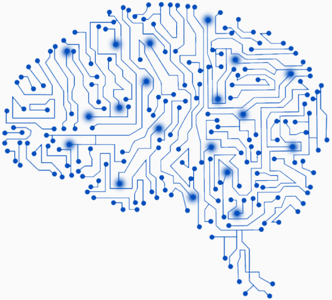
Ecole IoT: Data Science

Course IV – Introduction to Artificial Neural Networks for image classification

Bruno Galerne

Tuesday January 16, 2024



Credits

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

www.charles-deledalle.fr/

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

Computer Vision and Machine Learning

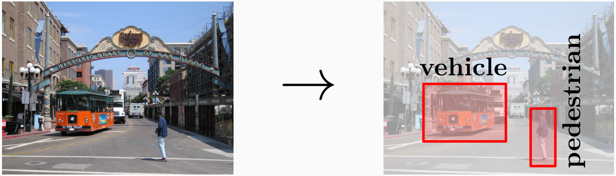


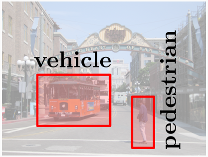
Image $\longrightarrow$ Symbols

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.

Image



Symbols

CV is a subfield of Artificial Intelligence.

Definition (Oxford dictionary)

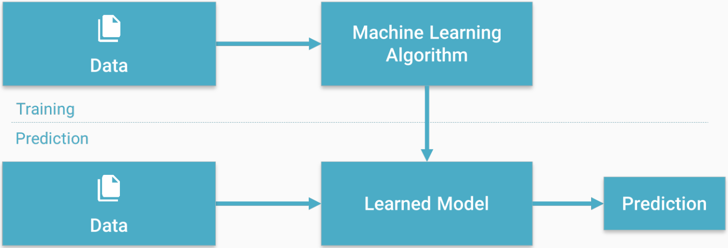
Artificial Intelligence, *noun*: the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation.

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.

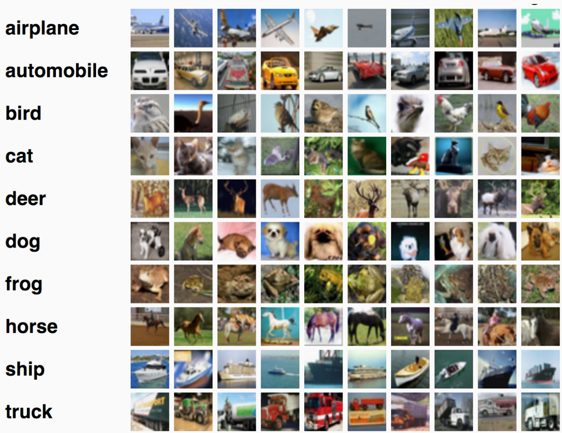
Definition

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:

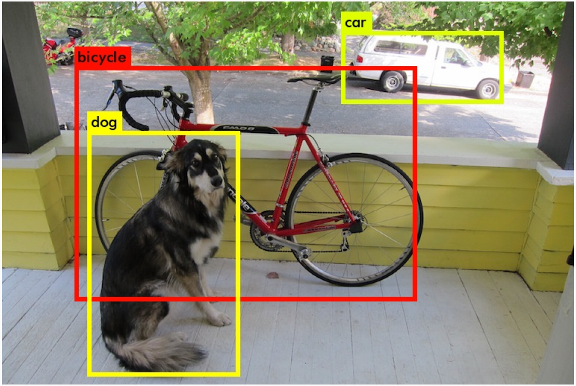


Computer vision – Image classification



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."

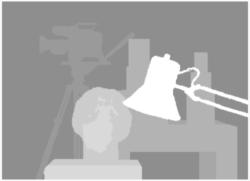
(Karpathy, Fei-Fei, CVPR, 2015)

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

Computer vision – Depth estimation

→

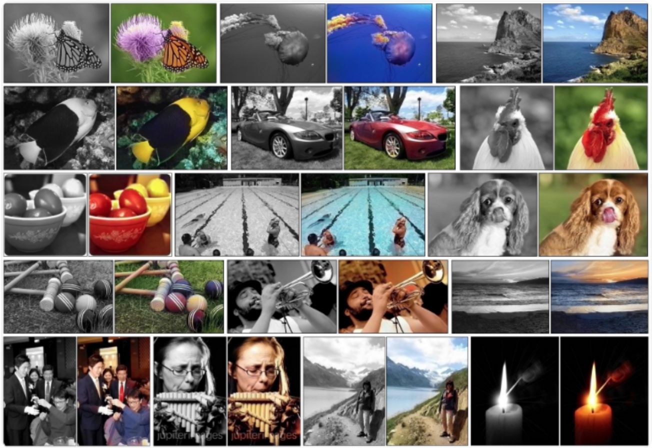
→



(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)

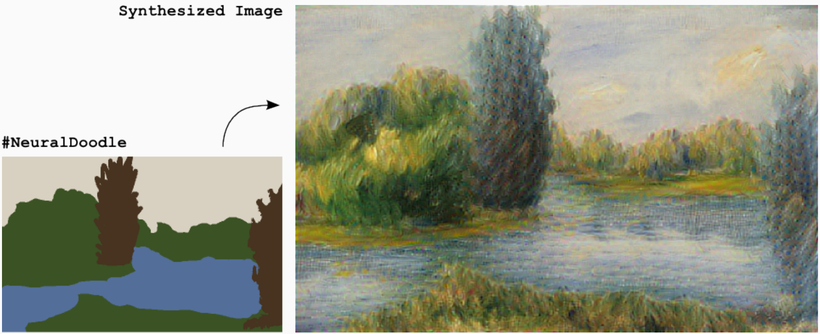
Goal: to add color to grayscale photographs.

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



(Source: Neural Doodle, Champandard, 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:

$$\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$$

2. Machine or model:

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

θ: parameters of the model

3. Loss, cost, objective function / energy:

$$\underset{\theta}{\operatorname{argmin}}\ E(\theta; \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N)$$

Learning from examples

Tools: $\begin{cases} \text{Data} & \leftrightarrow & \text{Statistics} \\ \text{Loss} & \leftrightarrow & \text{Optimization} \end{cases}$

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 $\boldsymbol{x}$. *Example: size, weight, citizenship, ...*

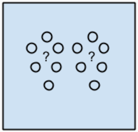
Training set: Set of data used to discover potentially predictive relationships.

Validation set: Set used to adjust the model hyperparameters.

Testing set: Set used to assess the performance of a model.

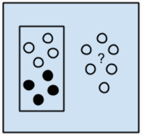
Label (Output): The class or outcome assigned to a sample. The actual prediction is often denoted by $\boldsymbol{y}$ and the desired/targeted class by $\boldsymbol{d}$ or $\boldsymbol{t}$. *Example: man/woman, wealth, education level, ...*

Learning approaches



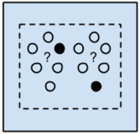
Unsupervised Learning Algorithms

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



Supervised Learning Algorithms

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

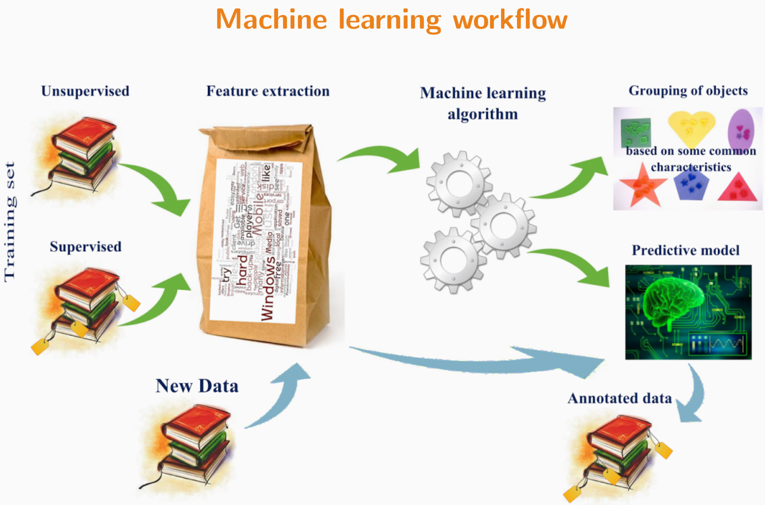


Semi-supervised Learning Algorithms

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.*

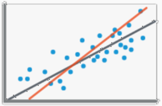
(Source: Jason Brownlee and Lucas Masuch)



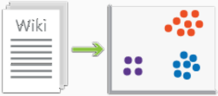
(Source: Michael Walker)

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

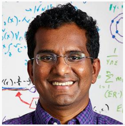
and many others...

- 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.

Quotes

I have worked all my life in Machine Learning, and I've never seen one algorithm knock over benchmarks like Deep Learning
– Andrew Ng (Stanford & Baidu)

Deep Learning is an algorithm which has no theoretical limitations of what it can learn; the more data you give and the more computational time you provide, the better it is *– Geoffrey Hinton (Google)*



Human-level artificial intelligence has the potential to help humanity thrive more than any invention that has come before it *– Dileep George (Co-Founder Vicarious)*

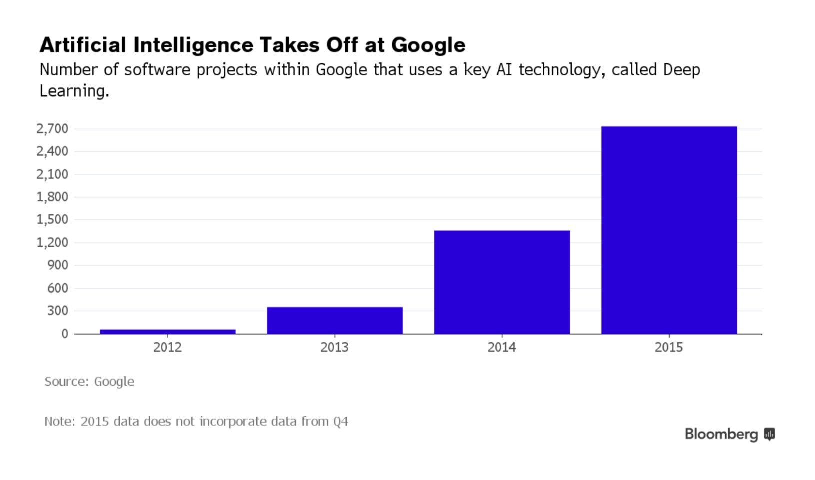
For a very long time it will be a complementary tool that human scientists and human experts can use to help them with the things that humans are not naturally good *– Demis Hassabis (Co-Founder DeepMind)*

(Source: Lucas Masuch)

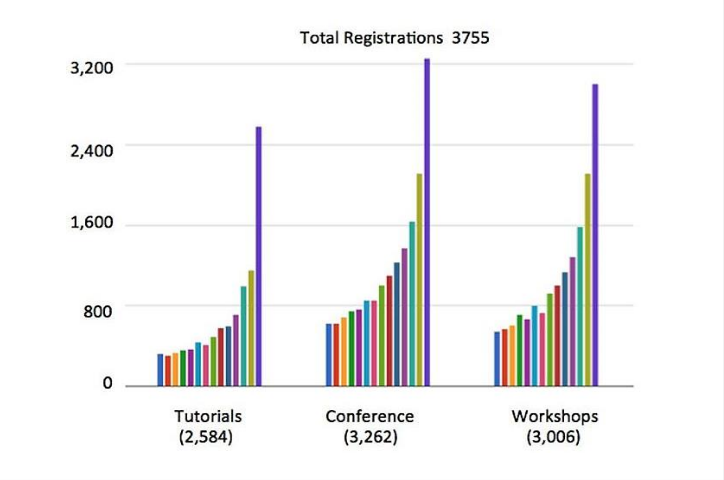
Actors and applications

- 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

Deep learning at Google

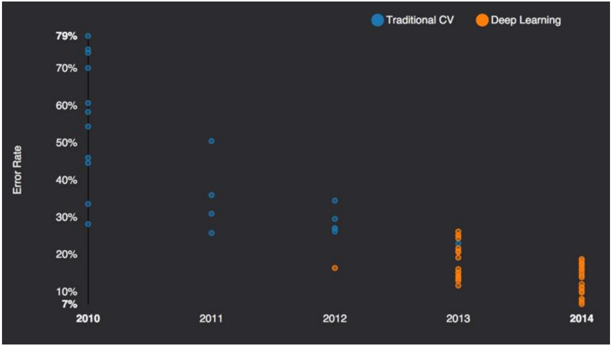


NIPS (Computational Neuroscience Conference) Growth



Success stories

AlexNet wins the ImageNet classification challenge (Krizhevsky, 2012)

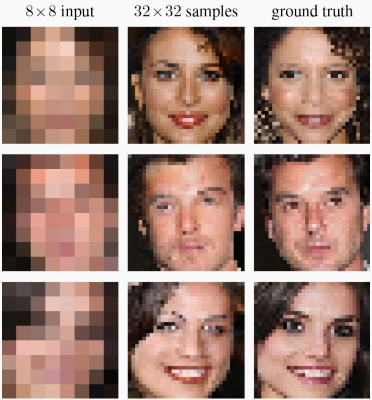


ImageNet: The "computer vision World Cup"

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.

Deep learning – Success stories

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."

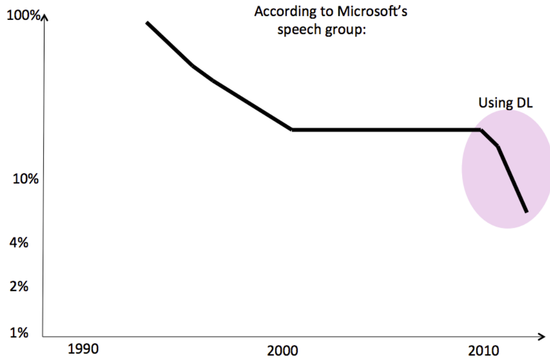
"woman is holding bunch of bananas."

"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

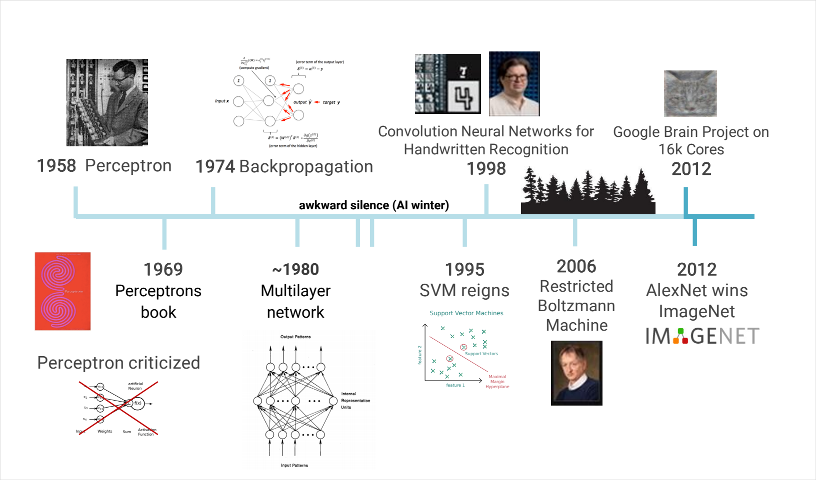
Speech recognition



Deep Learning in Speech Recognition

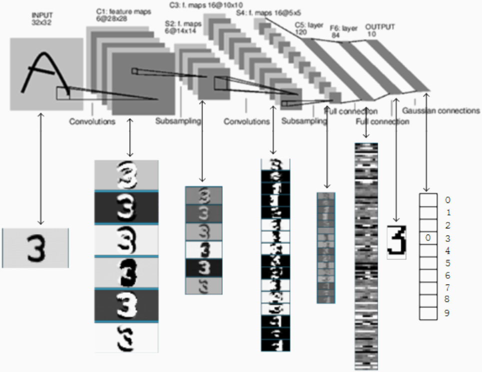
Same improvements in other fields: automatic translation, Go, . . .

Timeline of (deep) learning



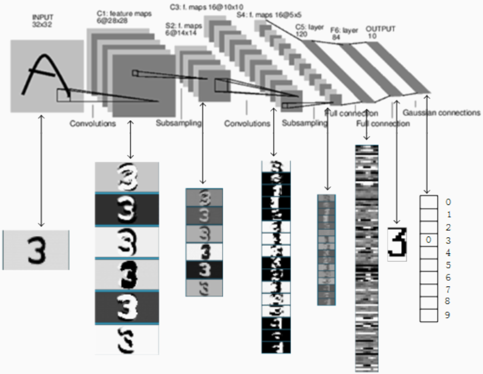
(Source: Lucas Masuch & Vincent Lepetit)

Neural networks for image classification



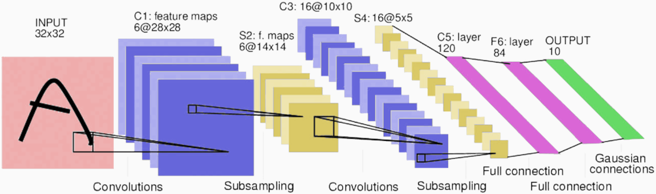
- **Goal:** Train a convolutional neural network for image classification

Neural networks for image classification

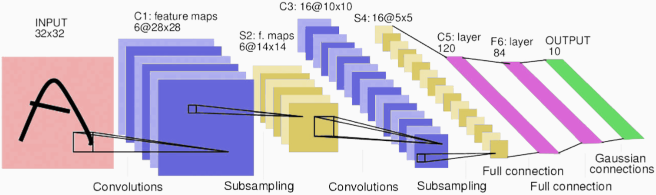


- **Goal:** Train a convolutional neural network for image classification
- **Goal: Understand the training** of a convolutional neural network for image classification

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

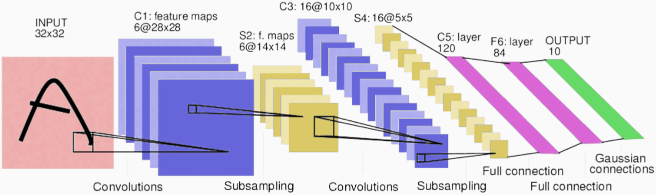


Understand the training of a convolutional neural network for image classification: **A lot of notions:** going backwards...



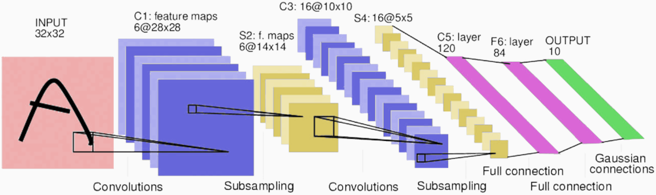
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.

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



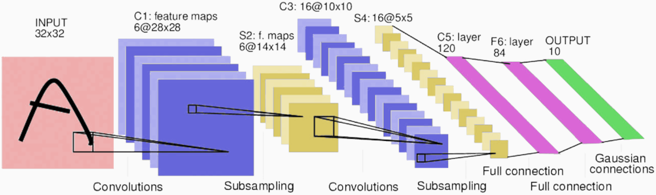
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.

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**.

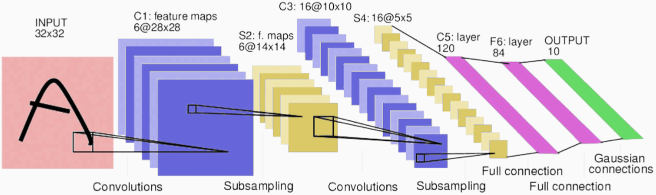
Understand the training of a convolutional neural network for image classification: **A lot of notions:** going backwards...



- **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**.

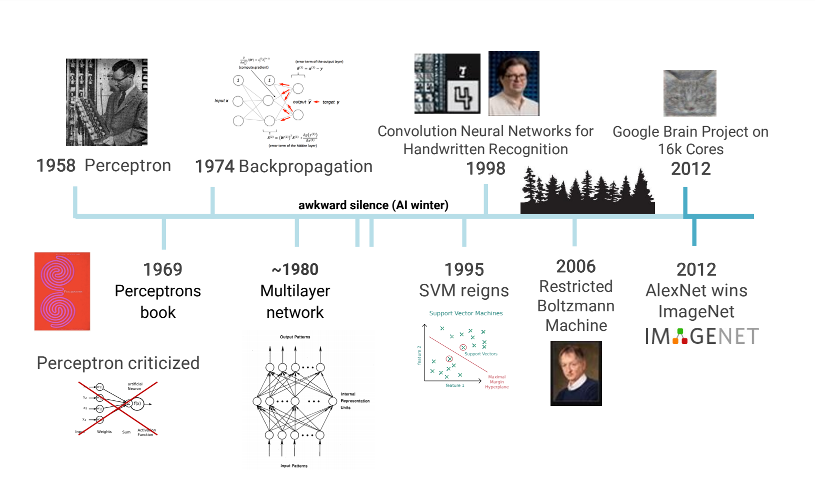
Goal of this course

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



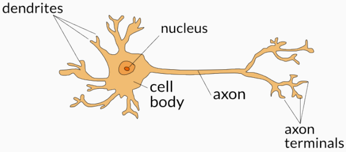
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done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

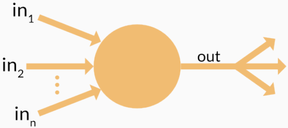
Timeline of (deep) learning



(Source: Lucas Masuch & Vincent Lepetit)

Perceptron


dendrites
nucleus
cell body
axon
axon terminals

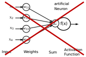

in_1
in_2
in_n
out

Perceptron

1958 Perceptron

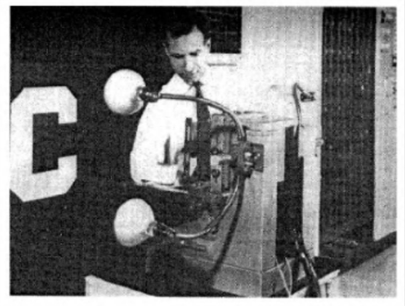
1969
Perceptrons
book

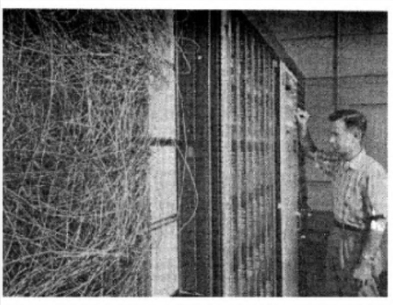
Perceptron criticized



(Source: Lucas Masuch & Vincent Lepetit)

Perceptron (Frank Rosenblatt, 1958)



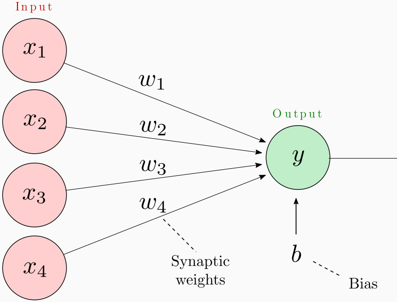


First binary classifier based on supervised learning (discrimination).

Foundation of modern artificial neural networks.

At that time: technological, scientific and philosophical challenges.

Representation of the Perceptron



$$\mathrm{sign}\,(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$y = \mathrm{sign}\left(\sum_{i=1}^{4} w_i x_i + b\right)$$

Parameters of the perceptron

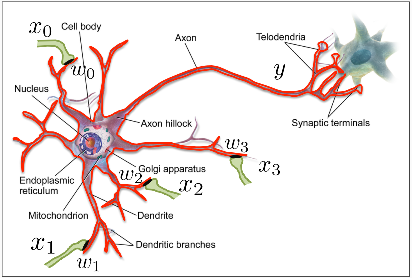
- $\boldsymbol{w}_k$: synaptic weights
- b: bias

$\Big\}\longleftarrow$ 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.



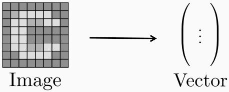
- When a neuron receives a stimulus with high enough voltage, it emits an action potential (aka, nerve impulse or spike). It is said to fire.
- The perceptron mimics this activation effect: it fires only when

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

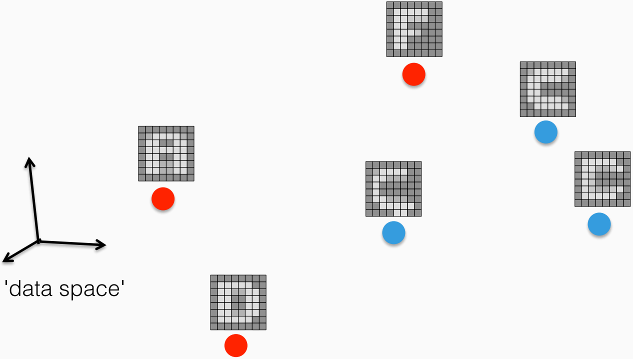
$$y = \underbrace{\mathrm{sign}(w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + b)}_{f(\boldsymbol{x};\boldsymbol{w})} = \begin{cases} +1 & \text{for the first class} \\ -1 & \text{for the second class} \end{cases}$$

Machine learning – Perceptron – Principle

1. Data are represented as vectors:



2. Collect training data with positive and negative examples:



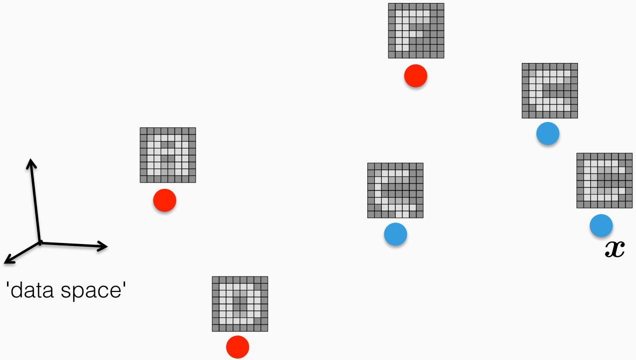
(Source: Vincent Lepetit)

③ **Training:** find $\boldsymbol{w}$ and b so that:

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

Dot product:

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{i=1}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$



(Source: Vincent Lepetit)

❸ **Training:** find $\boldsymbol{w}$ and b so that:

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

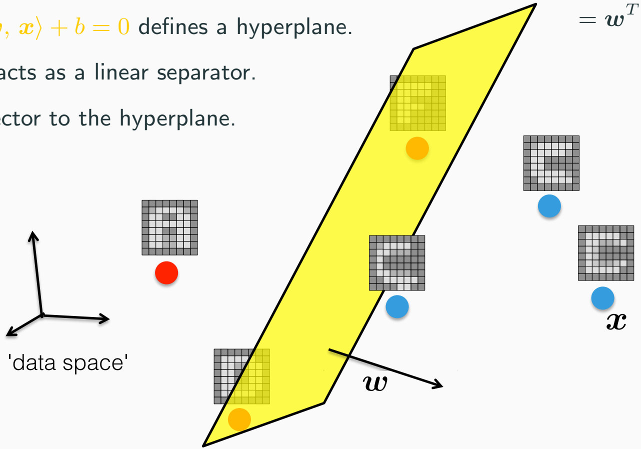
Dot product:

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{i=1}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

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

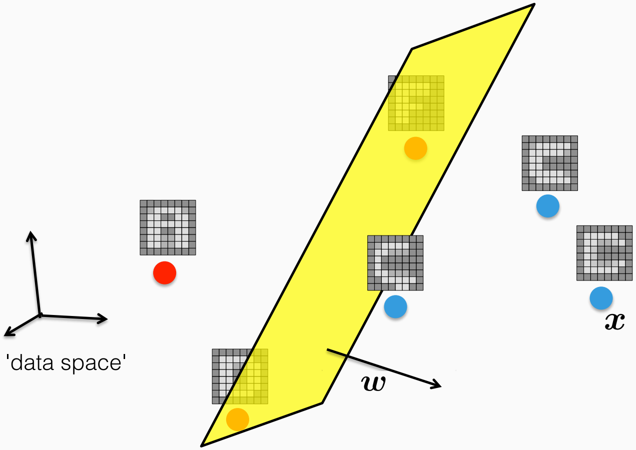
The hyperplane acts as a linear separator.

$\boldsymbol{w}$ is a normal vector to the hyperplane.



(Source: Vincent Lepetit)

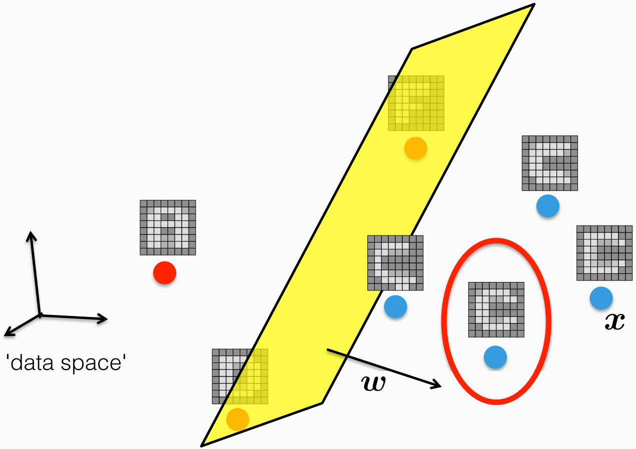
❹ **Testing:** the perceptron can now classify new examples.



(Source: Vincent Lepetit)

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

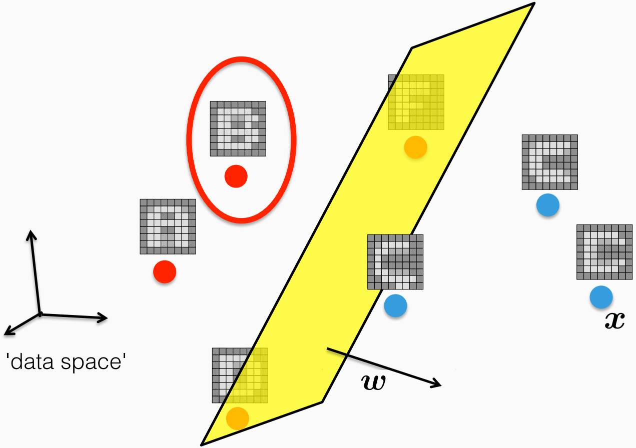
- A new example $\boldsymbol{x}$ is classified positive if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is positive,



(Source: Vincent Lepetit)

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

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



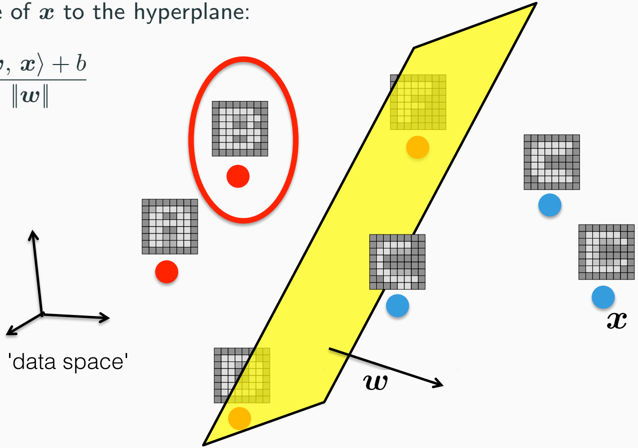
(Source: Vincent Lepetit)

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

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

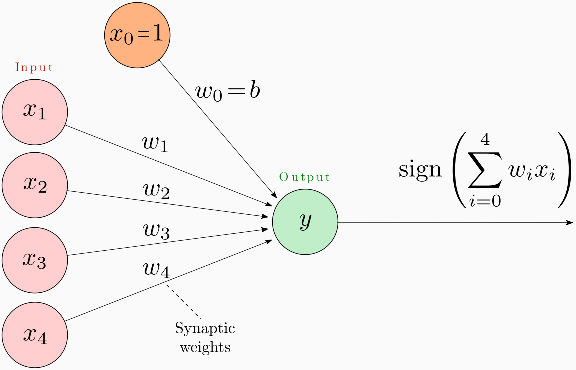
(signed) distance of $\boldsymbol{x}$ to the hyperplane:

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



(Source: Vincent Lepetit)

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 $\boldsymbol{w}$ from a labeled training dataset $\mathcal{T}$

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

i-th training example

desired output for sample i $\{-1, +1\}$

number of training samples

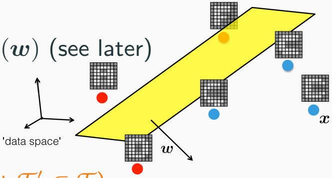
How: minimize classification errors

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

- penalize only misclassified samples ($y \neq d$) for which $d \times \langle \boldsymbol{w}, \boldsymbol{x} \rangle < 0$,
- 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 $t <= 0$.



Algorithm: (stochastic) gradient descent for $E(\boldsymbol{w})$ (see later)

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

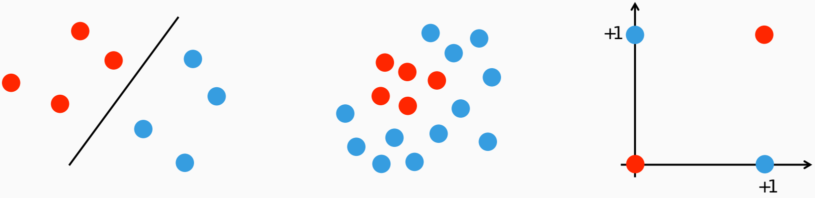
      Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma d \boldsymbol{x}$

- 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

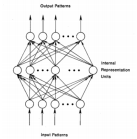
Artificial neural network

1958 Perceptron

1969
Perceptrons
book

Perceptron criticized

~1980
Multilayer
network



1989
Universal
Approximation
Theorem

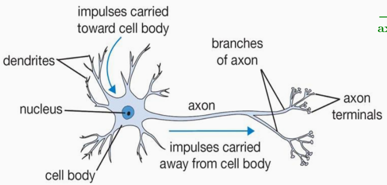
(Source: Lucas Masuch & Vincent Lepetit)

Artificial neural network

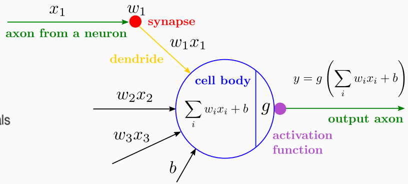
- Supervised learning method initially inspired by the behavior of the human brain.
- Consists of the inter-connection of several small units (just like in the human brain).
- Introduced in the late 50s, very popular in the 90s, reappeared in the 2010s with deep learning.
- Also referred to as Multi-Layer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

Machine learning – Artificial neural network

Artificial neuron (McCulloch & Pitts, 1943)



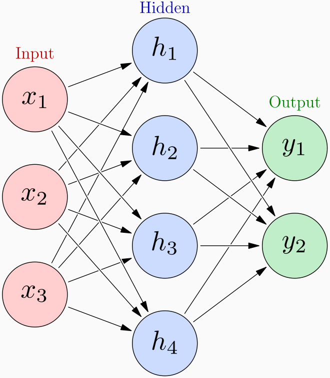
Biological neuron



Artificial neuron

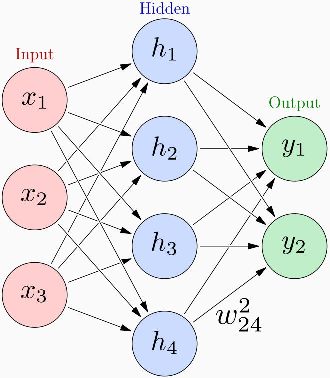
- An artificial neuron contains several incoming weighted connections, an outgoing connection and has a nonlinear activation function g.
- Neurons are trained to filter and detect specific features or patterns (e.g. edge, nose) by receiving weighted input, transforming it with the activation function 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



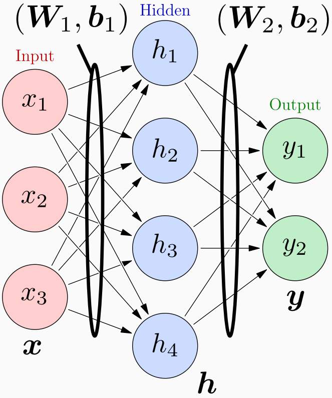
$$h_1 = g_1\left(w^1_{11}x_1 + w^1_{12}x_2 + w^1_{13}x_3 + b^1_1\right)$$
$$h_2 = g_1\left(w^1_{21}x_1 + w^1_{22}x_2 + w^1_{23}x_3 + b^1_2\right)$$
$$h_3 = g_1\left(w^1_{31}x_1 + w^1_{32}x_2 + w^1_{33}x_3 + b^1_3\right)$$
$$h_4 = g_1\left(w^1_{41}x_1 + w^1_{42}x_2 + w^1_{43}x_3 + b^1_4\right)$$

$$y_1 = g_2\left(w^2_{11}h_1 + w^2_{12}h_2 + w^2_{13}h_3 + w^2_{14}h_4 + b^2_1\right)$$
$$y_2 = g_2\left(w^2_{21}h_1 + w^2_{22}h_2 + w^2_{23}h_3 + w^2_{24}h_4 + b^2_2\right)$$

w^k_{ij} 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



$$h_1 = g_1\left(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1\right)$$
$$h_2 = g_1\left(w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1\right)$$
$$h_3 = g_1\left(w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1\right)$$
$$h_4 = g_1\left(w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1\right)$$

$$\boldsymbol{h} = g_1\left(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1\right)$$

$$y_1 = g_2\left(w_{11}^2 h_1 + w_{12}^2 h_2 + w_{13}^2 h_3 + w_{14}^2 h_4 + b_1^2\right)$$
$$y_2 = g_2\left(w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2\right)$$

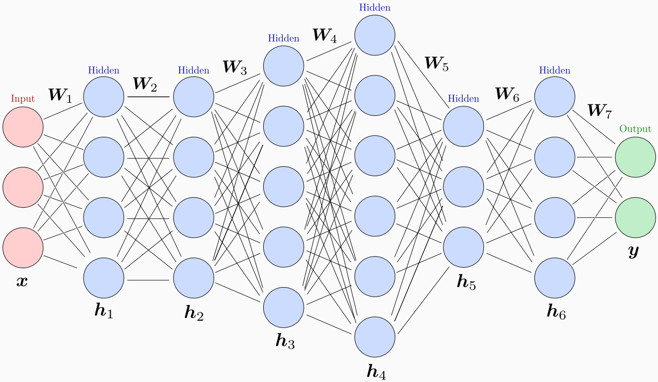
$$\boldsymbol{y} = g_2\left(\boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2\right)$$

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 $\boldsymbol{W_k}$ and biases $\boldsymbol{b}_k$ 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:

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

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

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

- It can be defined recursively as

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

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

Activation functions

Linear units: $g(a) = a$

$$
\begin{array}{r}
\boldsymbol{y} = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L \\
\boldsymbol{h}_{L-1} = \boldsymbol{W}_{L-1} \boldsymbol{h}_{L-2} + \boldsymbol{b}_{L-1} \\
\hline
\boldsymbol{y} = \boldsymbol{W}_L \boldsymbol{W}_{L-1} \boldsymbol{h}_{L-2} + \boldsymbol{W}_L \boldsymbol{b}_{L-1} + \boldsymbol{b}_L \\
\hline
\boldsymbol{y} = \boldsymbol{W}_L \dots \boldsymbol{W}_1 \boldsymbol{x} + \sum_{k=1}^{L-1} \boldsymbol{W}_L \dots \boldsymbol{W}_{k+1} \boldsymbol{b}_k + \boldsymbol{b}_L
\end{array}
$$

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

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

or Heaviside (aka, step) activation functions

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

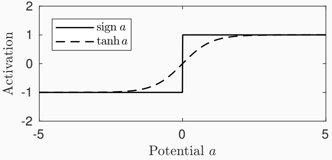
Discontinuities in the hidden layers
make the optimization really difficult.

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 :

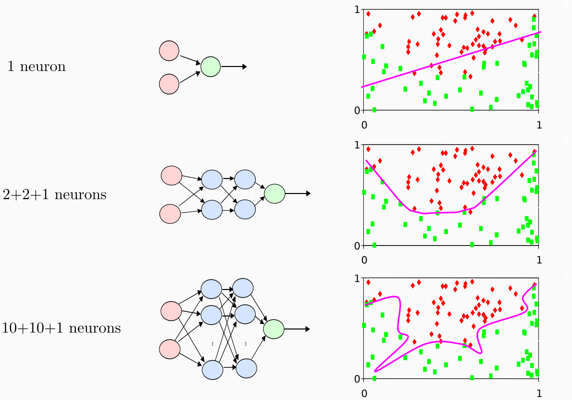
$$\tanh(a/2) = 2\text{logistic}(a) - 1$$

- 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:

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

The architecture of the network defines the shape of the separator



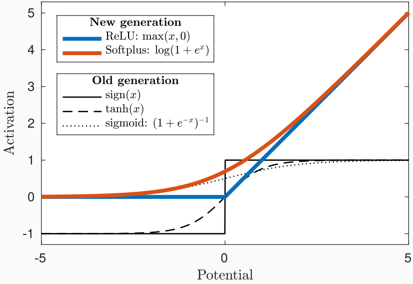
Separation
$\{\boldsymbol{x} \text{ s.t. } P(C_1|\boldsymbol{x}) = P(C_2|\boldsymbol{x})\}$

Complexity/capacity of the network
$\Rightarrow$
Trade-off between generalization and overfitting.

Activation functions

“Modern” units:

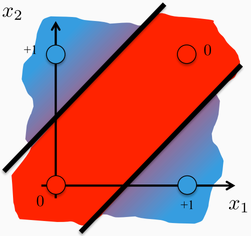
$$\underbrace{g(a) = \max(a, 0)}_{\text{ReLU}} \quad \text{or} \quad \underbrace{g(a) = \log(1 + e^{a})}_{\text{Softplus}}$$

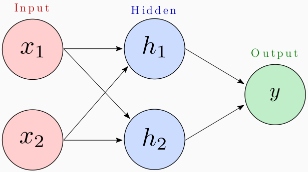


Most neural networks use ReLU (Rectifier linear unit) – $\max(a, 0)$ – nowadays for hidden layers, since it trains much faster, is more expressive than logistic function and prevents the gradient vanishing problem.

(Source: Lucas Masuch)

Neural networks solve non-linear separable problems

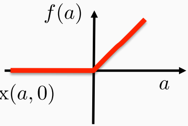




$$\boldsymbol{h} = g(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)$$

$$y = \langle \boldsymbol{w}_2, \boldsymbol{h} \rangle + b_2$$

$f(a) = \max(a, 0)$



$$\boldsymbol{W}_1 = \begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix}, \ \boldsymbol{b}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \ \boldsymbol{w}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \ b_2 = 0$$

(Source: Vincent Lepetit)

Tasks, architectures and loss functions

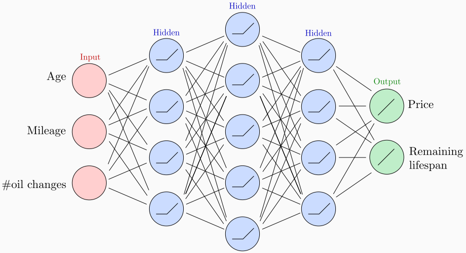
Approximation – Least square regression

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

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

i-th training example — desired output for sample i — number of training samples

- **Typical architecture:**



- Hidden layer:

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

- 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)

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

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

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

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

Multiclass classification – Multivariate logistic regression

(aka, multinomial classification)

- **Goal:** Classify an object $\boldsymbol{x}$ into one among K classes $C_1, \ldots, C_K$.

- **How:** Estimate the coefficients $\boldsymbol{W}$ of a multivariate function

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

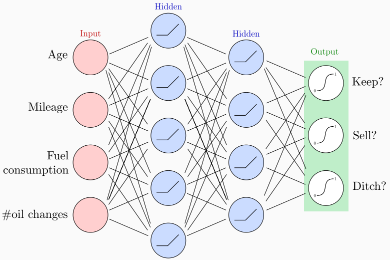
  from training examples $\mathcal{T} = \{(\boldsymbol{x}^i, \boldsymbol{d}^i)\}$ where $\boldsymbol{d}^i$ is a 1-of-K (one-hot) code

  - Class 1: $\boldsymbol{d}^i = (1, 0, \ldots, 0)^T$ if $\boldsymbol{x}^i \in C_1$
  - Class 2: $\boldsymbol{d}^i = (0, 1, \ldots, 0)^T$ if $\boldsymbol{x}^i \in C_2$
  - $\ldots$
  - Class K: $\boldsymbol{d}^i = (0, 0, \ldots, 1)^T$ if $\boldsymbol{x}^i \in C_K$

- $\boldsymbol{y}_k = f(\boldsymbol{x}; \boldsymbol{W})$ is understood as the probability of $\boldsymbol{x} \in C_k$.
- **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:

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

- Output layer:

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

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

$$\operatorname{argmax}_k \mathsf{softmax}(\boldsymbol{a}) = \operatorname{argmax}_k \boldsymbol{a}$$

Multiclass classification – Multivariate logistic regression

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

$$E(\boldsymbol{W}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} d_k^i \log y_k^i \quad \text{with} \quad \boldsymbol{y}^i = f(\boldsymbol{x}^i;\boldsymbol{W}) = \mathrm{softmax}(\boldsymbol{a}^i) \in (0,1)^K.$$

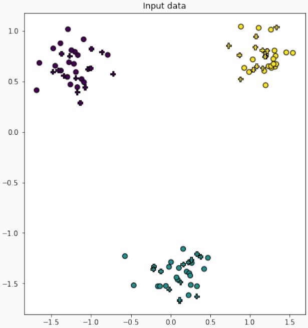
$$= -\sum_{i=1}^{N}\left[a_{d^i}^i - \log\left(\sum_{k=1}^{K}\exp(a_k^i)\right)\right] \quad \text{with } d^i \text{ the class of } \boldsymbol{x}^i.$$

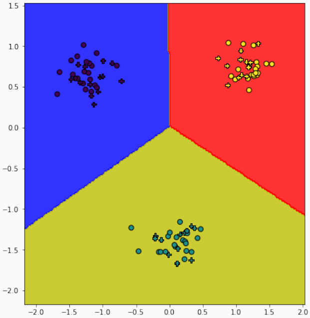
and look for $\boldsymbol{W}^*$ such that $\nabla E(\boldsymbol{W}^*) = 0$.

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

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

Multivariate logistic regression



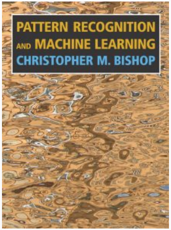


Multiclass classification – Multivariate logistic regression

- SVMs allow for multiclass classification but are not easily plugable 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

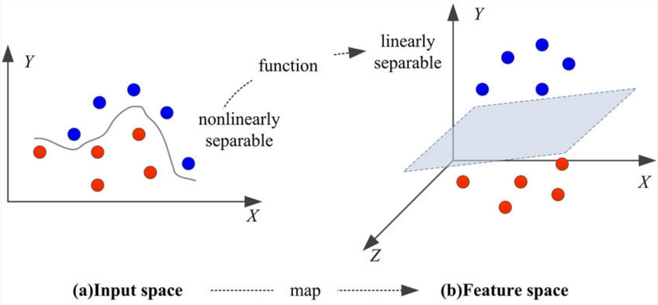
- **Goal:** Classify an object $\boldsymbol{x}$ into one among K classes $C_1, \ldots, C_K$.
- Training set: $\mathcal{T} = \{(\boldsymbol{x}_n, t_n),\ n = 1, \ldots, N\}$, $t_n \in \{1, \ldots, K\}$ encodes the class of $\boldsymbol{x}_n$.
- Each t_n is transformed into a vector $\boldsymbol{t}_n \in \{0, 1\}^K$ with a 1-of-K code:
  - Class 1: $\boldsymbol{t}_n = (1, 0, \ldots, 0)^T$ if $\boldsymbol{x}_n \in C_1$, i.e $t_n = 1$,
  - Class 2: $\boldsymbol{t}_n = (0, 1, \ldots, 0)^T$ if $\boldsymbol{x}_n \in C_2$, i.e $t_n = 2$,
  - $\ldots$
  - Class K: $\boldsymbol{t}_n = (0, 0, \ldots, 1)^T$ if $\boldsymbol{x}_n \in C_K$, 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 $\varphi : \mathbb{R}^p \to \mathbb{R}^D$ to each $\boldsymbol{x_n}$:

$$\varphi_n = \varphi(\boldsymbol{x_n}), \quad n = 1, \ldots, N.$$

- 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.



(Source: url)

Linear classifier

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

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

$$\boldsymbol{w}_k^T \varphi + b_k > \boldsymbol{w}_\ell^T \varphi + b_\ell?$$

- Class k region:

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

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

Bias trick for linear classifier

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

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

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

Bias trick for linear classifier

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

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

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

Multivariate logistic regression

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

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

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

$$p(C_k|\varphi) = \text{Probability of being in class } C_k \text{ given feature vector } \varphi.$$

- **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 $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K \in \mathbb{R}^D$ such that:

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

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

Multivariate logistic regression

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

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

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

Training:

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

Multivariate logistic regression

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

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

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

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

Multivariate logistic regression

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

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

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

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

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

Loglikelihood:

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

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

- Expression of loglikelihood:

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

Multivariate logistic regression

Loglikelihood:

- Expression of loglikelihood:

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

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

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

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

Multivariate logistic regression

Loglikelihood:

- Expression of loglikelihood:

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

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

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

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

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

Multivariate logistic regression

Gradient of loglikelihood:

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

- Linear part: Partial gradient with respect to column w_ℓ, $\ell \in \{1, \ldots, K\}$:

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

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

$$\nabla_{\boldsymbol{w}_\ell} \log\left(\sum_{j=1}^{K} \exp(\boldsymbol{w}_j^T \varphi_n)\right) = \nabla_{\boldsymbol{w}_\ell} \log\left(\exp(\boldsymbol{w}_\ell^T \varphi_n) + \text{constant}\right)$$

$$= ?$$

Multivariate logistic regression

Gradient of log-likelihood:

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

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

Here,

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

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

So,

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

since

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

Multivariate logistic regression

Gradient of log-likelihood:

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

- For each column $w_\ell \in \mathbb{R}^D$ of $\boldsymbol{W}$, $\ell \in \{1, \ldots, K\}$,

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

Multivariate logistic regression

Gradient of log-likelihood:

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

- For each column $w_\ell \in \mathbb{R}^D$ of $\boldsymbol{W}$, $\ell \in \{1, \ldots, K\}$,

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

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

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

- 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:

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

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

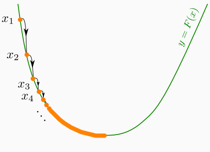
$$x^{(n+1)} = x^{(n)} \underbrace{- \gamma \nabla F(x^{(n)})}_{\text{direction of greatest descent}},$$

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

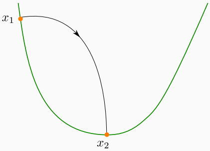
$$\nabla F(x^\star) = 0 \ .$$

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

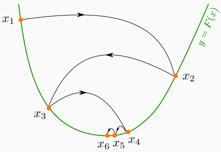
One dimension



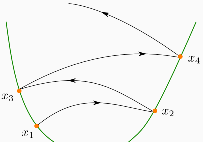
Small step size
Slow convergence



Good step size
Fast convergence

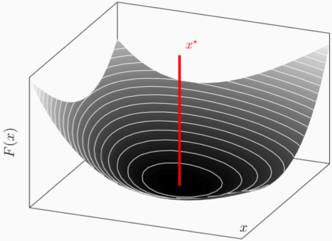


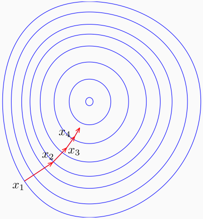
Large step size
Slow convergence

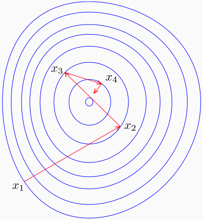


Too large step size
Divergence

Two dimensions







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

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:

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

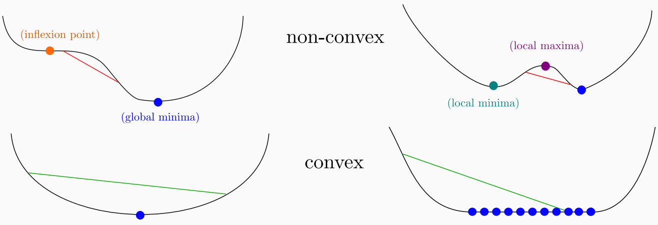
    and compute **averaged weights**

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

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

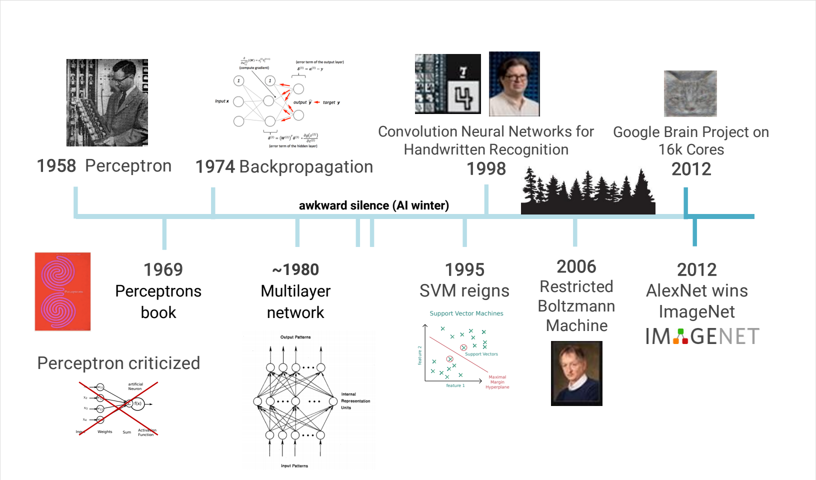
Non convexity in machine learning

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



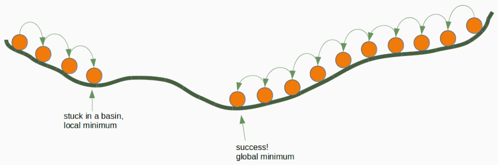
Machine learning – Timeline

Timeline of (deep) learning



(Source: Lucas Masuch & Vincent Lepetit)

Backpropagation



Questions?

Slides from Charles Deledalle

Sources, images courtesy and acknowledgment

K. Chatfield	A. Horodniceanu	L. Masuch
P. Gallinari,	Y. LeCun	A. Ng
C. Hazırbaş	V. Lepetit	M. Ranzato