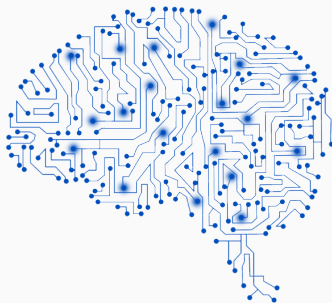


Apprentissage pour l'image

Machine learning for image processing

Course I – Introduction to Artificial Neural Networks: Multiclass logistic classification

Bruno Galerne
Jeudi 10 mars 2022



Most of the slides from **Charles Deledalle's** course "UCSD ECE285 Machine learning for image processing" (30 × 50 minutes course)



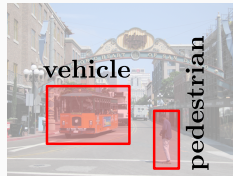
www.charles-deledalle.fr/

<https://www.charles-deledalle.fr/pages/teaching.php#learning>

Computer Vision and Machine Learning



Image

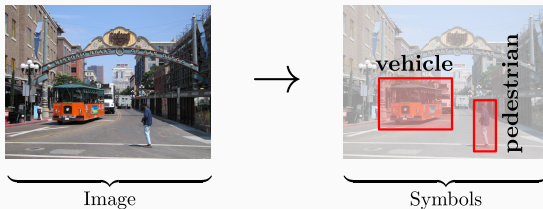


Symbols

Computer vision – Artificial Intelligence – Machine Learning

Definition (The British Machine Vision Association)

Computer vision (CV) is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images.



CV is a subfield of Artificial Intelligence.

Definition (Oxford dictionary)

Artificial Intelligence, *noun*: the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation.

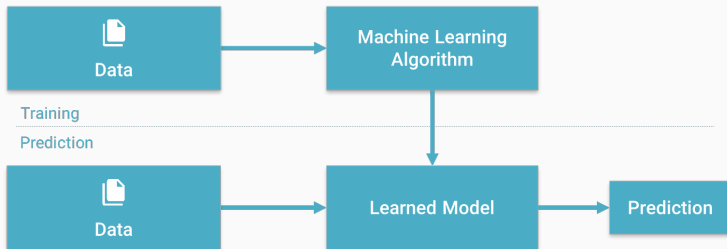
Computer vision – Artificial Intelligence – Machine Learning

CV is a subfield of AI, CV's new very best friend is **machine learning** (ML), ML is also a subfield of AI, but not all computer vision algorithms are ML.

Definition

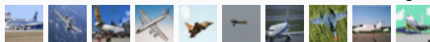
Machine Learning, *noun*: type of Artificial Intelligence that provides computers with the ability to **learn without being explicitly programmed**.

ML provides **various techniques** that can learn from and make predictions on data. Most of them follow the same general structure:

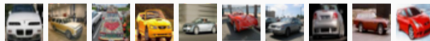


Computer vision – Image classification

airplane



automobile



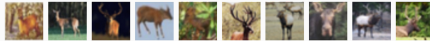
bird



cat



deer



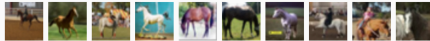
dog



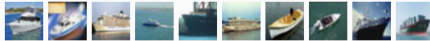
frog



horse



ship

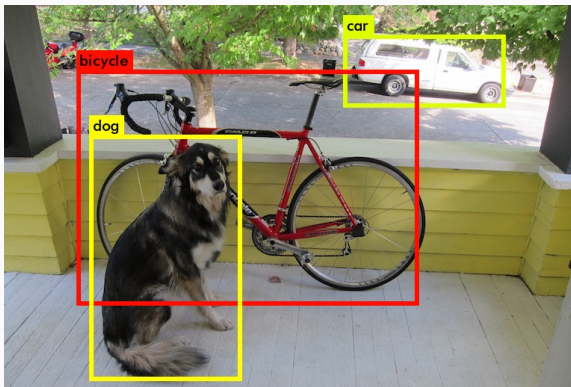


truck



Goal: to assign a given image into one of the predefined classes.

Computer vision – Object detection



(Source: Joseph Redmon)

Goal: to detect instances of objects of a certain class (such as human).

Computer vision – Image segmentation



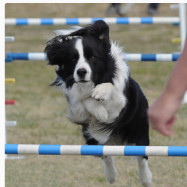
(Source: Abhijit Kundu)

Goal: to partition an image into multiple segments such that pixels in a same segment share certain characteristics (color, texture or semantic).

Computer vision – Image captioning



"girl in pink dress is jumping in air."



"black and white dog jumps over bar."



"young girl in pink shirt is swinging on swing."



"man in blue wetsuit is surfing on wave."



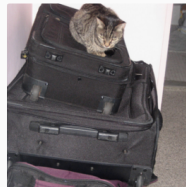
"little girl is eating piece of cake."



"baseball player is throwing ball in game."



"woman is holding bunch of bananas."



"black cat is sitting on top of suitcase."

(Karpathy, Fei-Fei, CVPR, 2015)

Goal: to write a sentence that describes what is happening.

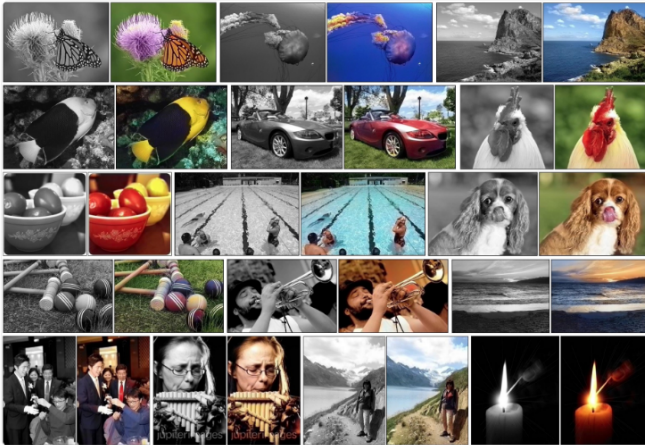
Computer vision – Depth estimation



(Stereo-vision: from two images acquired with different views.)

Goal: to estimate a depth map from one, two or several frames.

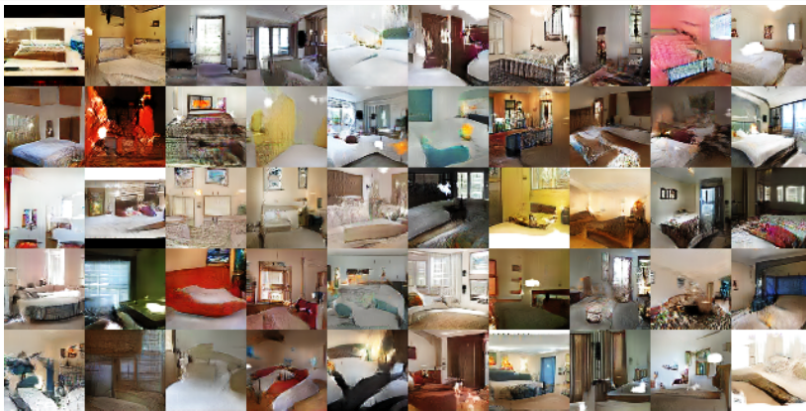
Image colorization



(Source: Richard Zhang, Phillip Isola and Alexei A. Efros, 2016)

Goal: to add color to grayscale photographs.

Image generation



Generated images of bedrooms (Source: Alec Radford, Luke Metz, Soumith Chintala, 2015)

Goal: to automatically create realistic pictures of a given category.

Image stylization

Synthesized Image

#NeuralDoodle



(Source: Neural Doodle, Champanard, 2016)

Goal: to create stylized images from rough sketches.

Style transfer



(Source: Gatys, Ecker and Bethge, 2015)

Goal: transfer the style of an image into another one.

Learning from examples

Learning from examples

3 main ingredients

- ① Training set / examples:

$$\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$$

- ② Machine or model:

$$\mathbf{x} \rightarrow \underbrace{f(\mathbf{x}; \theta)}_{\text{function / algorithm}} \rightarrow \underbrace{\mathbf{y}}_{\text{prediction}}$$

θ : parameters of the model

- ③ Loss, cost, objective function / energy:

$$\operatorname{argmin}_{\theta} E(\theta; \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$$

Learning from examples

Tools: $\left\{ \begin{array}{ll} \text{Data} & \leftrightarrow \text{Statistics} \\ \text{Loss} & \leftrightarrow \text{Optimization} \end{array} \right.$

Goal: to extract information from the training set

- relevant for the given task,
- relevant for other data of the same kind.

Terminology

Sample (Observation or Data): item to process (e.g., classify). *Example: an individual, a document, a picture, a sound, a video. . .*

Features (Input): set of distinct traits that can be used to describe each sample in a quantitative manner. Represented as a multi-dimensional vector usually denoted by x . *Example: size, weight, citizenship, . . .*

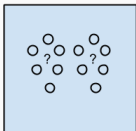
Training set: Set of data used to discover potentially predictive relationships.

Validation set: Set used to adjust the model hyperparameters (use as feedback).

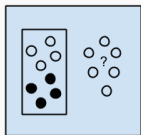
Testing set: Set used to assess the performance of a model (no feedback).

Label (Output): The class or outcome assigned to a sample. The actual prediction is often denoted by y and the desired/targeted class by d or t .

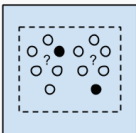
Learning approaches



Unsupervised Learning Algorithms



Supervised Learning Algorithms



Semi-supervised Learning Algorithms

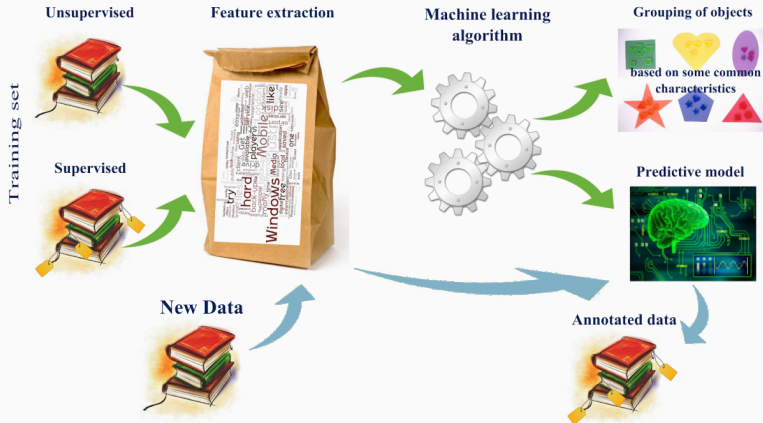
Unsupervised learning: Discovering patterns in unlabeled data. *Example: cluster similar documents based on the text content.*

Supervised learning: Learning with a labeled training set. *Example: email spam detector with training set of already labeled emails.*

Semisupervised learning: Learning with a small amount of labeled data and a large amount of unlabeled data. *Example: web content and protein sequence classifications.*

Reinforcement learning: Learning based on feedback or reward. *Example: learn to play chess by winning or losing.*

Machine learning workflow

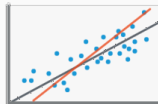


(Source: Michael Walker)

Problem types



Classification
(supervised – predictive)



Regression
(supervised – predictive)



Clustering
(unsupervised – descriptive)



Anomaly Detection
(unsupervised – descriptive)

(Source: Lucas Masuch)

What is deep learning?

- Part of the machine learning field of learning representations of data. Exceptionally effective at learning patterns.
- Utilizes learning algorithms that derive meaning out of data by using a hierarchy of multiple layers that mimic the neural networks of our brain.
- If you provide the system tons of information, it begins to understand it and respond in useful ways.
- Rebirth of artificial neural networks.

(Source: Lucas Masuch)

Deep learning: Academic actors

- Popularized by Hinton in 2006 with Restricted Boltzmann Machines



Geoffrey Hinton: University of Toronto & Google

- Developed by different actors:



Yann LeCun: New York University & Facebook



Andrew Ng: Stanford & Baidu



Yoshua Bengio: University of Montreal



Jürgen Schmidhuber: Swiss AI Lab & NNAISENSE

and many others...

Deep learning: Academic actors

- Popularized by Hinton in 2006 with Restricted Boltzmann Machines



Geoffrey Hinton: University of Toronto & Google

- Developed by different actors:



Yann LeCun: New York University & Facebook



Andrew Ng: Stanford & Baidu



Yoshua Bengio: University of Montreal



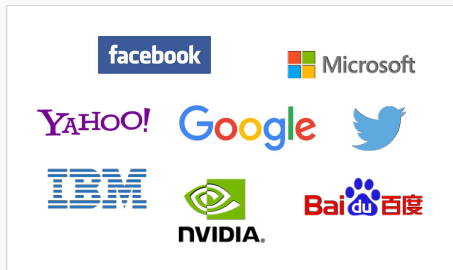
Jürgen Schmidhuber: Swiss AI Lab & NNAISENSE

and many others...

- Yoshua Bengio, Geoffrey Hinton, and Yann LeCun recipients of the 2018 ACM A.M. Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

Actors and applications

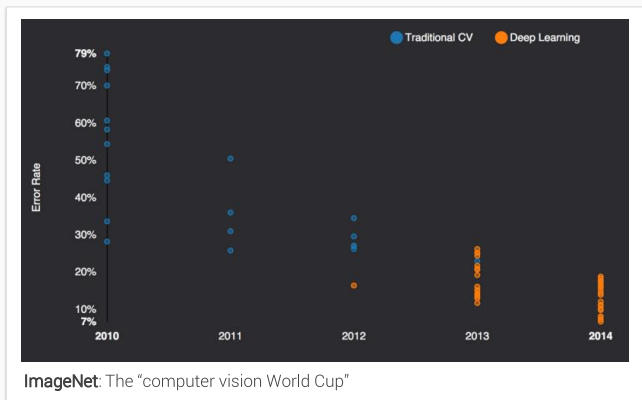
- Very active technology adopted by big actors



- Success story for many different academic problems
 - Image processing
 - Computer vision
 - Speech recognition
 - Natural language processing
 - Translation
 - etc
- Today all industries wonder if DL can improve their process.

Success stories

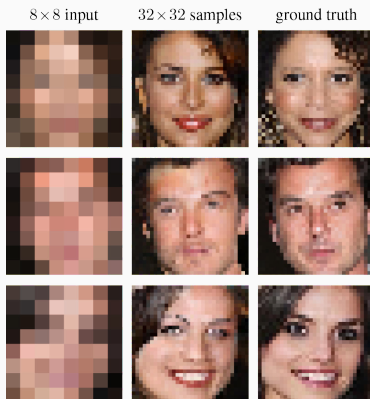
AlexNet wins the ImageNet classification challenge (Krizhevsky, 2012)



Since, Deep Learning has been the standard and achieved remarkable results.

Success stories

Google Brains's image super-resolution (Dahl *et al.*, 2017).



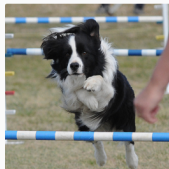
"Google's neural networks have achieved the dream of CSI viewers", The Guardian.

Success stories

Image captioning (Karpathy, Fei-Fei, CVPR, 2015).



"girl in pink dress is jumping in air."



"black and white dog jumps over bar."



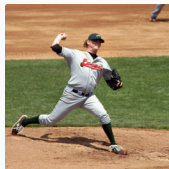
"young girl in pink shirt is swinging on swing."



"man in blue wetsuit is surfing on wave."



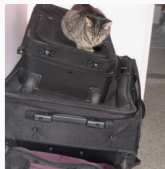
"little girl is eating piece of cake."



"baseball player is throwing ball in game."



"woman is holding bunch of bananas."

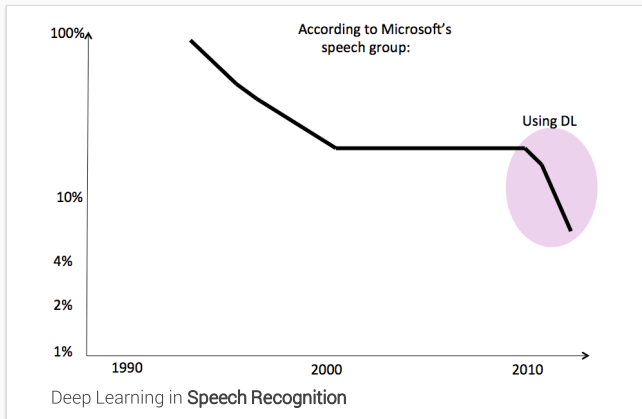


"black cat is sitting on top of suitcase."

"I had never thought AI would arrive at this stage that fast", D. J. Lasiman.

Success stories

Speech recognition



Same improvements in other fields: automatic translation, Go, ...

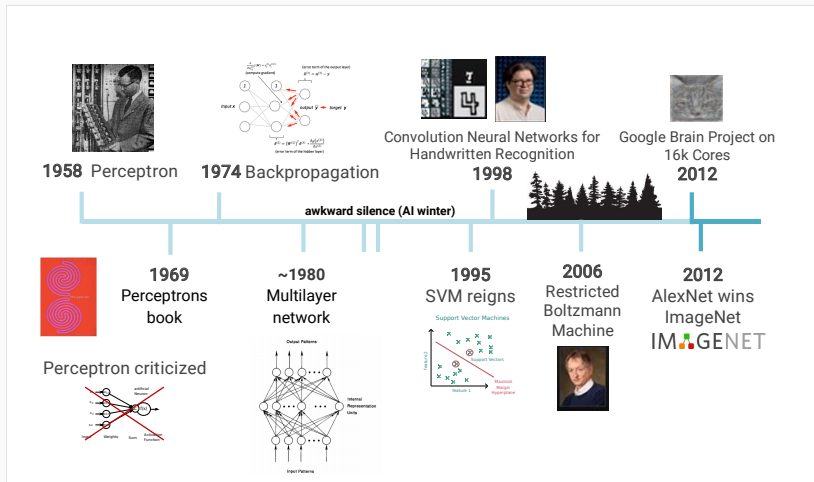
Success stories

Face generation (Style GAN, (Karras et al., 2018) (NVIDIA)):

These persons do not exist.



Timeline of (deep) learning



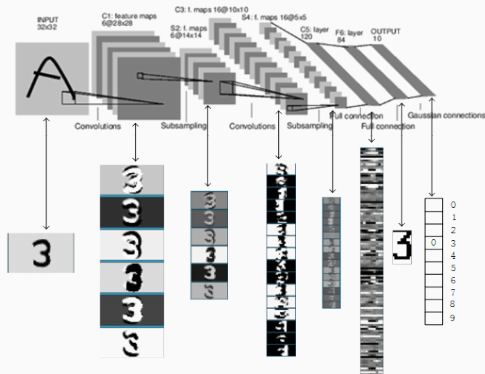
- ➊ Introduction to neural networks
- ➋ Multi-class classification using logistic loss
- ➌ Convolutional neural networks for image classification

Software: Python + PyTorch

Remark: Focus on image processing and computer vision, but deep learning works for many other applications:

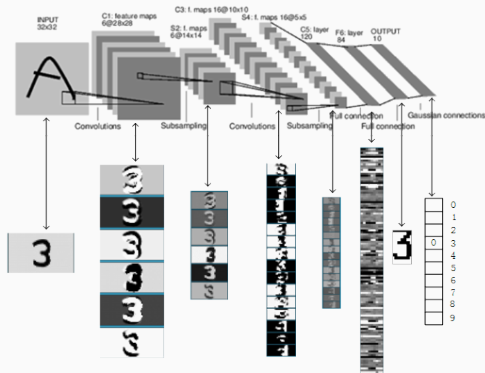
- Signal processing, speech recognition,...
- Text processing
- Graph processing (discrete geometry, social networks,...)
- Physics, chemistry,...

Neural networks for image classification



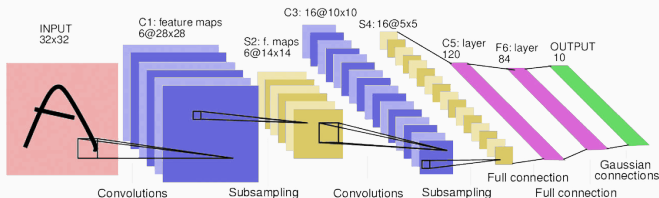
- **Goal:** Train a convolutional neural network for image classification

Neural networks for image classification

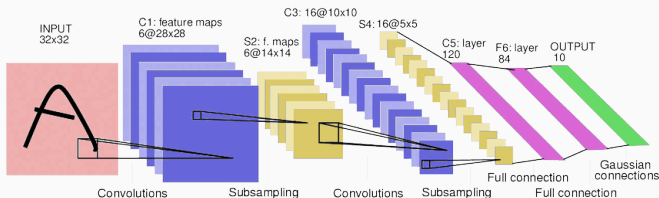


- **Goal:** Train a convolutional neural network for image classification
- **Goal:** Understand the training of a convolutional neural network for image classification

Understand the training of a convolutional neural network for image classification: A lot of notions: going backwards...

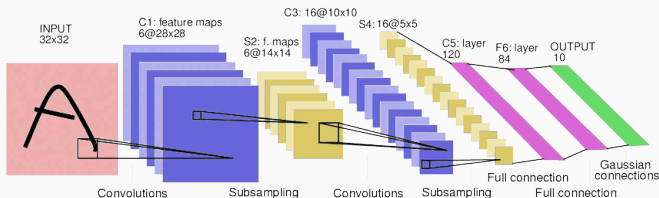


Understand the training of a convolutional neural network for image classification: A lot of notions: going backwards...



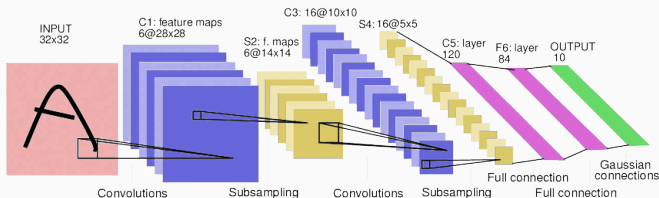
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.

Understand the training of a convolutional neural network for image classification: A lot of notions: going backwards...



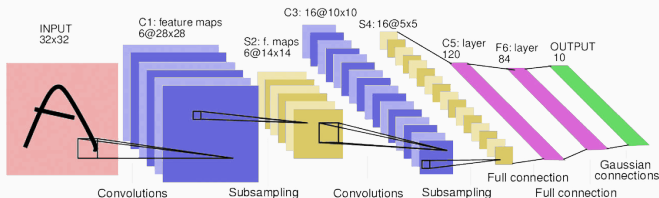
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.

Understand the training of a convolutional neural network for image classification: A lot of notions: going backwards...



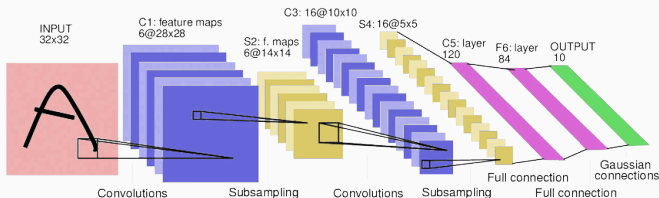
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done by optimizing a **classification loss** $L(W)$ on a training dataset: Typically this is a **linear classifier using cross-entropy**.

Understand the training of a convolutional neural network for image classification: A lot of notions: going backwards...



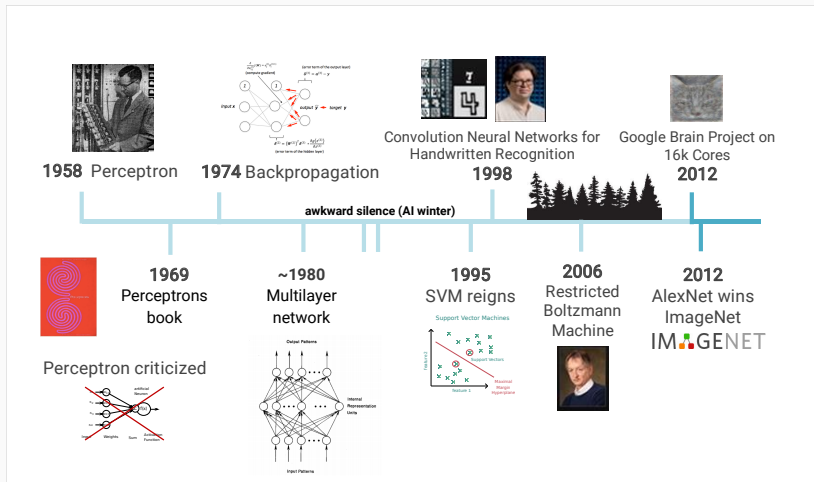
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done by optimizing a **classification loss** $L(W)$ on a training dataset: Typically this is a **linear classifier using cross-entropy**.
- The optimization of the classification loss is done using **stochastic gradient descent** on **batches of training data**.

Understand the training of a convolutional neural network for image classification: A lot of notions: going backwards...

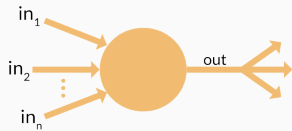
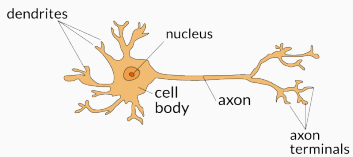


- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done by optimizing a **classification loss** $L(W)$ on a training dataset: Typically this is a **linear classifier using cross-entropy**.
- The optimization of the classification loss is done using **stochastic gradient descent** on **batches of training data**.
- The gradient $\nabla L(W)$ is computed using **backpropagation**.

Timeline of (deep) learning



Perceptron



Perceptron



1958 Perceptron

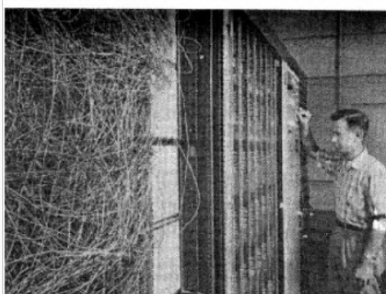
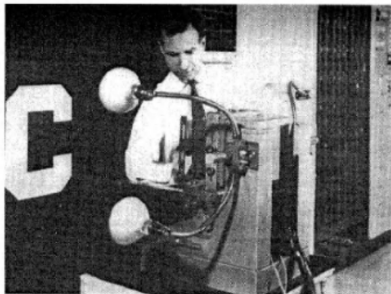


1969
Perceptrons
book

Perceptron criticized



Perceptron (Frank Rosenblatt, 1958)

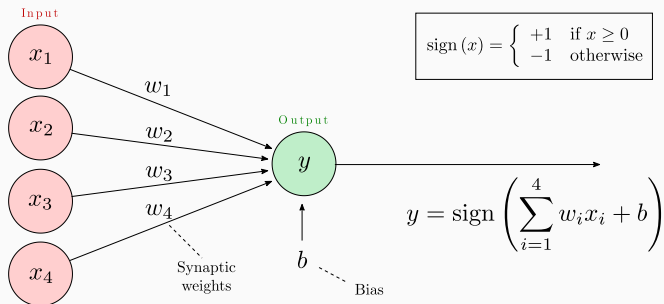


First binary classifier based on supervised learning (discrimination).

Foundation of modern artificial neural networks.

At that time: technological, scientific and philosophical challenges.

Representation of the Perceptron



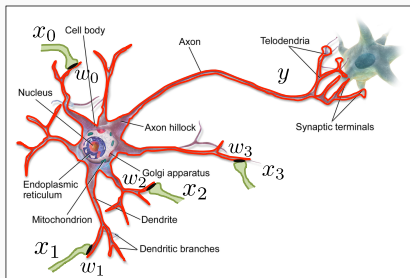
Parameters of the perceptron

- w_k : synaptic weights
 - b : bias
- } $\leftarrow$ real parameters to be estimated.

Training = adjusting the weights and biases

The origin of the Perceptron

Takes inspiration from the visual system known for its ability to learn patterns.



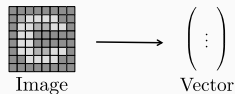
- When a neuron receives a stimulus with high enough voltage, it emits an **action potential** (aka, nerve impulse or spike). It is said to **fire**.
- The perceptron mimics this activation effect: it fires only when

$$\sum_i w_i x_i + b > 0$$

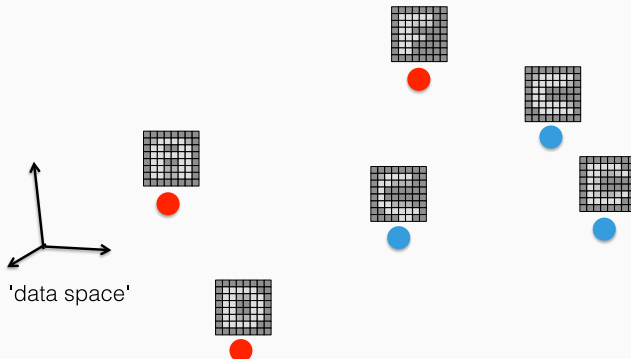
$$y = \underbrace{\text{sign}(w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + b)}_{f(\mathbf{x}; \mathbf{w})} = \begin{cases} +1 & \text{for the first class} \\ -1 & \text{for the second class} \end{cases}$$

Machine learning – Perceptron – Principle

- 1 Data are represented as vectors:



- 2 Collect training data with **positive** and **negative** examples:

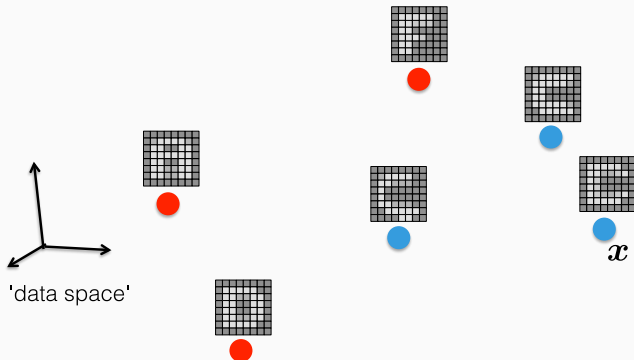


③ **Training:** find w and b so that:

- $\langle w, x \rangle + b$ is **positive** for **positive samples** x ,
- $\langle w, x \rangle + b$ is **negative** for **negative samples** x .

Dot product:

$$\begin{aligned}\langle w, x \rangle &= \sum_{i=1}^d w_i x_i \\ &= w^T x\end{aligned}$$



③ **Training:** find w and b so that:

- $\langle w, x \rangle + b$ is **positive** for **positive samples** x ,
- $\langle w, x \rangle + b$ is **negative** for **negative samples** x .

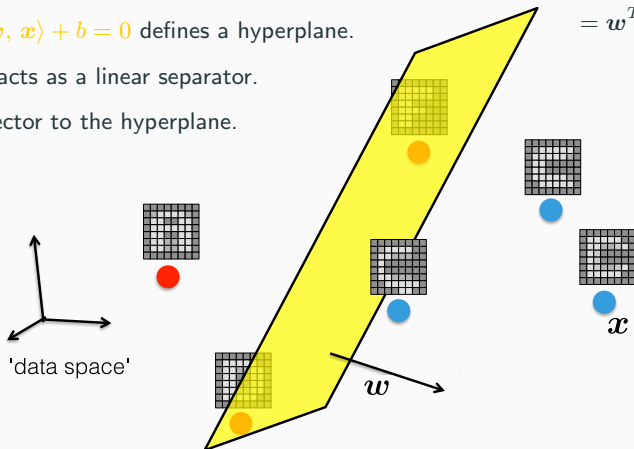
Dot product:

$$\begin{aligned}\langle w, x \rangle &= \sum_{i=1}^d w_i x_i \\ &= w^T x\end{aligned}$$

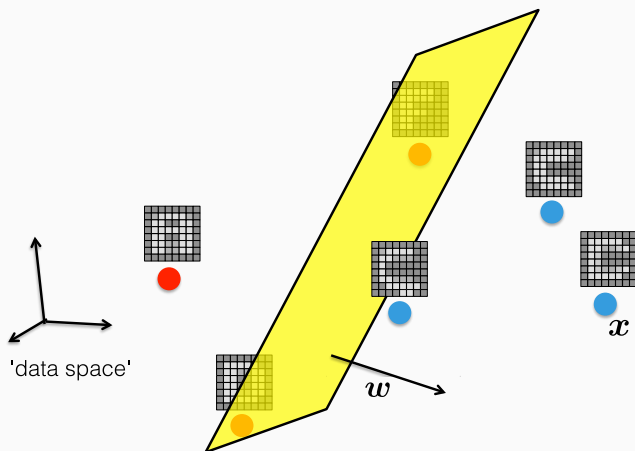
The equation $\langle w, x \rangle + b = 0$ defines a hyperplane.

The hyperplane acts as a linear separator.

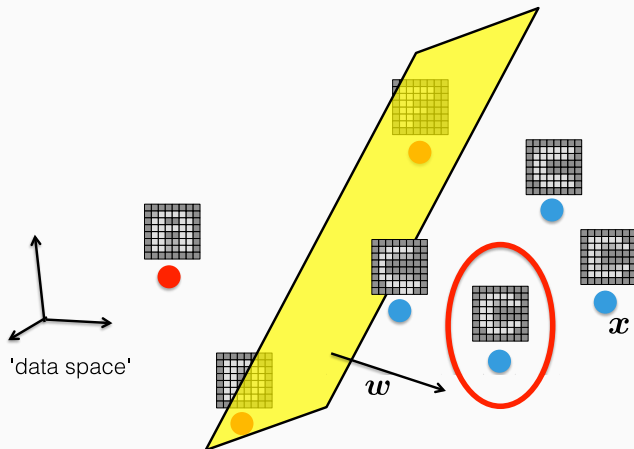
w is a normal vector to the hyperplane.



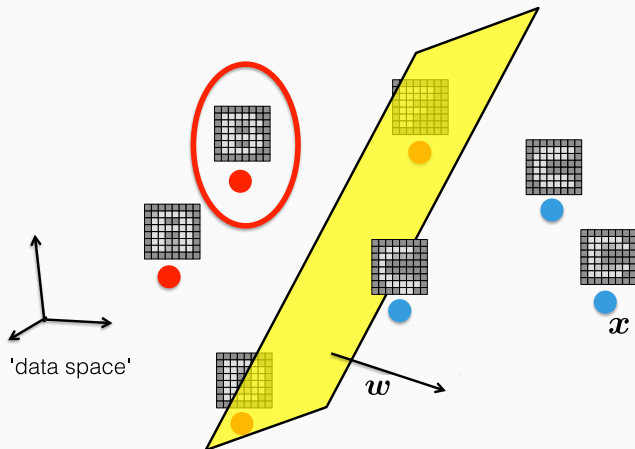
- ④ **Testing:** the perceptron can now classify new examples.



- ④ **Testing:** the perceptron can now classify new examples.
- A new example x is classified **positive** if $\langle w, x \rangle + b$ is **positive**,



- ④ **Testing:** the perceptron can now classify new examples.
- A new example x is classified **positive** if $\langle w, x \rangle + b$ is **positive**,
 - and **negative** if $\langle w, x \rangle + b$ is **negative**.

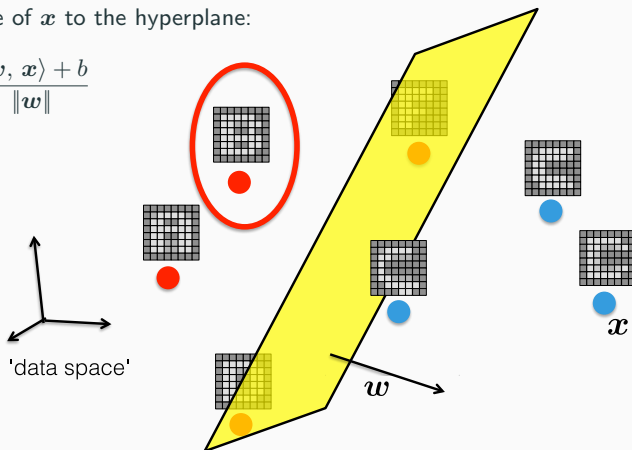


④ **Testing:** the perceptron can now classify new examples.

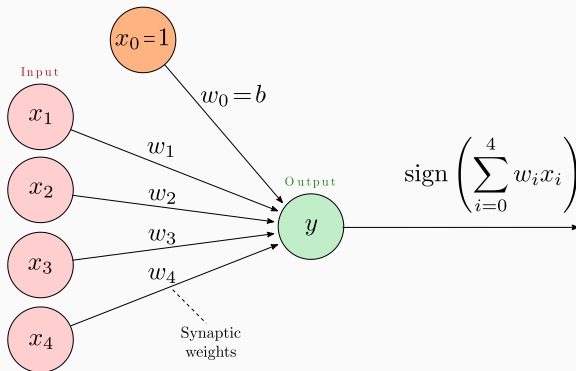
- A new example x is classified **positive** if $\langle w, x \rangle + b$ is **positive**,
- and **negative** if $\langle w, x \rangle + b$ is **negative**.

(signed) distance of x to the hyperplane:

$$r = \frac{\langle w, x \rangle + b}{\|w\|}$$



Alternative representation



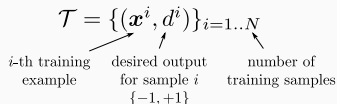
Use the zero-index to encode the bias as a synaptic weight.

Simplifies algorithms as all parameters can now be processed in the same way.

Perceptron algorithm

Goal: find the vector of weights $\mathbf{w}$ from a labeled training dataset $\mathcal{T}$

$$\mathcal{T} = \{(\mathbf{x}^i, d^i)\}_{i=1..N}$$



i -th training example desired output for sample i $\{-1, +1\}$ number of training samples

How: minimize classification errors

$$\min_{\mathbf{w}} E(\mathbf{w}) = - \sum_{\substack{(\mathbf{x}, d) \in \mathcal{T} \\ \text{st } y \neq d}} d \times \langle \mathbf{w}, \mathbf{x} \rangle = \sum_{(\mathbf{x}, d) \in \mathcal{T}} \max(-d \times \langle \mathbf{w}, \mathbf{x} \rangle, 0)$$

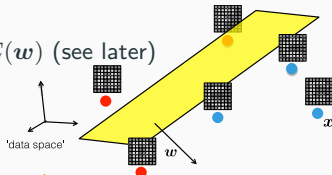
- penalize only misclassified samples ($y \neq d$) for which $d \times \langle \mathbf{w}, \mathbf{x} \rangle < 0$,
- zero if all samples are correctly classified.

Perceptron algorithm

- We assume that $\max(0, t)$ is derivable with derivative 1 if $t > 0$, 0 if $t \leq 0$.

Algorithm: (stochastic) gradient descent for $E(w)$ (see later)

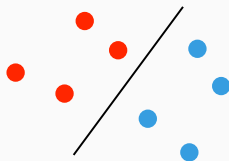
- Initialize w randomly
- Repeat until convergence
 - For all $(x, d) \in \mathcal{T}$ (or a random subset $\mathcal{T}' \subset \mathcal{T}$)
 - Compute: $y = \text{sign} \langle w, x \rangle$
 - If $y \neq d$:
Update: $w \leftarrow w + \gamma d x$



- Converges to some solution if the training data are linearly separable,
- But may pick any of many solutions of varying quality.
 $\Rightarrow$ Poor generalization error, compared with SVM and logistic loss.

Perceptrons book (Minsky and Papert, 1969)

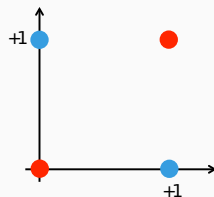
A perceptron can only classify data points that are linearly separable:



Linearly separable



Nonlinearly separable



The xor function

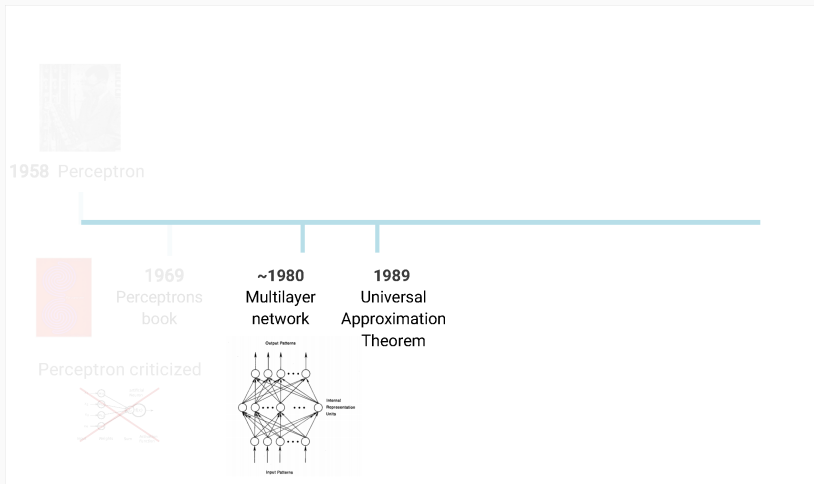
Seen by many as a justification to stop research on perceptrons.

(Source: Vincent Lepetit)

Artificial neural network



Artificial neural network

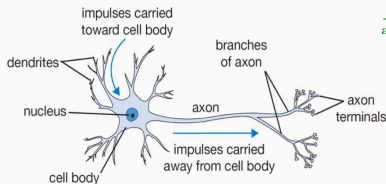


Artificial neural network

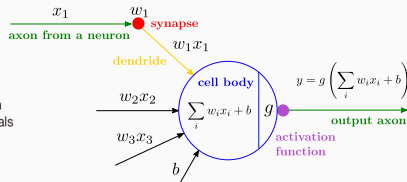


- Supervised learning method initially inspired by the behavior of the human brain.
- Consists of the inter-connection of several small units (just like in the human brain).
- Introduced in the late 50s, very popular in the 90s, reappeared in the 2010s with deep learning.
- Also referred to as **Multi-Layer Perceptron** (MLP).
- Historically used after feature extraction.

Artificial neuron (McCulloch & Pitts, 1943)



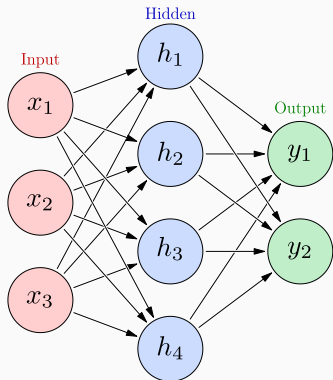
Biological neuron



Artificial neuron

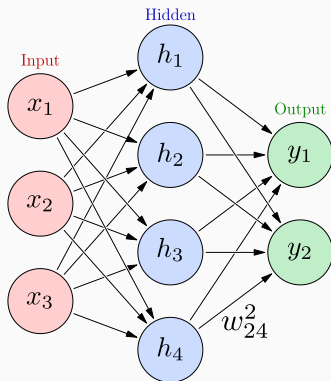
- An artificial neuron contains several incoming **weighted connections**, an outgoing connection and has a **nonlinear activation function** g .
- Neurons are **trained to filter and detect specific features** or patterns (e.g. edge, nose) by receiving weighted input, transforming it with the activation function and passing it to the outgoing connections.
- Unlike the perceptron, can be used for regression (with proper choice of g).

Artificial neural network / Multilayer perceptron / NeuralNet



- Inter-connection of several artificial neurons (also called nodes or units).
- Each level in the graph is called a layer:
 - Input layer,
 - Hidden layer(s),
 - Output layer.
- Each neuron in the hidden layers acts as a classifier / feature detector.
- Feedforward NN (no cycle)
 - first and simplest type of NN,
 - information moves in one direction.
- Recurrent NN (with cycle)
 - used for time sequences,
 - such as speech-recognition.

Artificial neural network / Multilayer perceptron / NeuralNet



$$h_1 = g_1 (w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1)$$

$$h_2 = g_1 (w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1)$$

$$h_3 = g_1 (w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1)$$

$$h_4 = g_1 (w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1)$$

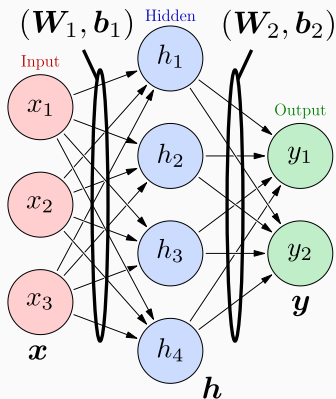
$$y_1 = g_2 (w_{11}^2 h_1 + w_{12}^2 h_2 + w_{13}^2 h_3 + w_{14}^2 h_4 + b_1^2)$$

$$y_2 = g_2 (w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2)$$

w_{ij}^k synaptic weight between previous node j and next node i at layer k .

g_k are any activation function applied to each coefficient of its input vector.

Artificial neural network / Multilayer perceptron / NeuralNet



$$h_1 = g_1 (w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1)$$

$$h_2 = g_1 (w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1)$$

$$h_3 = g_1 (w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1)$$

$$h_4 = g_1 (w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1)$$

$$\mathbf{h} = g_1 (\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$y_1 = g_2 (w_{11}^2 h_1 + w_{12}^2 h_2 + w_{13}^2 h_3 + w_{14}^2 h_4 + b_1^2)$$

$$y_2 = g_2 (w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2)$$

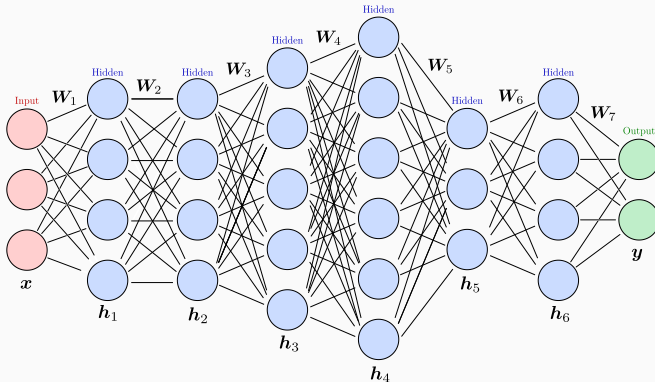
$$\mathbf{y} = g_2 (\mathbf{W}_2 \mathbf{h} + \mathbf{b}_2)$$

w_{ij}^k synaptic weight between previous node j and next node i at layer k .

g_k are any activation function applied to each coefficient of its input vector.

The matrices $\mathbf{W}_k$ and biases $\mathbf{b}_k$ are learned from labeled training data.

Artificial neural network / Multilayer perceptron



It can have 1 hidden layer only (**shallow network**),
It can have more than 1 hidden layer (**deep network**),
each layer may have a different size, and
hidden and output layers often have different activation functions.

Artificial neural network / Multilayer perceptron

- As for the perceptron, the biases can be integrated into the weights:

$$\mathbf{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k = \underbrace{\begin{pmatrix} \mathbf{b}_k & \mathbf{W}_k \end{pmatrix}}_{\tilde{\mathbf{W}}_k} \underbrace{\begin{pmatrix} 1 \\ \mathbf{h}_{k-1} \end{pmatrix}}_{\tilde{\mathbf{h}}_{k-1}} = \tilde{\mathbf{W}}_k \tilde{\mathbf{h}}_{k-1}$$

- A neural network with L layers is a function of $\mathbf{x}$ parameterized by $\tilde{\mathbf{W}}$:

$$\mathbf{y} = f(\mathbf{x}; \tilde{\mathbf{W}}) \quad \text{where} \quad \tilde{\mathbf{W}} = (\tilde{\mathbf{W}}_1, \tilde{\mathbf{W}}_2, \dots, \tilde{\mathbf{W}}_L)^T$$

- It can be defined recursively as

$$\mathbf{y} = f(\mathbf{x}; \tilde{\mathbf{W}}) = \mathbf{h}_L, \quad \mathbf{h}_k = g_k \left(\tilde{\mathbf{W}}_k \tilde{\mathbf{h}}_{k-1} \right) \quad \text{and} \quad \mathbf{h}_0 = \mathbf{x}$$

- For simplicity, $\tilde{\mathbf{W}}$ will be denoted $\mathbf{W}$ (when no possible confusions).

Activation functions

Linear units: $g(a) = a$

$$\mathbf{y} = \mathbf{W}_L \mathbf{h}_{L-1} + \mathbf{b}_L$$

$$\mathbf{h}_{L-1} = \mathbf{W}_{L-1} \mathbf{h}_{L-2} + \mathbf{b}_{L-1}$$

$$\mathbf{y} = \mathbf{W}_L \mathbf{W}_{L-1} \mathbf{h}_{L-2} + \mathbf{W}_L \mathbf{b}_{L-1} + \mathbf{b}_L$$

$$\mathbf{y} = \mathbf{W}_L \dots \mathbf{W}_1 \mathbf{x} + \sum_{k=1}^{L-1} \mathbf{W}_L \dots \mathbf{W}_{k+1} \mathbf{b}_k + \mathbf{b}_L$$

We can always find an equivalent network without hidden units,
because compositions of affine functions are affine.

In general, **non-linearity** is needed to learn complex (non-linear) representations of data, otherwise the NN would be just a linear function. Otherwise, back to the problem of nonlinearly separable datasets.

Activation functions

Threshold units: for instance the sign function

$$g(a) = \begin{cases} -1 & \text{if } a < 0 \\ +1 & \text{otherwise.} \end{cases}$$

or Heaviside (aka, step) activation functions

$$g(a) = \begin{cases} 0 & \text{if } a < 0 \\ 1 & \text{otherwise.} \end{cases}$$

Discontinuities in the hidden layers
make the optimization really difficult.

We prefer functions that are continuous and differentiable.

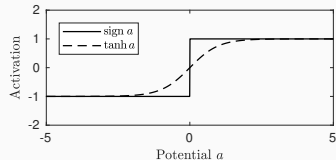
Activation functions

Sigmoidal units: for instance the hyperbolic tangent function

$$g(a) = \tanh a = \frac{e^a - e^{-a}}{e^a + e^{-a}} \in [-1, 1]$$

or the logistic sigmoid function

$$g(a) = \frac{1}{1 + e^{-a}} \in [0, 1]$$



- In fact equivalent by linear transformations :

$$\tanh(a/2) = 2\text{logistic}(a) - 1$$

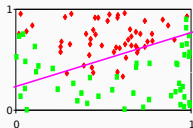
- Differentiable approximations of the sign and step functions, respectively.
- Act as threshold units for large values of $|a|$ and as linear for small values.

Sigmoidal units: logistic activation functions are used in binary classification (class C_1 vs C_2) as they can be **interpreted as posterior probabilities**:

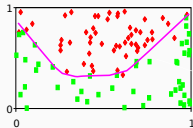
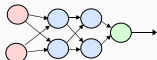
$$y = P(C_1|\mathbf{x}) \quad \text{and} \quad 1 - y = P(C_2|\mathbf{x})$$

The architecture of the network defines the shape of the separator

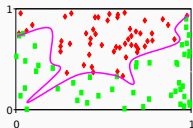
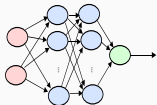
1 neuron



2+2+1 neurons



10+10+1 neurons



Separation
 $\{\mathbf{x} \text{ s.t. } P(C_1|\mathbf{x}) = P(C_2|\mathbf{x})\}$

Complexity/capacity of the
network

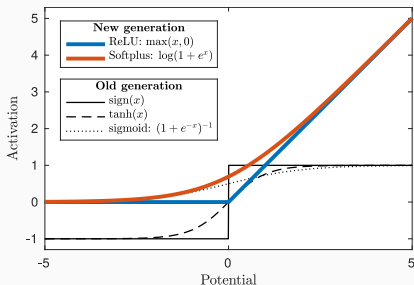
$\Rightarrow$

**Trade-off between
generalization and overfitting.**

Activation functions

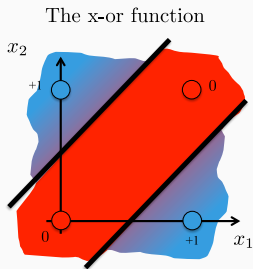
“Modern” units:

$$\underbrace{g(a) = \max(a, 0)}_{\text{ReLU}} \quad \text{or} \quad \underbrace{g(a) = \log(1 + e^a)}_{\text{Softplus}}$$



Most neural networks use **ReLU** (Rectifier linear unit) – $\max(a, 0)$ – nowadays for hidden layers, since it trains much faster, is more expressive than logistic function and prevents the gradient vanishing problem.

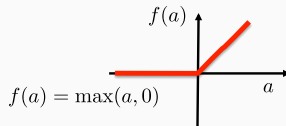
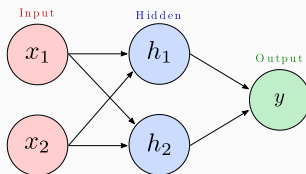
Neural networks solve non-linear separable problems



$$h = g(W_1 x + b_1)$$

$$y = \langle w_2, h \rangle + b_2$$

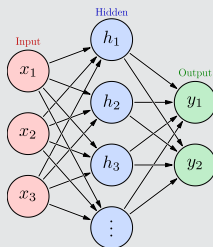
$$W_1 = \begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix}, b_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, w_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, b_2 = 0$$



Universal Approximation Theorem

(Hornik et al, 1989; Cybenko, 1989)

Any continuous function $f : K \subset \mathbb{R}^N \rightarrow \mathbb{R}^K$ can be uniformly approximated by a feedforward **shallow network** (i.e., with 1-hidden layer only) with a sufficient number of neurons in the hidden layer.



$$h = g(W_1x + b_1)$$
$$y = W_2h + b_2$$

- Works if and only if g is not polynomial (and thus non linear).
- The theorem does not say how large the network needs to be.
- No guarantee that the training algorithm will be able to train the network.

Tasks, architectures and loss functions



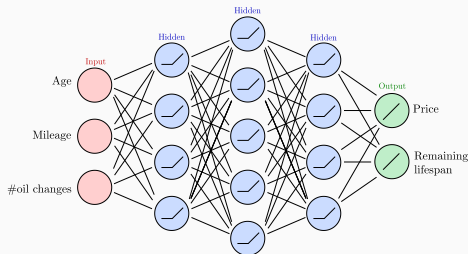
Approximation – Least square regression

- **Goal:** Predict a **real multivariate function**.
- **How:** estimate the coefficients $\mathbf{W}$ of $\mathbf{y} = f(\mathbf{x}; \mathbf{W})$ from labeled training examples where labels are real vectors:

$$\mathcal{T} = \{(\mathbf{x}^i, \mathbf{d}^i)\}_{i=1..N}$$

$\swarrow$ $\uparrow$ $\searrow$
 i -th training example desired output for sample i number of training samples

- **Typical architecture:**



- **Hidden layer:**

$$\text{ReLU}(a) = \max(a, 0)$$

- **Linear output:**

$$g(a) = a$$

Approximation – Least square regression

- **Loss:** As for the polynomial curve fitting, it is standard to consider the sum of square errors (assumption of Gaussian distributed errors)

$$E(\mathbf{W}) = \sum_{i=1}^N \|\mathbf{y}^i - \mathbf{d}^i\|_2^2 = \sum_{i=1}^N \|f(\mathbf{x}^i; \mathbf{W}) - \mathbf{d}^i\|_2^2$$

and look for $\mathbf{W}^*$ such that $\nabla E(\mathbf{W}^*) = 0$.

- **Solution:** Provided the network has enough flexibility and the size of the training set grows to infinity

$$\mathbf{y}^* = f(\mathbf{x}; \mathbf{W}^*) = \underbrace{\mathbb{E}[\mathbf{d}|\mathbf{x}] = \int \mathbf{d} p(\mathbf{d}|\mathbf{x}) d\mathbf{d}}_{\text{posterior mean}}$$

Multiclass classification – Multivariate logistic regression

(aka, multinomial classification)

- **Goal:** Classify an object $\mathbf{x}$ into **one among K classes** $C_1, \dots, C_K$.
- **How:** Estimate the coefficients $\mathbf{W}$ of a multivariate function

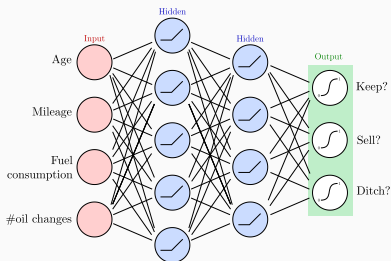
$$\mathbf{y} = f(\mathbf{x}; \mathbf{W}) \in [0, 1]^K \quad \text{s.t.} \quad \sum_{k=1}^K y_k = 1.$$

from training examples $\mathcal{T} = \{(\mathbf{x}^i, \mathbf{d}^i)\}$ where $\mathbf{d}^i$ is a 1-of-K (one-hot) code

- Class 1: $\mathbf{d}^i = (1, 0, \dots, 0)^T$ if $\mathbf{x}^i \in C_1$
 - Class 2: $\mathbf{d}^i = (0, 1, \dots, 0)^T$ if $\mathbf{x}^i \in C_2$
 - ...
 - Class K: $\mathbf{d}^i = (0, 0, \dots, 1)^T$ if $\mathbf{x}^i \in C_K$
- $y_k = f(\mathbf{x}; \mathbf{W})$ is understood as the probability of $\mathbf{x} \in C_k$.
 - **Remark:** Do not use the class index k directly as a scalar label: The order of label is not informative.

Multiclass classification – Multivariate logistic regression

- Typical architecture:



- Hidden layer:

$$\text{ReLU}(a) = \max(a, 0)$$

- Output layer:

$$\text{softmax}(\mathbf{a})_k = \frac{\exp(a_k)}{\sum_{\ell=1}^K \exp(a_\ell)}$$

- Softmax maps $\mathbb{R}^K$ to the set of probability vectors $\{\mathbf{y} \in (0, 1)^K, \sum_{k=1}^K \mathbf{y}_k = 1\}$.
- Smooth version of winner-takes-all activation model (maxout).
- The final decision function is winner-takes-all

$$\text{argmax}_k \text{softmax}(\mathbf{a}) = \text{argmax}_k \mathbf{a}$$

Multiclass classification – Multivariate logistic regression

- **Loss:** it is standard to consider the **cross-entropy** for K classes (assumption of multinomial distributed data)

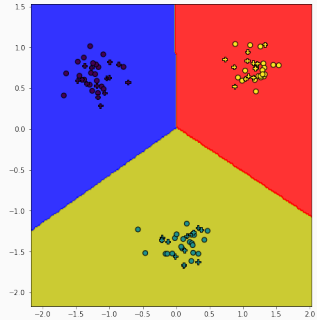
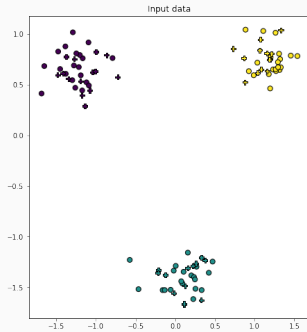
$$\begin{aligned} E(\mathbf{W}) &= - \sum_{i=1}^N \sum_{k=1}^K d_k^i \log y_k^i \quad \text{with} \quad \mathbf{y}^i = f(\mathbf{x}^i; \mathbf{W}) = \text{softmax}(\mathbf{a}) \in (0, 1)^K. \\ &= - \sum_{i=1}^N \left[a_{d^i} - \log \left(\sum_{k=1}^K \exp(a_k) \right) \right] \quad \text{with } d^i \text{ the class of } \mathbf{x}^i. \end{aligned}$$

and look for $\mathbf{W}^*$ such that $\nabla E(\mathbf{W}^*) = 0$.

- **Solution:** Provided the network has enough flexibility and the size of the training set grows to infinity

$$y_k^* = f_k(\mathbf{x}; \mathbf{W}^*) = \underbrace{\mathbb{P}(C_k | \mathbf{x})}_{\text{posterior probability}}$$

Multivariate logistic regression

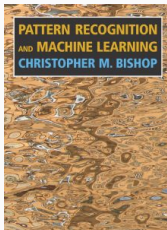


Multiclass classification – Multivariate logistic regression

- SVMs allow for multiclass classification but are not easily pluggable to neural networks.
- Instead neural networks generally use multivariate logistic regression.

Goal of this section:

- Mathematics of multivariate logistic regression.
- Reference: Section "4.3.4 Multiclass logistic regression" of C. M. Bishop, *Pattern Recognition and Machine Learning*, Information Science and Statistics, Springer, 2006



New notation

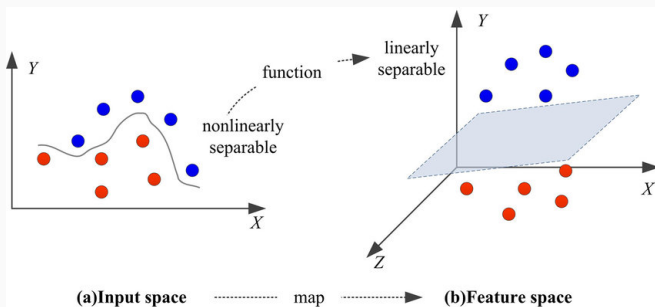
- **Goal:** Classify an object $\mathbf{x}$ into **one among K classes** $C_1, \dots, C_K$.
- Training set: $\mathcal{T} = \{(\mathbf{x}_n, t_n), n = 1, \dots, N\}$, $t_n \in \{1, \dots, K\}$ encodes the class of $\mathbf{x}_n$.
- Each t_n is transformed into a vector $\mathbf{t}_n \in \{0, 1\}^K$ with a 1-of- K code:
 - Class 1: $\mathbf{t}_n = (1, 0, \dots, 0)^T$ if $\mathbf{x}_n \in C_1$, i.e $t_n = 1$,
 - Class 2: $\mathbf{t}_n = (0, 1, \dots, 0)^T$ if $\mathbf{x}_n \in C_2$, i.e $t_n = 2$,
 - ...
 - Class K: $\mathbf{t}_n = (0, 0, \dots, 1)^T$ if $\mathbf{x}_n \in C_K$, i.e $t_n = K$,
- **Remark:** Do not use the class index k directly as a scalar label: The order of label is not informative.

Feature transform

- We apply a feature transform $\varphi : \mathbb{R}^p \rightarrow \mathbb{R}^D$ to each $\mathbf{x}_n$:

$$\varphi_n = \varphi(\mathbf{x}_n), \quad n = 1, \dots, N.$$

- Depending on context it allows to increase ($D > p$) or decrease ($D < p$) the dimension in a way to favor class discrimination (e.g. PCA...).
- This is a non linear map that should make the classes linearly separable.



Linear classifier

We will consider **linear multiclass classifier in feature space $\mathbb{R}^D$** :

- **Classifier parameters:** Each class k has a weight vector $\mathbf{w}_k \in \mathbb{R}^D$ and bias $b_k \in \mathbb{R}$
- Class separation between class k and class ℓ :

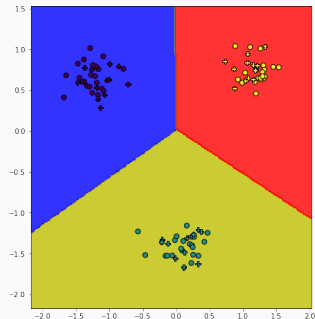
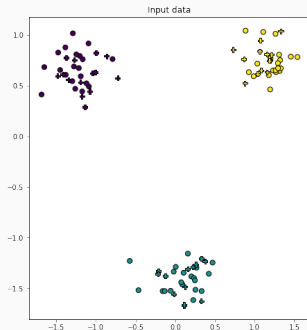
$$\mathbf{w}_k^T \varphi + b_k > \mathbf{w}_\ell^T \varphi + b_\ell? \Leftrightarrow (\mathbf{w}_k - \mathbf{w}_\ell)^T \varphi + b_k - b_\ell > 0?$$

- Class k region:

$$\begin{aligned} & \left\{ \varphi \in \mathbb{R}^D, \forall \ell \neq k, \mathbf{w}_k^T \varphi + b_k > \mathbf{w}_\ell^T \varphi + b_\ell \right\} \\ &= \bigcap_{\ell=1, \ell \neq k}^K \left\{ \varphi \in \mathbb{R}^D, \mathbf{w}_k^T \varphi + b_k > \mathbf{w}_\ell^T \varphi + b_\ell \right\} \\ &= \text{intersection of } K - 1 \text{ half-planes} \end{aligned}$$

- The classification partition is made of (unbounded) convex polygons (in feature space).

Multivariate logistic regression



Bias trick for linear classifier

- **Classifier parameters:** Each class k has a weight vector $\mathbf{w}_k \in \mathbb{R}^D$ and bias $b_k \in \mathbb{R}$
- Add an additional dummy coordinate 1 to φ so that

$$\mathbf{w}_k^T \varphi + b_k = \begin{pmatrix} \mathbf{w}_k \\ b_k \end{pmatrix}^T \begin{pmatrix} \varphi \\ 1 \end{pmatrix} = \tilde{\mathbf{w}}_k^T \tilde{\varphi}.$$

- **From now on this is implicit:** We assume that the feature transform has a 1 component so that $\mathbf{w}_k^T \varphi$ has an **implicit bias component**.
- The classifier parameters are just a set of weight vectors $\{\mathbf{w}_k, k = 1, \dots, K\}$.

Bias trick for linear classifier

- **Classifier parameters:** Each class k has a weight vector $\mathbf{w}_k \in \mathbb{R}^D$ and bias $b_k \in \mathbb{R}$
- Add an additional dummy coordinate 1 to φ so that

$$\mathbf{w}_k^T \varphi + b_k = \begin{pmatrix} \mathbf{w}_k \\ b_k \end{pmatrix}^T \begin{pmatrix} \varphi \\ 1 \end{pmatrix} = \tilde{\mathbf{w}}_k^T \tilde{\varphi}.$$

- **From now on this is implicit:** We assume that the feature transform has a 1 component so that $\mathbf{w}_k^T \varphi$ has an **implicit bias component**.
- The classifier parameters are just a set of weight vectors $\{\mathbf{w}_k, k = 1, \dots, K\}$.
- What are the boundaries if we forget the bias ?

Multivariate logistic regression

- After feature transform the training set is: $\mathcal{T} = \{(\varphi_n, t_n), n = 1, \dots, N\}$,
- We want to estimate

$$\mathbf{y} = f(\varphi) \in [0, 1]^K \quad \text{s.t.} \quad \sum_{k=1}^K y_k = 1.$$

such that ideally $\mathbf{y}_k \simeq P(C_k|\varphi)$ is an estimate of the **posterior probability**

$P(C_k|\varphi)$ = Probability of being in class C_k given feature vector φ .

- **Model assumption:** Posterior probabilities $P(C_k|\varphi)$ given the feature is a softmax transformation of a linear function of the feature variable:

There exists K vectors $\mathbf{w}_1, \dots, \mathbf{w}_K \in \mathbb{R}^D$ such that:

$$\mathbf{y}_k(\varphi) = P(C_k|\varphi) = \frac{\exp(\mathbf{w}_k^T \varphi)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi)}, \quad k = 1, \dots, K.$$

- By construction of the softmax, one has $\mathbf{y} \in (0, 1)^K$ s.t. $\sum_{k=1}^K y_k = 1$.

- **Model assumption:** There exists K vectors $\mathbf{w}_1, \dots, \mathbf{w}_K \in \mathbb{R}^D$ such that:

$$\mathbf{y}_k(\varphi) = P(C_k|\varphi) = \frac{\exp(\mathbf{w}_k^T \varphi)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi)}, \quad k = 1, \dots, K.$$

- We denote $\mathbf{W} = (\mathbf{w}_1, \dots, \mathbf{w}_K) \in \mathbb{R}^{D \times K}$ the matrix containing all the weights.

Training:

- Training = Find the best weight matrix $\mathbf{W}$ to explain the dataset.
- Performed using maximum **likelihood**.

Likelihood: Assume a multinomial model for the classes

- For each φ , associate the multinomial random variable $T(\varphi)$ that takes the value k with probability

$$P(C_k|\varphi) = \frac{\exp(\mathbf{w}_k^T \varphi)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi)}$$

- Each realization (φ_n, t_n) of the dataset are assumed independent.
- Then the likelihood of the training dataset is:

$$\begin{aligned} P((T(\varphi_1), \dots, T(\varphi_N) = (t_1, \dots, t_N)) &= \prod_{n=1}^N P(T(\varphi_n) = t_n) \\ &= \prod_{n=1}^N P(C_{t_n}|\varphi_n) \end{aligned}$$

Likelihood: Assume a multinomial model for the classes

- For each φ , associate the multinomial random variable $T(\varphi)$ that takes the value k with probability

$$P(C_k|\varphi) = \frac{\exp(\mathbf{w}_k^T \varphi)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi)}$$

- Each realization (φ_n, t_n) of the dataset are assumed independent.
- Then the likelihood of the training dataset is:

$$\begin{aligned} P((T(\varphi_1), \dots, T(\varphi_N) = (t_1, \dots, t_N)) &= \prod_{n=1}^N P(T(\varphi_n) = t_n) \\ &= \prod_{n=1}^N P(C_{t_n}|\varphi_n) \end{aligned}$$

- We want to **maximize the likelihood** with respect to $\mathbf{W} = (\mathbf{w}_1, \dots, \mathbf{w}_K) \in \mathbb{R}^{D \times K}$ the matrix containing all the weights.
- We minimize $L(\mathbf{W}) = -\log P$ instead (maximize the loglikelihood).

Loglikelihood:

- **Model assumption:** There exists K vectors $\mathbf{w}_1, \dots, \mathbf{w}_K \in \mathbb{R}^D$ such that:

$$\mathbf{y}_k(\varphi) = P(C_k|\varphi) = \frac{\exp(\mathbf{w}_k^T \varphi)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi)}, \quad k = 1, \dots, K.$$

- Expression of loglikelihood:

$$\begin{aligned} L(\mathbf{W}) &= -\log \prod_{n=1}^N P(C_{t_n}|\varphi_n) \\ &= -\sum_{n=1}^N \log \mathbf{y}_{t_n}(\varphi_n) \\ &= -\sum_{n=1}^N \mathbf{w}_{t_n}^T \varphi_n - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right) \end{aligned}$$

Loglikelihood:

- Expression of loglikelihood:

$$L(\mathbf{W}) = - \sum_{n=1}^N \log \mathbf{y}_{t_n}(\varphi_n) = - \sum_{n=1}^N \mathbf{w}_{t_n}^T \varphi_n - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right)$$

- Alternative expression with 1-of- K code: Recall that

$$\mathbf{t}_{n,k} = \begin{cases} 1 & \text{if } k = t_n, \\ 0 & \text{otherwise.} \end{cases} \quad \text{so that} \quad \mathbf{w}_{t_n} = \sum_{k=1}^K \mathbf{t}_{n,k} \mathbf{w}_k.$$

$$L(\mathbf{W}) = - \sum_{n=1}^N \sum_{j=1}^K \mathbf{t}_{n,j} \mathbf{w}_j^T \varphi_n - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right)$$

Loglikelihood:

- Expression of loglikelihood:

$$L(\mathbf{W}) = - \sum_{n=1}^N \log \mathbf{y}_{t_n}(\varphi_n) = - \sum_{n=1}^N \mathbf{w}_{t_n}^T \varphi_n - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right)$$

- Alternative expression with 1-of- K code: Recall that

$$\mathbf{t}_{n,k} = \begin{cases} 1 & \text{if } k = t_n, \\ 0 & \text{otherwise.} \end{cases} \quad \text{so that} \quad \mathbf{w}_{t_n} = \sum_{k=1}^K \mathbf{t}_{n,k} \mathbf{w}_k.$$

$$L(\mathbf{W}) = - \sum_{n=1}^N \sum_{j=1}^K \mathbf{t}_{n,j} \mathbf{w}_j^T \varphi_n - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right)$$

- One can show that $\mathbf{W} \mapsto L(\mathbf{W})$ is convex.
- What do we need to optimize $L(\mathbf{W})$?

Gradient of loglikelihood:

$$L(\mathbf{W}) = - \sum_{n=1}^N \sum_{j=1}^K \mathbf{t}_{n,j} \mathbf{w}_j^T \varphi_n - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right)$$

- Linear part: Partial gradient with respect to column $\mathbf{w}_\ell$, $\ell \in \{1, \dots, K\}$:

$$\nabla_{\mathbf{w}_\ell} \left[\sum_{j=1}^K \mathbf{t}_{n,j} \mathbf{w}_j^T \varphi_n \right] = \nabla_{\mathbf{w}_\ell} \left[\mathbf{t}_{n,\ell} \mathbf{w}_\ell^T \varphi_n + \text{constant} \right] = \mathbf{t}_{n,\ell} \varphi_n$$

- Partial gradient of $\nabla_{\mathbf{w}_\ell} \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right)$?

$$\begin{aligned} \nabla_{\mathbf{w}_\ell} \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right) &= \nabla_{\mathbf{w}_\ell} \log \left(\exp(\mathbf{w}_\ell^T \varphi_n) + \text{constant} \right) \\ &=? \end{aligned}$$

Multivariate logistic regression

Gradient of log-likelihood:

Recall that for $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$,

$$\nabla(g \circ f)(x) = g'(f(x))\nabla f(x).$$

Here,

$$g(t) = \log(\exp(t) + c) \quad g'(t) = \frac{\exp(t)}{\exp(t) + c}$$

$$f(\mathbf{w}_\ell) = \mathbf{w}_\ell^T \varphi_n \quad \nabla f(\mathbf{w}_\ell) = \varphi_n.$$

So,

$$\nabla_{\mathbf{w}_\ell} \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right) = \frac{\exp(\mathbf{w}_\ell^T \varphi_n)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n)} \varphi_n = \mathbf{y}_\ell(\varphi_n) \varphi_n$$

since

$$\mathbf{y}_k(\varphi) = \frac{\exp(\mathbf{w}_k^T \varphi)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi)}, \quad k = 1, \dots, K.$$

Gradient of log-likelihood:

$$L(\mathbf{W}) = - \sum_{n=1}^N \sum_{j=1}^K \mathbf{t}_{n,j} w_j^T \varphi_n - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right)$$

- For each column $\mathbf{w}_\ell \in \mathbb{R}^D$ of $\mathbf{W}$, $\ell \in \{1, \dots, K\}$,

$$\nabla_{\mathbf{w}_\ell} L(\mathbf{W}) = - \sum_{n=1}^N (\mathbf{t}_{n,\ell} - \mathbf{y}_\ell(\varphi_n)) \varphi_n = \sum_{n=1}^N (\mathbf{y}_\ell(\varphi_n) - \mathbf{t}_{n,\ell}) \varphi_n \in \mathbb{R}^D$$

Gradient of log-likelihood:

$$L(\mathbf{W}) = - \sum_{n=1}^N \sum_{j=1}^K \mathbf{t}_{n,j} w_j^T \varphi_n - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \varphi_n) \right)$$

- For each column $\mathbf{w}_\ell \in \mathbb{R}^D$ of $\mathbf{W}$, $\ell \in \{1, \dots, K\}$,

$$\nabla_{\mathbf{w}_\ell} L(\mathbf{W}) = - \sum_{n=1}^N (\mathbf{t}_{n,\ell} - \mathbf{y}_\ell(\varphi_n)) \varphi_n = \sum_{n=1}^N (\mathbf{y}_\ell(\varphi_n) - \mathbf{t}_{n,\ell}) \varphi_n \in \mathbb{R}^D$$

- Full gradient for $\mathbf{W}$:

$$\nabla L(\mathbf{W}) = \sum_{n=1}^N \varphi_n (\mathbf{y}(\varphi_n) - \mathbf{t}_n)^T \in \mathbb{R}^{D \times K}.$$

- OK with intuition ?

Optimization:

- We can apply the **gradient descent algorithm** to minimize L .

An iterative algorithm trying to find a minimum of a real function.

Gradient descent

- Let F be a real function, coercive, and twice-differentiable such that:

$$\underbrace{\|\nabla^2 F(x)\|_2}_{\text{Hessian matrix of } F} \leq L, \quad \text{for some } L > 0.$$

- Then, whatever the initialization x^0 , if $0 < \gamma < 2/L$, the sequence

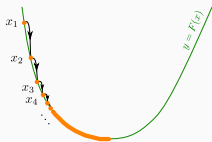
$$x^{(n+1)} = x^{(n)} - \underbrace{\gamma \nabla F(x^{(n)})}_{\text{direction of greatest descent}},$$

converges to a **stationary point** x^* (i.e., it cancels the gradient)

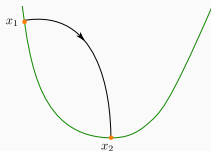
$$\nabla F(x^*) = 0.$$

- The parameter γ is called the step size (or **learning rate** in ML field).
- A too small step size γ leads to slow convergence.

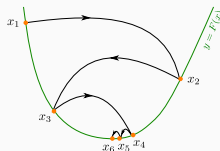
One dimension



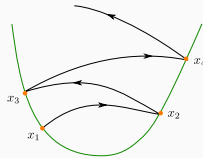
Small step size
Slow convergence



Good step size
Fast convergence

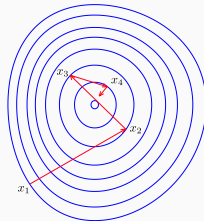
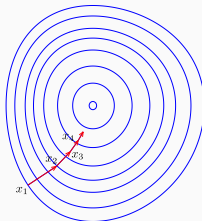
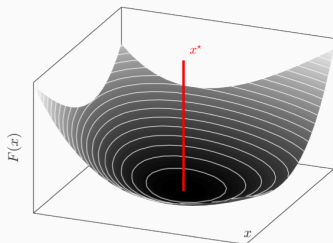


Large step size
Slow convergence



Too large step size
Divergence

Two dimensions



Multivariate logistic regression

Gradient of log-likelihood: $\nabla L(\mathbf{W}) = \sum_{n=1}^N \varphi_n(\mathbf{y}(\varphi_n) - \mathbf{t}_n)^T \in \mathbb{R}^{D \times K}.$

Optimization:

- Problem: In machine learning, the larger the dataset the better... but then more and more computation for the gradient.
- **Solution:** Use **(averaged) stochastic gradient descent**:
 - Draw randomly a small subset $\mathcal{S} \subset \mathcal{T}$ of the training set
 - Compute a noisy gradient with this small set only and update weights:

$$\mathbf{W}^{(n)} = \mathbf{W}^{(n-1)} - \gamma^{(n)} \nabla L(\mathbf{W}^{(n-1)}, \mathcal{S}) = \mathbf{W}^{(n-1)} - \gamma^{(n)} \sum_{n \in \mathcal{S}} \varphi_n(\mathbf{y}(\varphi_n) - \mathbf{t}_n)^T$$

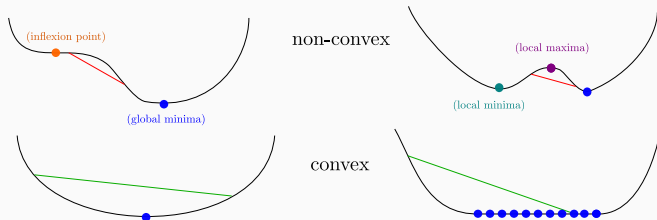
and compute **averaged weights**

$$\bar{\mathbf{W}}^{(n)} = \frac{1}{n+1} \sum_{k=0}^n \mathbf{W}^{(k)} = \frac{n}{n+1} \bar{\mathbf{W}}^{(n-1)} + \frac{1}{n+1} \mathbf{W}^{(n)}.$$

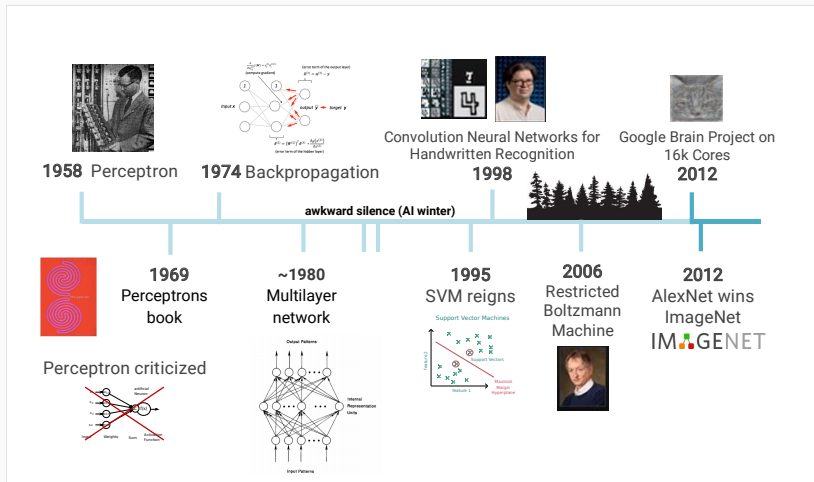
- Convergence results for L (strongly) convex if $\gamma^{(n)}$ decays well etc.

Non convexity in machine learning

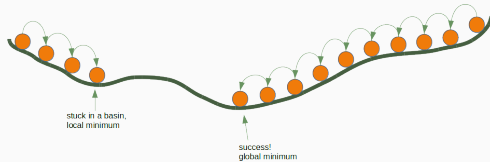
But for neural network the cost is **not convex**...



Timeline of (deep) learning



Backpropagation



Questions?

Next class: Backpropagation

Slides from Charles Deledalle

Sources, images courtesy and acknowledgment

K. Chatfield

P. Gallinari,

C. Hazırbaş

A. Horodniceanu

Y. LeCun

V. Lepetit

L. Masuch

A. Ng

M. Ranzato