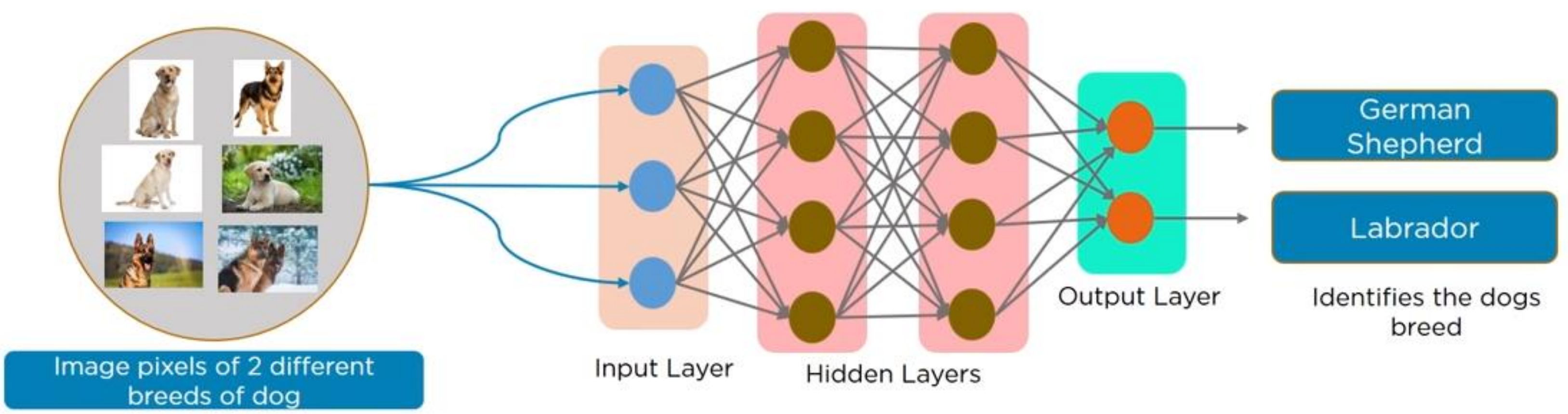
What We See

08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

What Computers See

Visual recognition

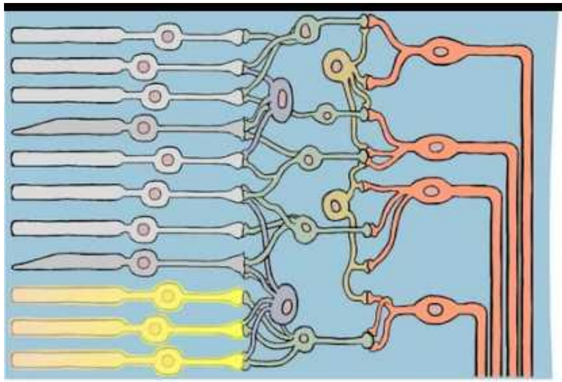
- Let's say we have a JPG color image in of size 480 x 480 pixels.
- Having 3 color channels (RGB), the representative array will contain 480 x 480 x 3 discrete numbers in the range [0, 255].
- The idea is to give the ANN an array of these numbers and output class probabilities of objects:



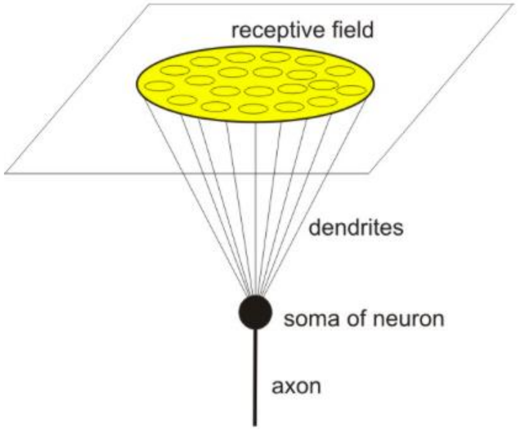
Convolutional neural networks (CNN)

- Convolutional neural networks (CNNs) is a class of ANNs inspired by biological processes:

  ☞ Each individual neuron in retina responds to stimuli in a restricted region of the visual field known as the receptive field.



Light reflected by objects in visual space is absorbed by light-sensitive cells in the retina called **photoreceptors**.



Convolutional neural networks (CNN)

- CNNs learn local filters extracting features in a manner similar to the visual cortex.

- Note that traditional algorithms used hand-engineered features.

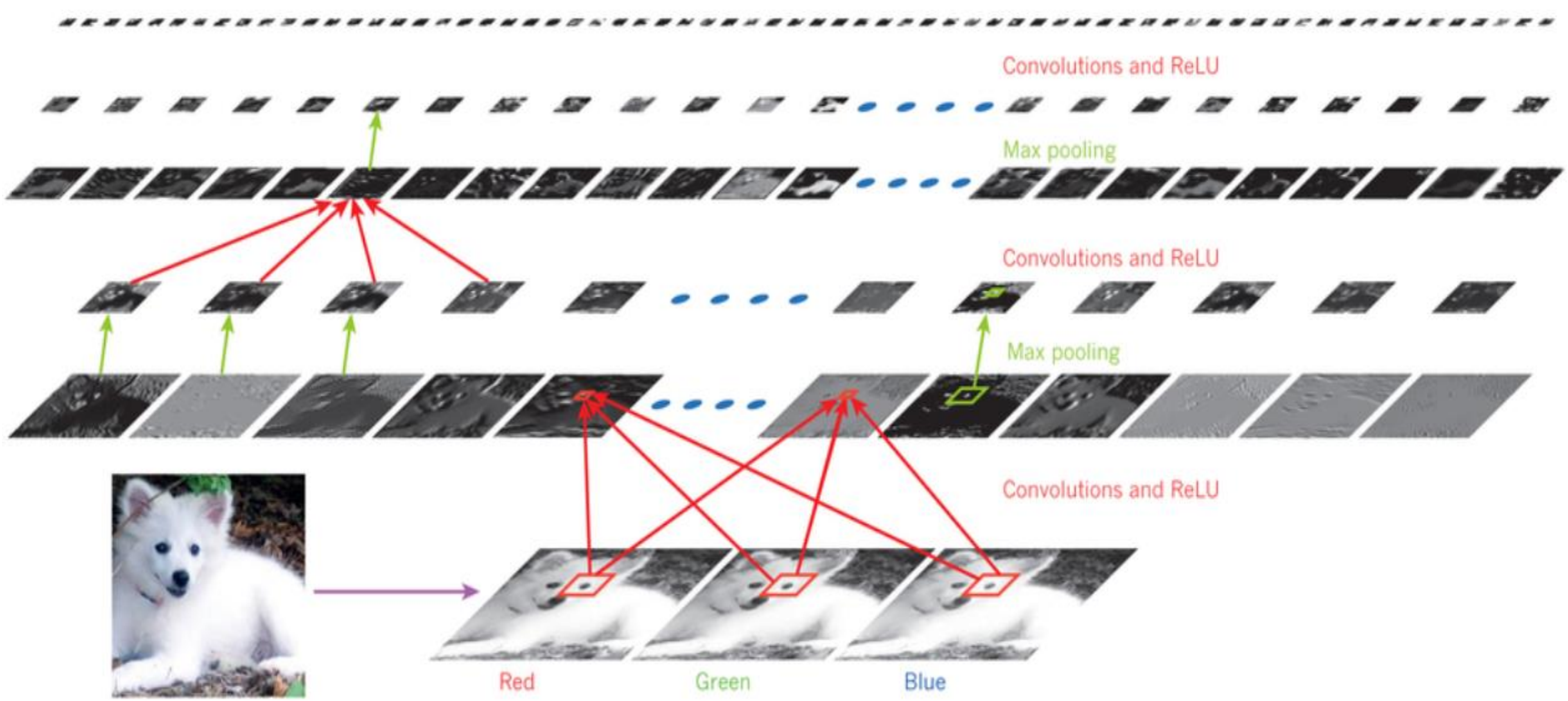
- A CNN architecture is based on weight sharing, making recognition shift/space invariant.

- CNNs are most commonly applied to analyzing visual imagery. They have also applications in:

  ☞ Recommender systems,
  ☞ Natural language processing, etc.

CNN hierarchical structure

- The layers of CNN represent a hierarchy of feature abstraction.

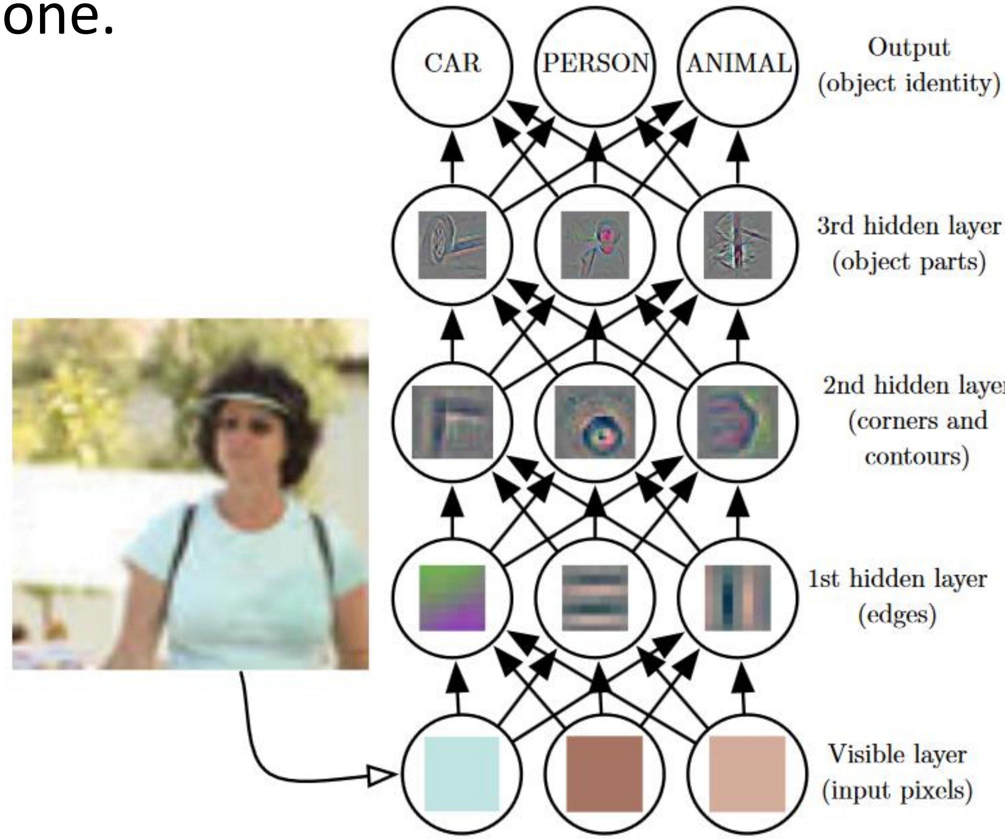


Feature abstraction

- The information flows from bottom-up, the first lower level acts as an oriented edge detector.

- Each hidden layer represents a level of abstraction slightly higher than the lower one.

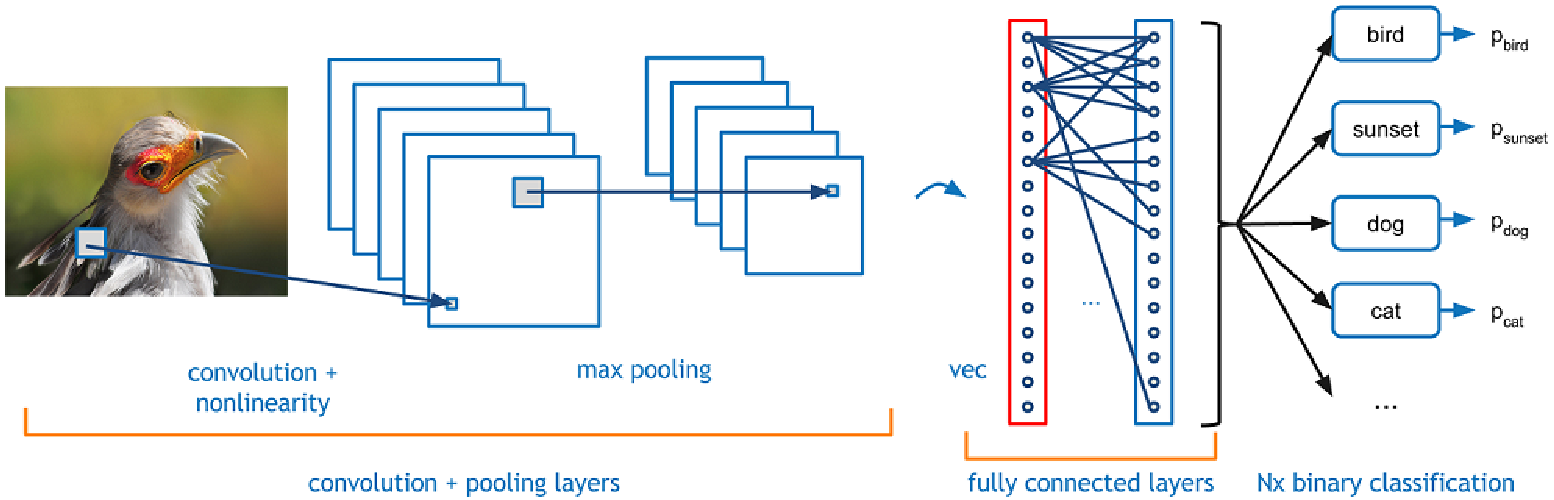
Goodfellow et al. (2016)
Deep Learning. MIT Press



Convolutional neural networks (CNN)

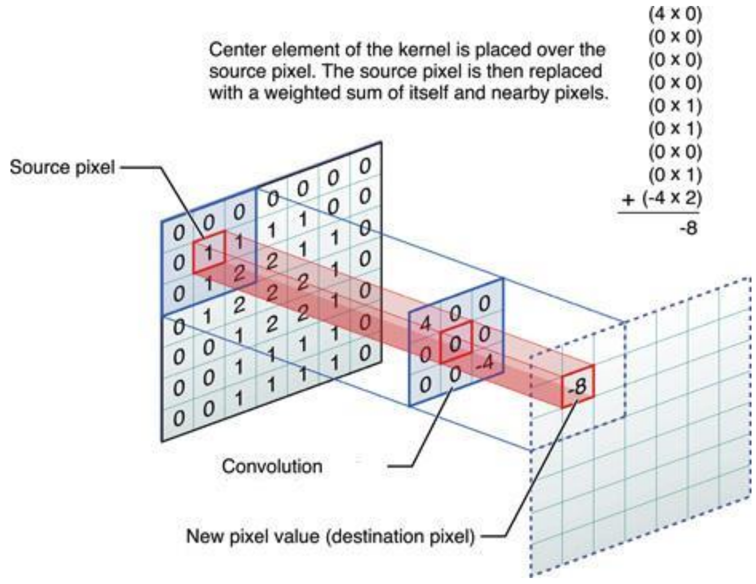
- The hidden layers of a CNN typically consist of:

☞Convolutional layers.
☞Pooling layers.
☞Fully connected layers.



Convolutional layers

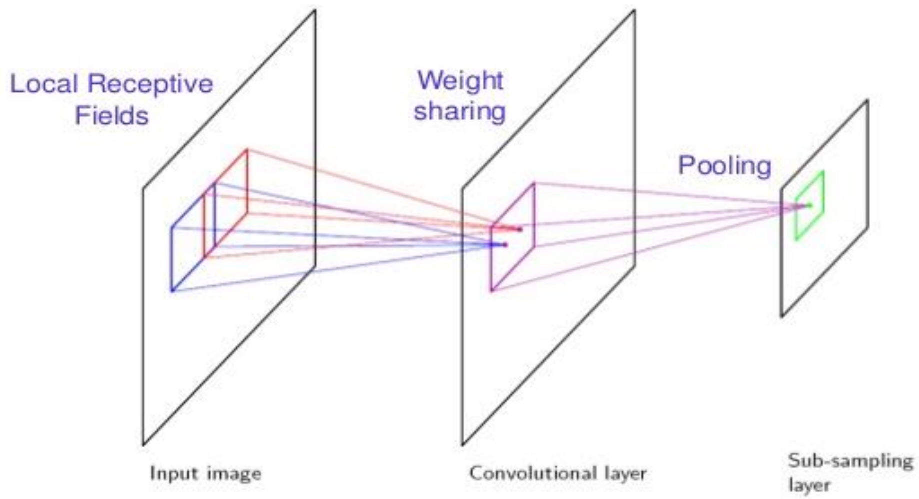
- The convolution operation emulates the response of an individual neuron to visual stimuli it the receptive field.



- Convolutional layers apply this operation to each location of the input image, then pass the result to the next layer.

Weight sharing

- Compared to fully connected networks, CNNs use weight sharing allowing to reduce the number of free parameters and building deeper networks with fewer parameters.



(LeCun et al., 1989)

Example:

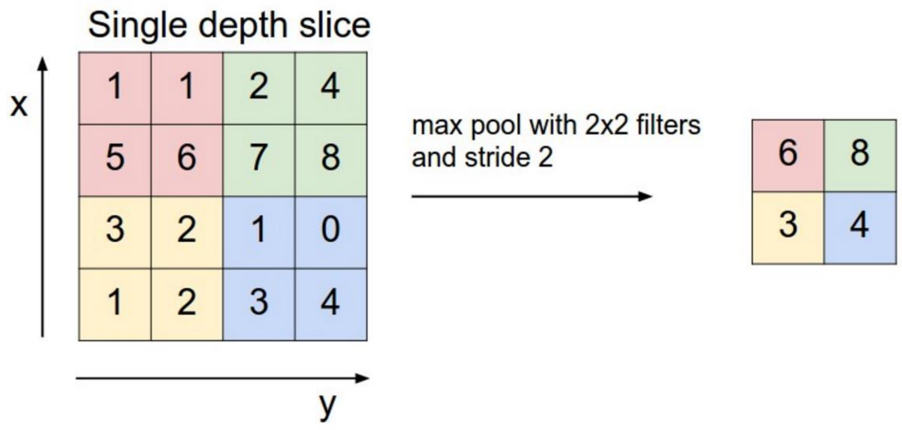
Regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters.

Pooling layers

- Pooling (down-sampling) combines the outputs of neuron clusters at one layer into a single neuron in the next layer.

Example:

Max/average pooling uses the maximum/average value from each of a cluster of neurons at the previous layer.



Pooling layers

- The intuitive reasoning behind pooling:

  ☞ When a feature is present in the original image, it exact location is not as important as its relative location to the other features.

- Pooling has also the additional role of:

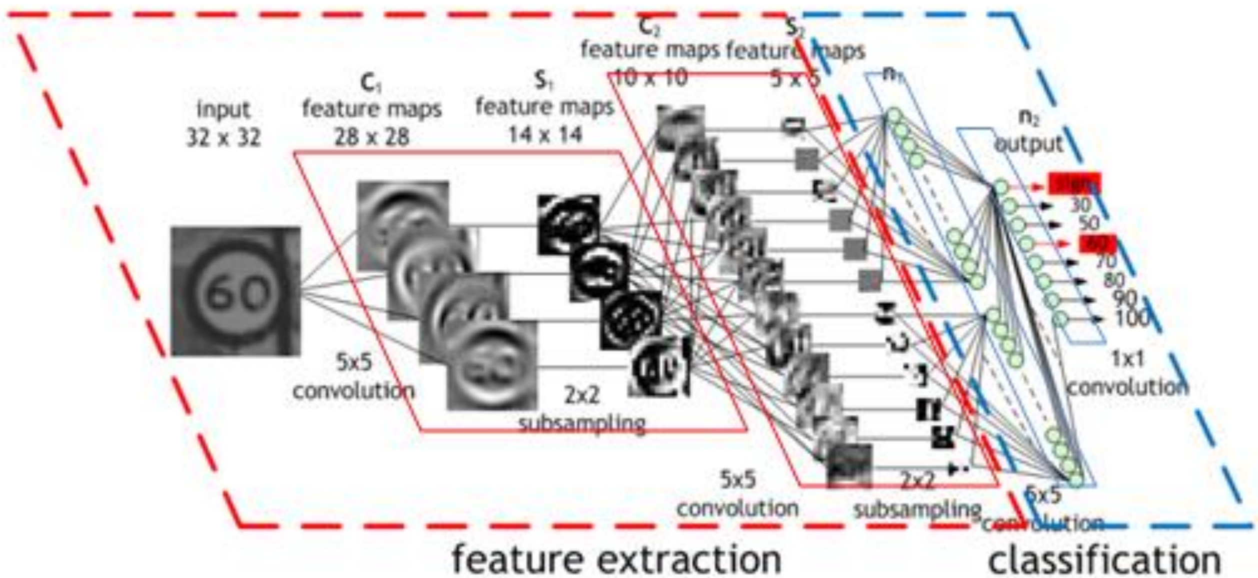
  ☞Reducing the spatial dimension of the input image.

  ☞Reducing the amount of learned parameters (e.g., 75%)

Fully connected layers

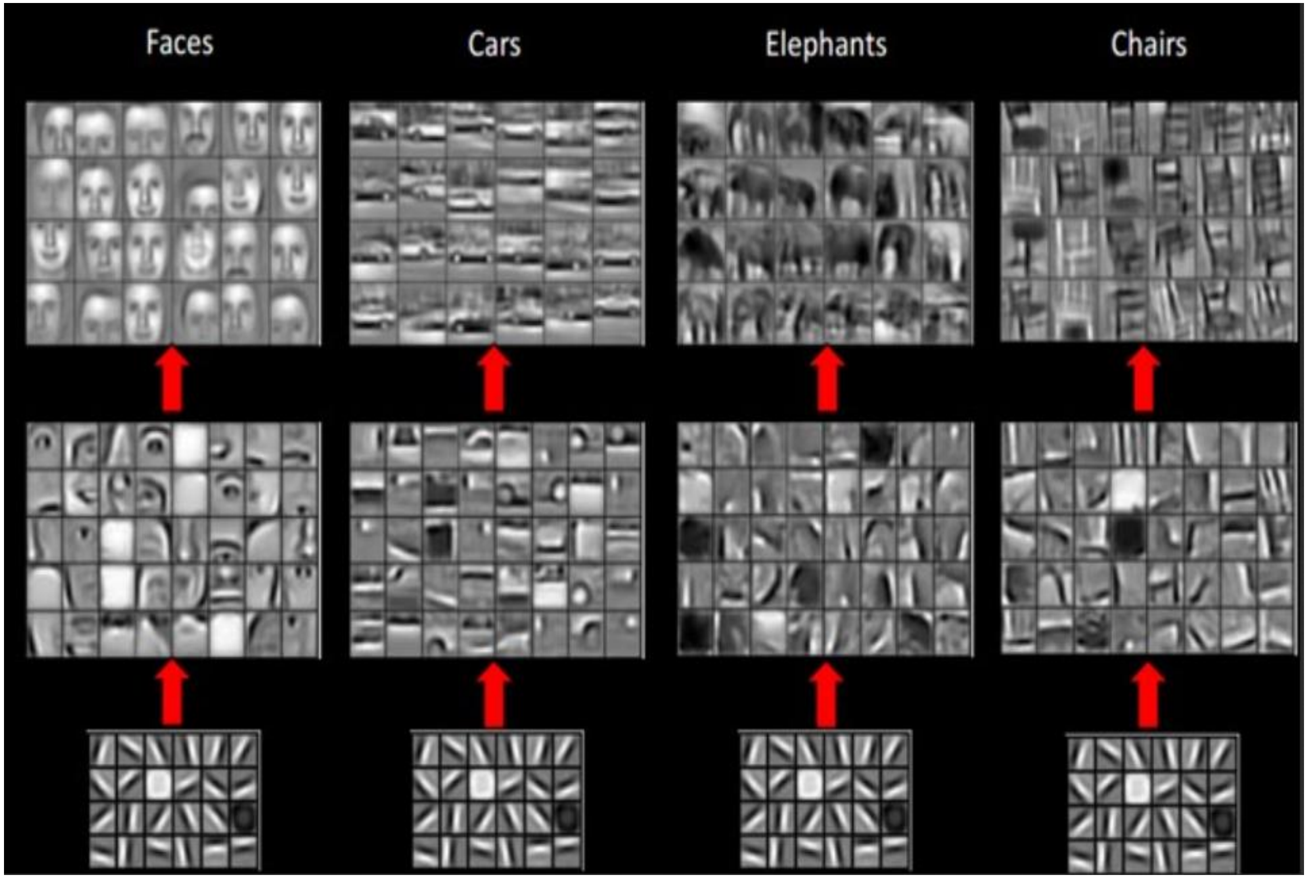
- Connect every neuron in one layer to every neuron in another layer (similar to an ANN).

- On top of high-level features, a fully connected layer is used to derive class probabilities using the softmax function.



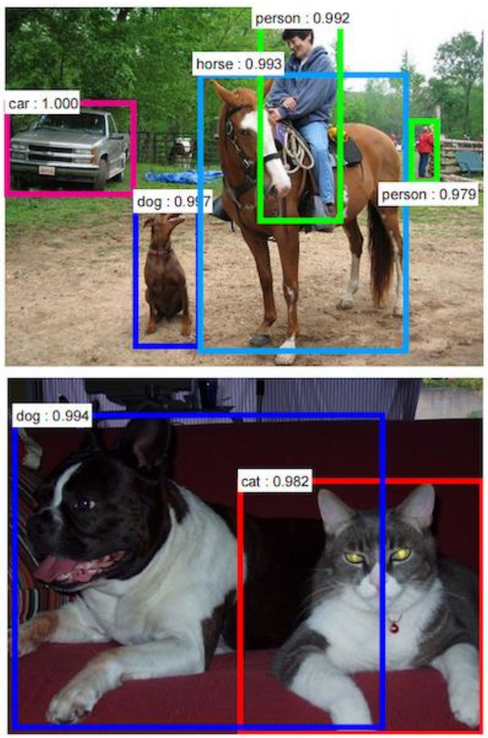
Representation learning

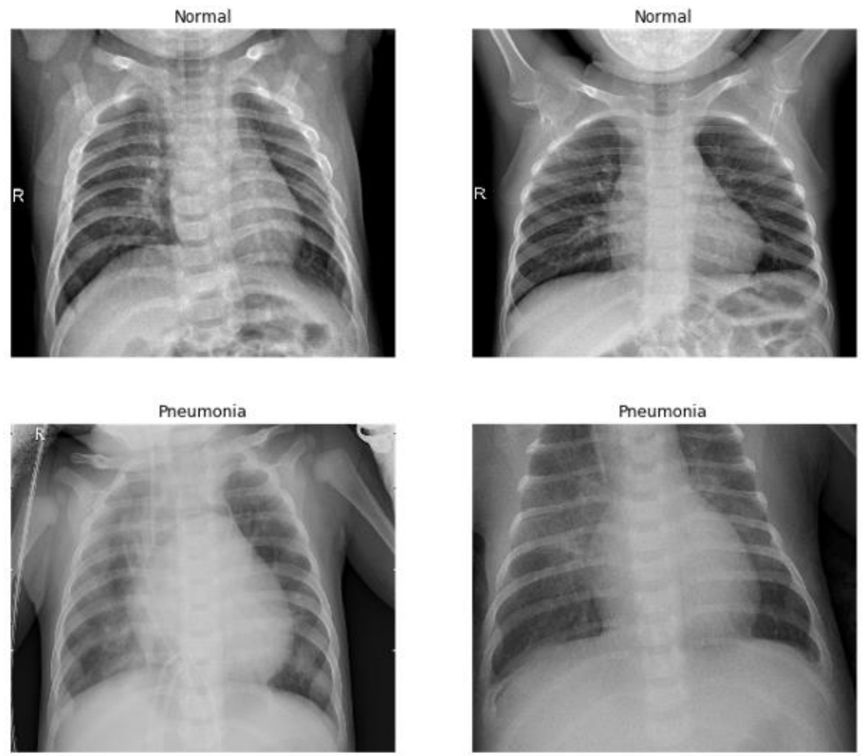
- Examples of learned features using CNN for different object categories.



Applications for CNNs

- Several applications: object recognition, medical diagnosis from images, etc.





Applications for CNNs

- Autonomous vehicles.

Waymo / Google Self-Driving Car

Tesla Autopilot

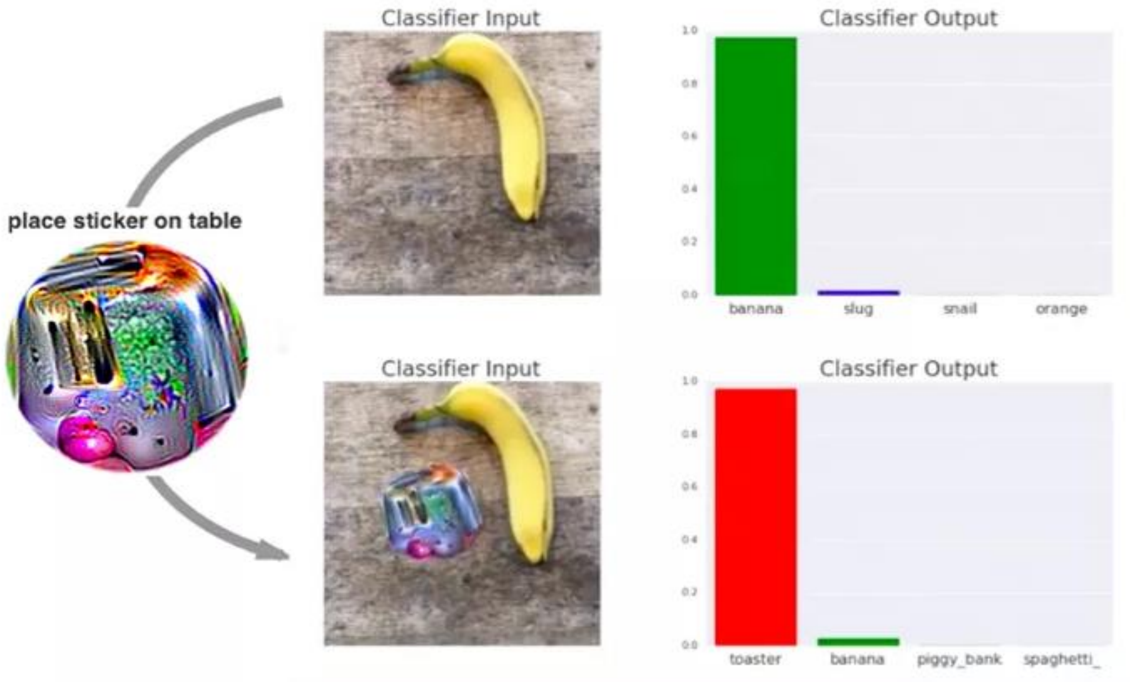
Uber

nuTonomy

Limitations for CNNs

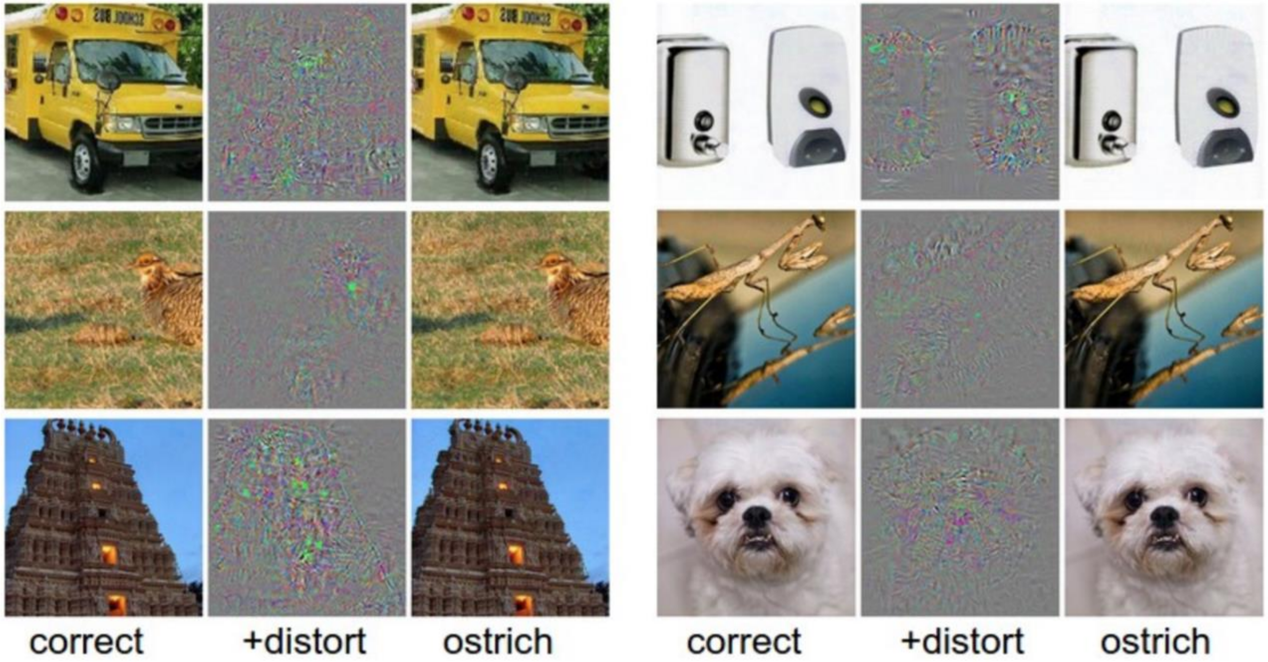
- Despite their performance, CNNs can be fooled by simple "adversial" examples.

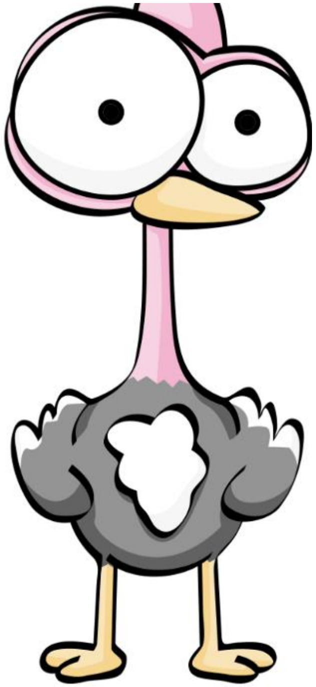
Example: Adding false salient objects can false the CNN output.



Limitations for CNNs

- Adding distortions can also false the CNN output.





Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. **Intriguing properties of neural networks**. ICLR (2013).

Recurrent neural networks

Sequential data

- Not all problems can be converted into one with fixed length inputs and outputs.

- Problems such as speech recognition or time-series prediction require a system to store and use context information.

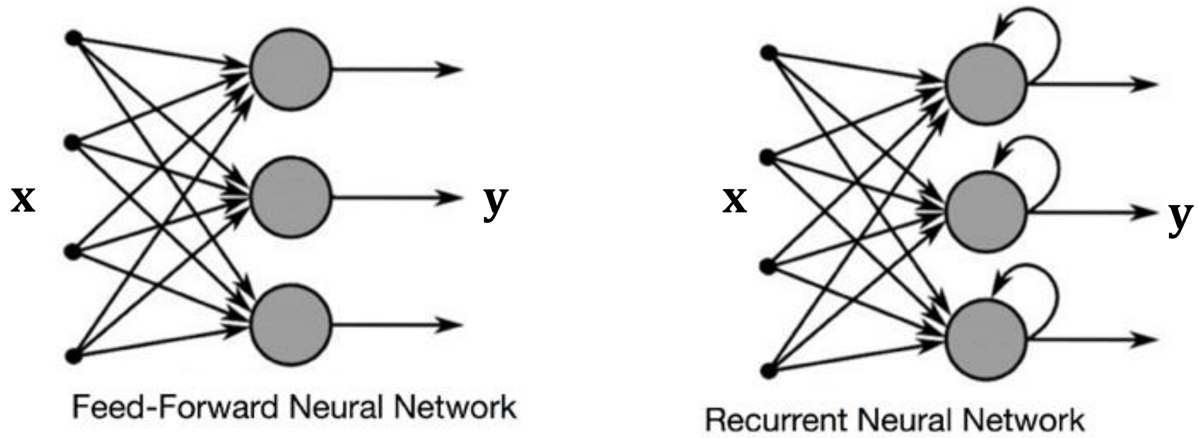
Example: Output YES if the number of 1s in a sequence is even, else NO.

1000010101 → YES.
100011 → NO.

- Standard feedforward networks have no notion of order in time. They can not process sequential data.

Recurrent neural networks (RNNs)

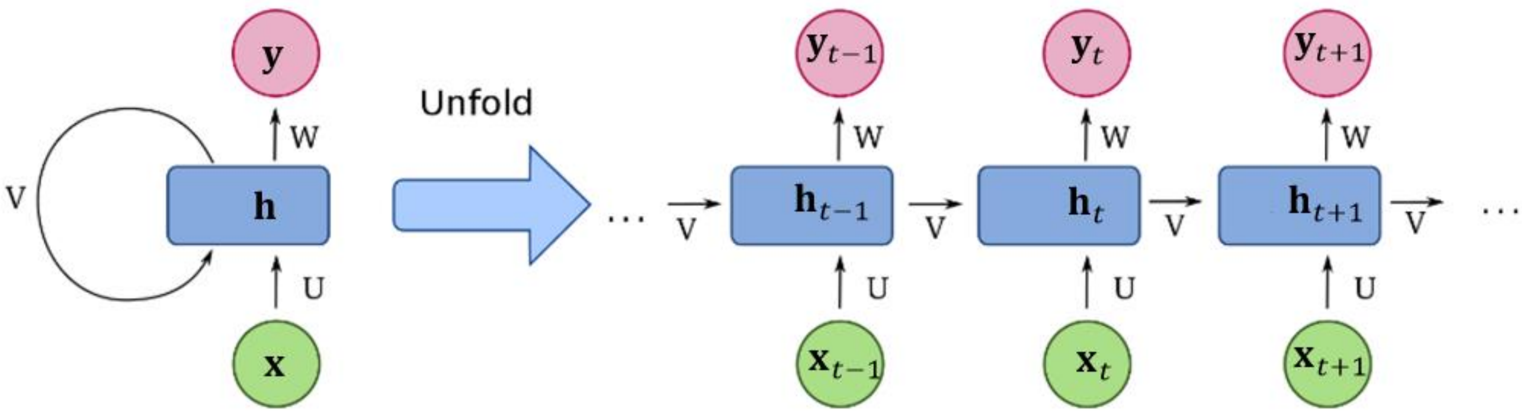
- Recurrent neural nets take as their input not just the current example an input, but also the previous output/hidden states.



- Note that weights are shared over time steps.

Recurrent neural networks (RNNs)

- Copies of the RNN cell are made over time (unfolding), with different inputs at different time steps



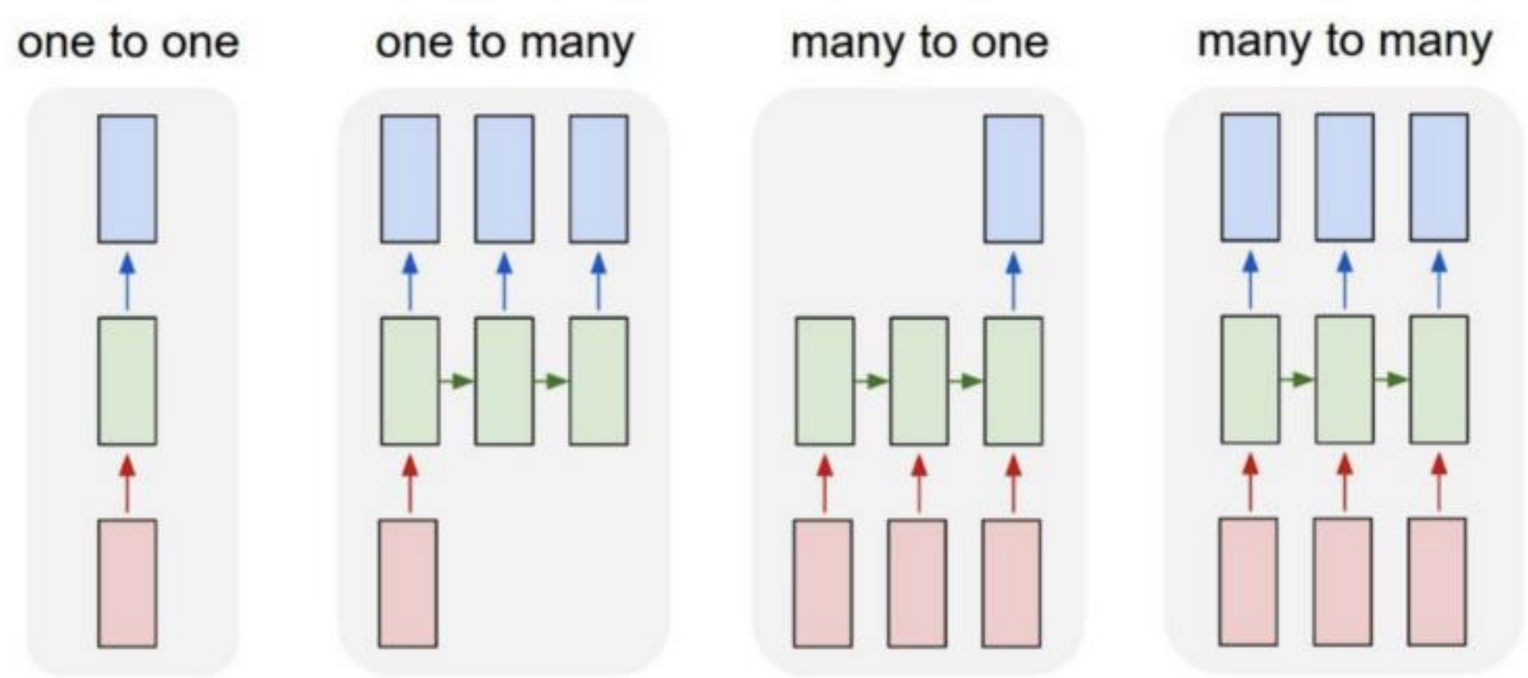
- At each time step, values of the hidden weights and output are given by:

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{y}_t = \sigma\left(\mathbf{W}\mathbf{h}_t + \mathbf{b}_y\right)$$

RNN decision making

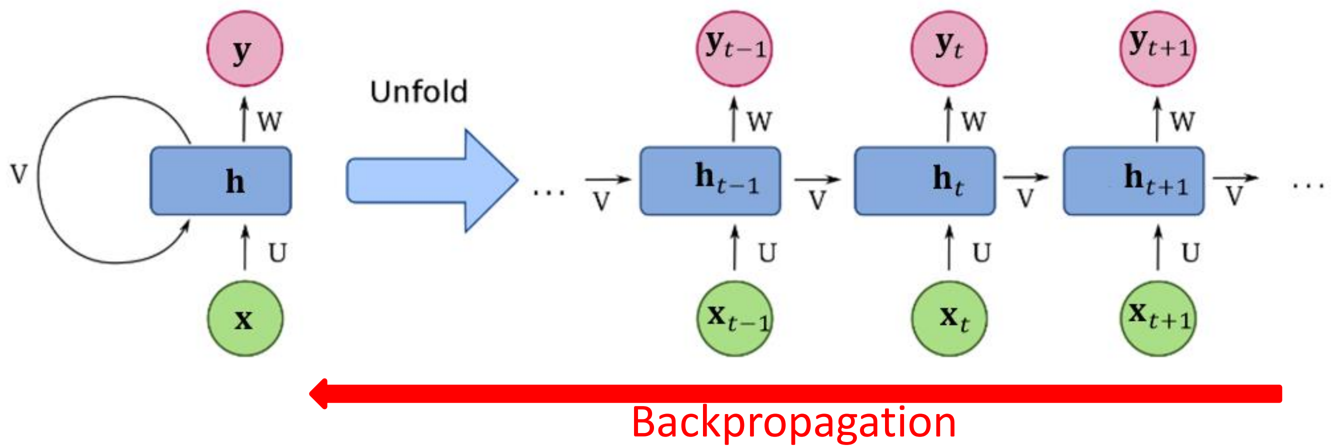
- Many scenarios for decision making can be achieved out using RNN implementation.



Backpropagation through time (BPTT)

- BPTT is basically the same as in the feedforward ANN, but the propagation is made on the unfolded RNN.

- The error is back-propagated from the last to the first time step. This allows calculating the error for each time step, which allows updating the weights.



Issues with standard RNNs

- The BPTT can be computationally expensive when we have a high number of time steps.

- There are also two main issues:

  ☞ Exploding gradients: High gradients produce high weights, without much reason.

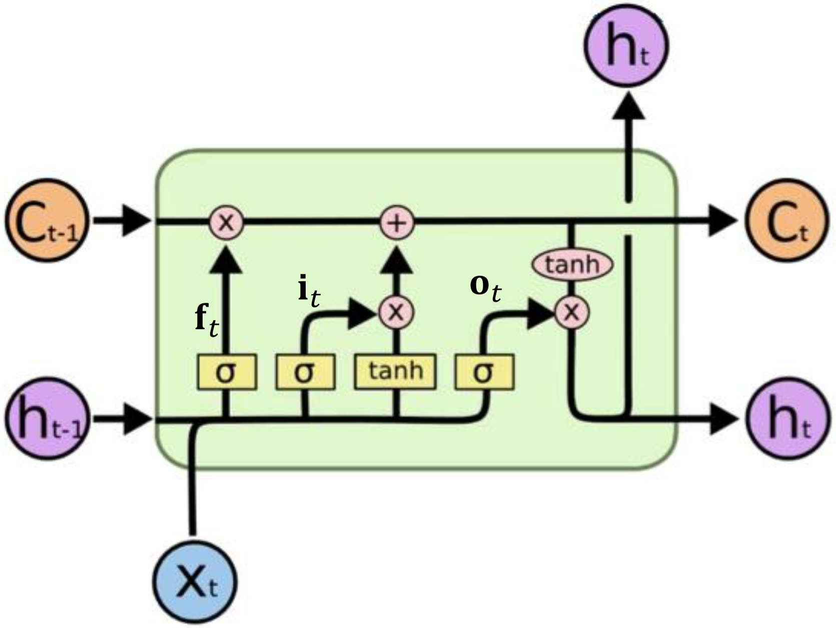
  Can be solved by truncating the gradients 

  ☞ Vanishing gradients: very small gradients makes the model stop learning or take way too long to converge.

Long Short-Term Memory Units (LSTMs)

- To resolve the vanishing gradient problem, LSTMs are an extension for RNNs, which basically extends their memory.

- LSTMs enable RNNs to remember their inputs over a long period of time by using gates allowing to read (input), write (output) and delete (forget) information from memory.



LSTMs activation functions

- The gates are activated using sigmoid functions.
- The compact equations (in vector form) for the forward pass of an LSTM unit with a forget gate are as follows:

$$\text{Gates}\begin{cases}\mathbf{f}_t = \sigma_g\left(\mathbf{W}_f\mathbf{x}_t + \mathbf{V}_f\mathbf{h}_{t-1} + \mathbf{b}_f\right)\\[2ex] \mathbf{i}_t = \sigma_g(\mathbf{W}_i\mathbf{x}_t + \mathbf{V}_i\mathbf{h}_{t-1} + \mathbf{b}_i)\\[2ex] \mathbf{o}_t = \sigma_g(\mathbf{W}_o\mathbf{x}_t + \mathbf{V}_o\boldsymbol{h}_{t-1} + \mathbf{b}_o)\end{cases}$$

$$\text{Memory}\left\{\mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \sigma_c(\mathbf{W}_c\mathbf{x}_t + \mathbf{V}_c\mathbf{h}_{t-1} + \mathbf{b}_c)\right.$$

$$\text{Output}\left\{\mathbf{h}_t = \mathbf{o}_t \otimes \sigma_c(\mathbf{c}_t)\right.$$

σ_c: tanh

σ_g: sigmoid

$\mathbf{c}_0 = 0$

$\mathbf{h}_0 = 0$

LSTM parameter learning

- LSTM total error on a set of training sequences is minimized by the gradient descent.

- The learning of LSTM parameters is performed by using backpropagation.

- To avoid gradient vanishing, LSTM continuously feeds error back to each of the gates until they learn to cut off the value.

- Thus, regular backpropagation is effective at training an LSTM unit to remember values for long durations.

Application of LSTMs

- Several applications for LSTMs:

    - Time series prediction
    - Anomaly detection in times series
    - Speech recognition
    - Handwriting recognition
    - Human action recognition
    - Music composition
    - Grammar learning
    - Language translation, etc.

Generative adversial networks

Limitations of DL models

- Reminder:

  ☞ *Discriminative models : learn the boundary between classes.*

  ☞ *Generative models: model the distribution of individual classes.*

- Most of DL methods for classification (e.g., CNNs) are based in discriminative models.

- However, the performance of these methods is very dependent of the training sets. Too limited training sets can:

  ☞ Produce overfitting.

  ☞ Make algorithms vulnerable to adversial examples.

Generative adversial networks (GANs)

- GANs are DL models combining discriminative and generative nets in their structure.

    ☞ One network generates candidates (generator)

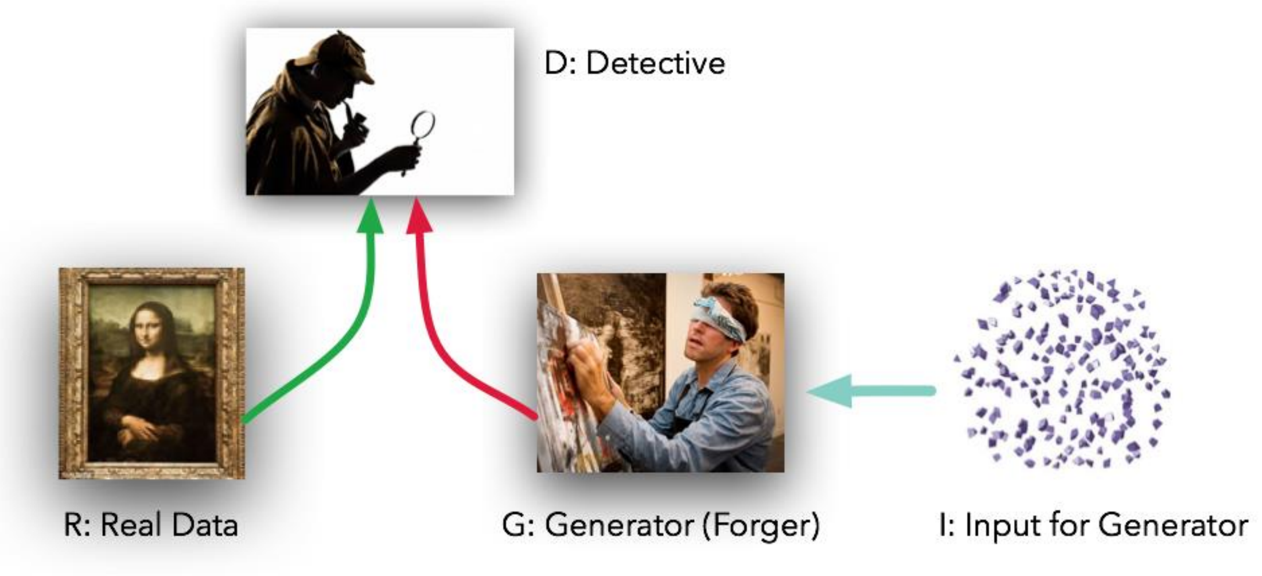
    ☞ One network evaluates the generated data (discriminator).

- The generator tries to fool the discriminator (i.e., increase its error rate) by producing synthesized instances.

- Synthesized instances appear to have come from the true data distribution.

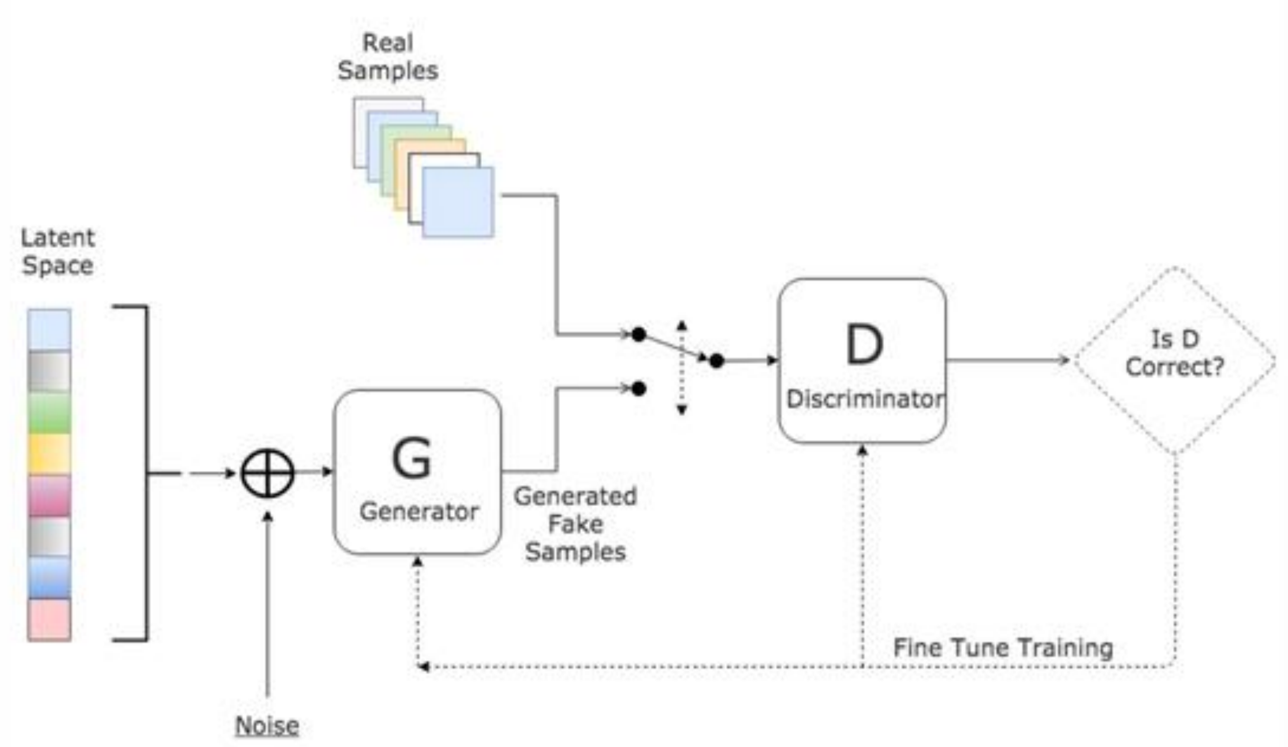
Generative adversial networks (GANs)

- The generator : takes random numbers and generates an image. 
- Discriminator: uses generated image alongside real images as inputs and returns probabilities (1: authentic and 0 faked). 



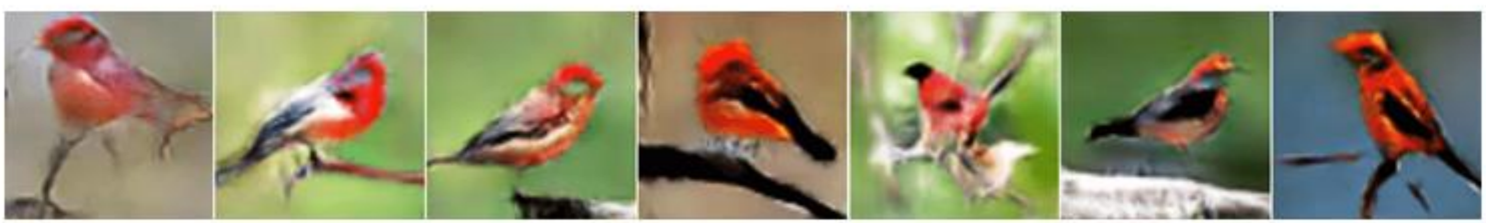
Generative adversial networks (GANs)

- The discriminator is fine-tuned to become more efficient in distinguishing between false and true images.



Generative adversial networks (GANs)

- Here are some examples of authentic and faked images:



Conclusion & short discussion

- DL is an a rapidly expanding and very promising field for improving many applications performance.

- Many questions/challenges, however, are still persisting for DL. Among these, I report the following:

  ☞ How to exploit unlabelled data (unsupervised learning) in DL?
  ☞ Can we reach general artificial intelligence using DL?
  ☞ What is the risk of deploying DL models at large scale?

'Science progresses one funeral at a time.' The future depends on some graduate student who is deeply suspicious of everything I have said.

Geoff Hinton, grandfather of deep learning
September 15, 2017

References

1) Bishop, C. M. (2006) Pattern Recognition and Machine Learning. Chapter 5: Neural Networks.

2) Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. Neural Networks 61: 85-117.

3) Bengio, Y., LeCun, Y., Hinton, G. (2015). Deep Learning. Nature 521: 436-44

4) Goodfellow, I., Bengio, Y. and Courville, A. (2016) Deep Learning. MIT Press.

5) Matthew D. Zeiler and Rob Fergus (2014) . Visualizing and Understanding Convolutional Networks. ECCV, pp. 818-83.

6) You can find more good references here: https://project.inria.fr/deeplearning/files/2016/05/deepLearning.pdf

Thank You!

Any Questions?