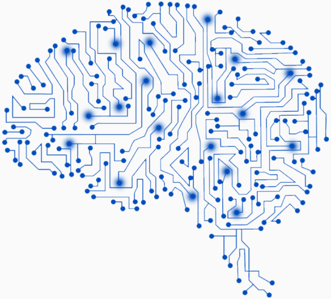
Atelier Réseaux de neurones

Emile Pierret

Mercredi 19 juin 2024



Credits

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

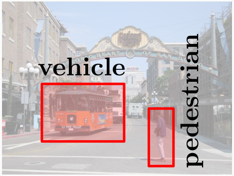
www.charles-deledalle.fr/

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

AI and Machine Learning

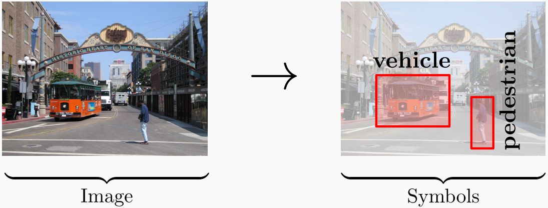
Image

$\longrightarrow$



Symbols

Computer vision – Artificial Intelligence – Machine Learning



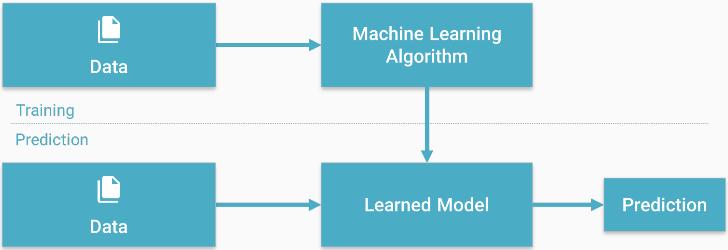
Definition (Oxford dictionary)

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.

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:



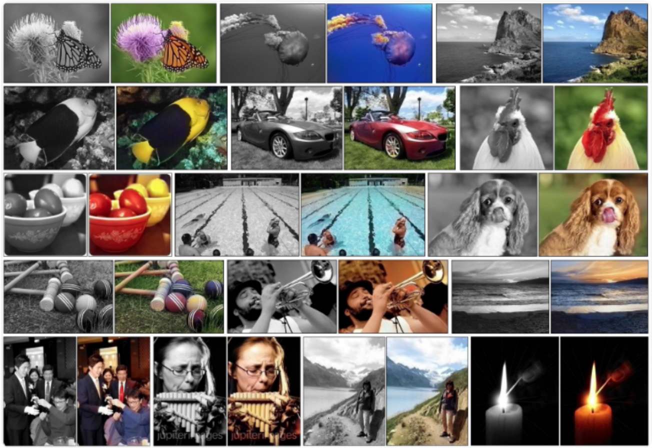
(Source: Lucas Masuch)

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).

Image colorization



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

Goal: to add color to grayscale photographs.

IP ∩ CV – Style transfer

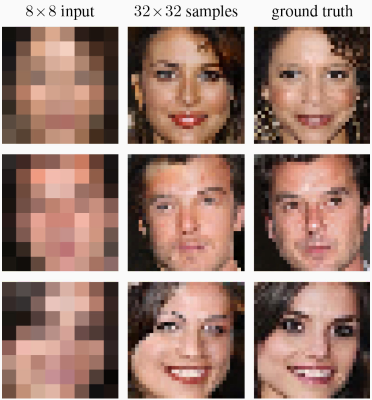
Style transfer

(Source: Gatys, Ecker and Bethge, 2015)

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

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.

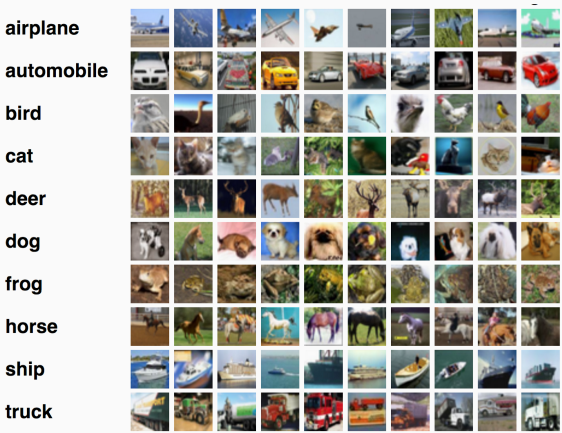
Success stories

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

These people do not exist.

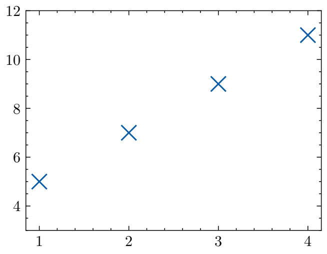
Computer vision – Image classification

Our goal: Image classification



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

Example of linear regression:



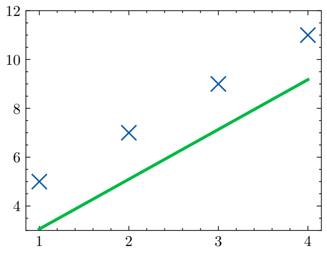
- Data: $(x_i, y_i)_{1 \leqslant i \leqslant N}$
- Model: $\{f_\theta : x \mapsto ax + b\}_{\theta=(a,b)\in\mathbb{R}^2}$
- Loss :

$$E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right) = \sum_{i=1}^{N} (f_\theta(x_i) - y_i)^2$$

- Objective:

$$\theta^\star = \mathrm{argmin}_{\theta\in\mathbb{R}^2}\, E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right)$$

Example of linear regression:



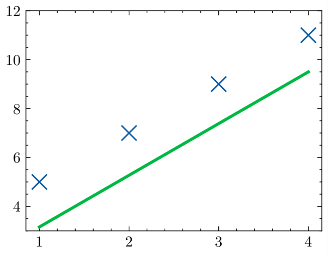
- Data: $(x_i, y_i)_{1 \leqslant i \leqslant N}$
- Model: $\{f_\theta : x \mapsto ax + b\}_{\theta=(a,b)\in\mathbb{R}^2}$
- Loss :

$$E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right) = \sum_{i=1}^{N} \left(f_\theta(x_i) - y_i\right)^2$$

- Objective:

$$\theta^\star = \operatorname{argmin}_{\theta\in\mathbb{R}^2} E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right)$$

Example of linear regression:



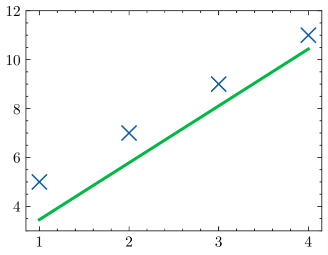
- Data: $(x_i, y_i)_{1 \leqslant i \leqslant N}$
- Model: $\{f_\theta : x \mapsto ax + b\}_{\theta=(a,b)\in\mathbb{R}^2}$
- Loss :

$$E\left(\theta, (x_i, y_i)_{1\leqslant i\leqslant N}\right) = \sum_{i=1}^{N} \left(f_\theta(x_i) - y_i\right)^2$$

- Objective:

$$\theta^\star = \mathrm{argmin}_{\theta\in\mathbb{R}^2}\, E\left(\theta, (x_i, y_i)_{1\leqslant i\leqslant N}\right)$$

Example of linear regression:



- Data: $(x_i, y_i)_{1 \leqslant i \leqslant N}$
- Model: $\{f_\theta : x \mapsto ax + b\}_{\theta=(a,b)\in\mathbb{R}^2}$
- Loss :

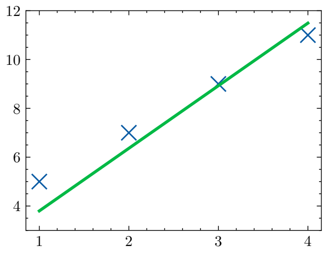
$$E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right) = \sum_{i=1}^{N} \left(f_\theta(x_i) - y_i\right)^2$$

- Objective:

$$\theta^\star = \operatorname{argmin}_{\theta\in\mathbb{R}^2} E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right)$$

Machine learning – Learning from examples

Example of linear regression:



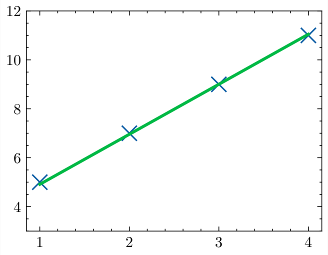
- Data: $(x_i, y_i)_{1 \leqslant i \leqslant N}$
- Model: $\{f_\theta : x \mapsto ax + b\}_{\theta=(a,b)\in\mathbb{R}^2}$
- Loss :

$$E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right) = \sum_{i=1}^{N} \left(f_\theta(x_i) - y_i\right)^2$$

- Objective:

$$\theta^\star = \mathrm{argmin}_{\theta\in\mathbb{R}^2} \, E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right)$$

Example of linear regression:



- Data: $(x_i, y_i)_{1 \leqslant i \leqslant N}$
- Model: $\{f_\theta : x \mapsto ax + b\}_{\theta=(a,b)\in\mathbb{R}^2}$
- Loss :

$$E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right) = \sum_{i=1}^{N} (f_\theta(x_i) - y_i)^2$$

- Objective:

$$\theta^\star = \mathrm{argmin}_{\theta\in\mathbb{R}^2}\, E\left(\theta, (x_i, y_i)_{1 \leqslant i \leqslant N}\right)$$

Learning from examples

Learning from examples

3 main ingredients

1. Training set / examples:

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

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

Learning from examples

3 main ingredients

1. Training set / examples:

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

2. Machine or model:

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

θ: parameters of the model

3. Loss, cost, objective function / energy:

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

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.

Data

Terminology

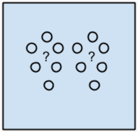
Sample (Observation or Data): item to process (*e.g.*, classify). *Example: an individual, a document, a picture, a sound, a video. . .*

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

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

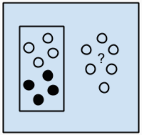
Label (Output): The class or outcome assigned to a sample. The actual prediction is often denoted by $\boldsymbol{y}$ and the desired/targeted class by $\boldsymbol{d}$ or $\boldsymbol{t}$.

Learning approaches



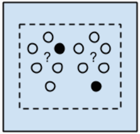
Unsupervised Learning Algorithms

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.*

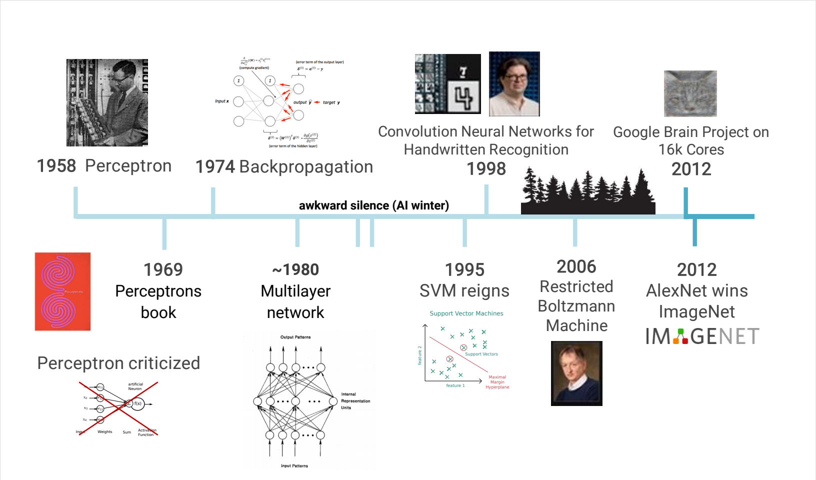
Reinforcement learning: Learning based on feedback or reward. *Example: learn to play chess by winning or losing.*

(Source: Jason Brownlee and Lucas Masuch)

The model: artificial neural network

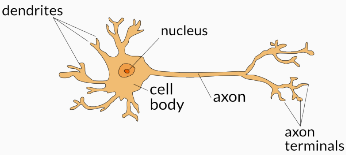
Machine learning – Timeline

Timeline of (deep) learning

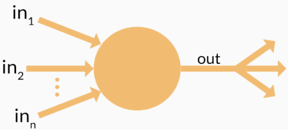


(Source: Lucas Masuch & Vincent Lepetit)

Perceptron



dendrites
nucleus
cell
body
axon
axon
terminals



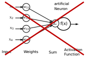
in1
in2
inn
out

Perceptron

1958 Perceptron

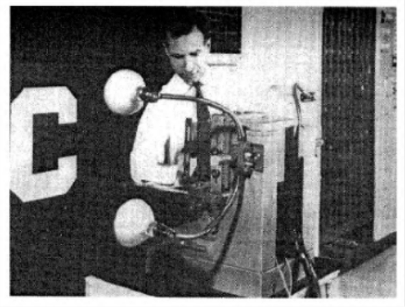
1969
Perceptrons
book

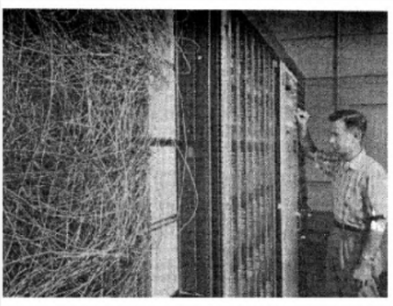
Perceptron criticized



(Source: Lucas Masuch & Vincent Lepetit)

Perceptron (Frank Rosenblatt, 1958)



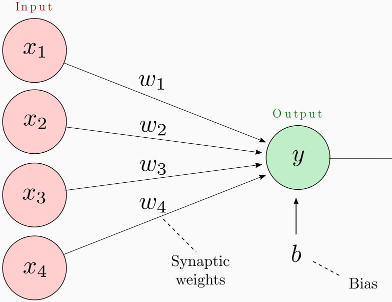


First binary classifier based on supervised learning (discrimination).

Foundation of modern artificial neural networks.

At that time: technological, scientific and philosophical challenges.

Representation of the Perceptron



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

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

Parameters of the perceptron

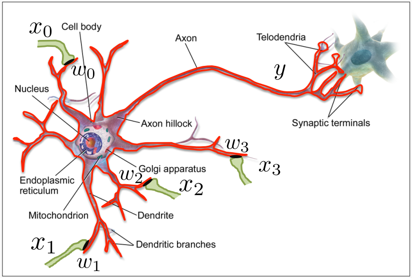
- $\boldsymbol{w}_k$: synaptic weights
- b: bias

$\longleftarrow$ 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.



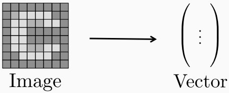
- When a neuron receives a stimulus with high enough voltage, it emits an action potential (aka, nerve impulse or spike). It is said to fire.
- The perceptron mimics this activation effect: it fires only when

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

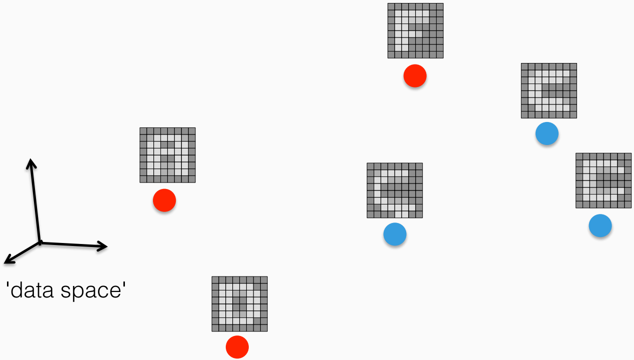
$$y = \underbrace{\text{sign}(w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + b)}_{f(\boldsymbol{x};\boldsymbol{w})} = \begin{cases} +1 & \text{for the first class} \\ -1 & \text{for the second class} \end{cases}$$

Machine learning – Perceptron – Principle

1. Data are represented as vectors:



2. Collect training data with positive and negative examples:



(Source: Vincent Lepetit)

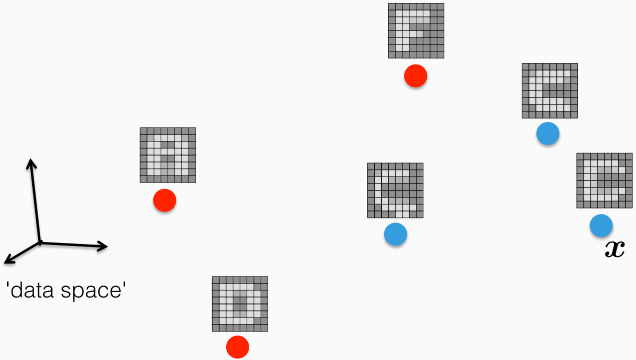
Machine learning – Perceptron – Principle

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:

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



(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 $\boldsymbol{x}$.

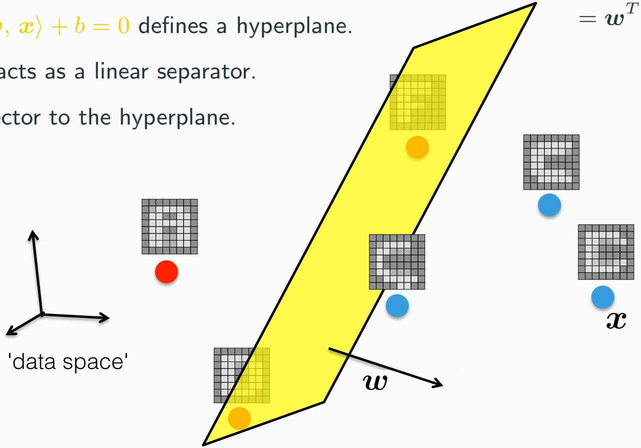
Dot product:

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

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

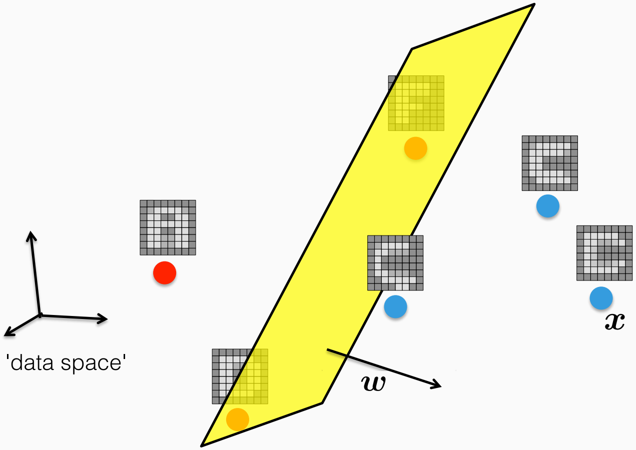
The hyperplane acts as a linear separator.

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



(Source: Vincent Lepetit)

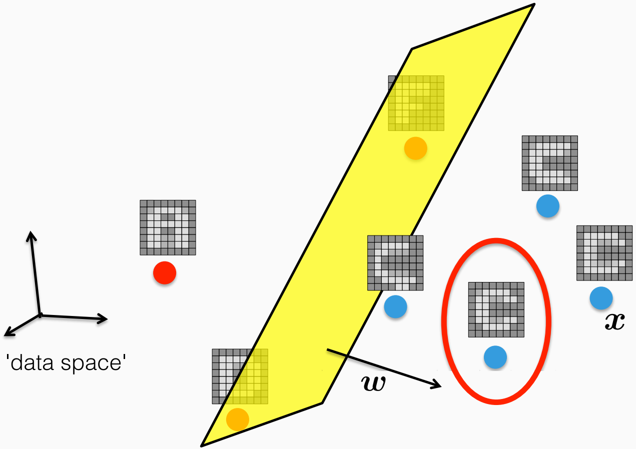
❹ **Testing:** the perceptron can now classify new examples.



(Source: Vincent Lepetit)

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

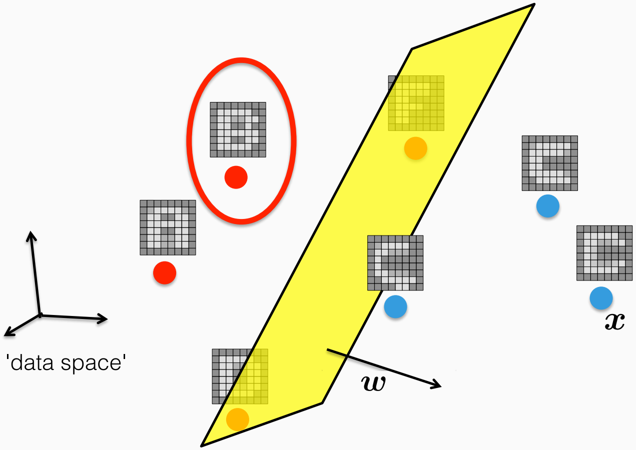
- A new example $\boldsymbol{x}$ is classified positive if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is positive,



(Source: Vincent Lepetit)

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

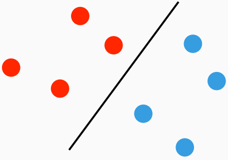
- A new example $\boldsymbol{x}$ is classified positive if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is positive,
- and negative if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is negative.



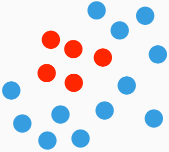
(Source: Vincent Lepetit)

Perceptrons book (Minsky and Papert, 1969)

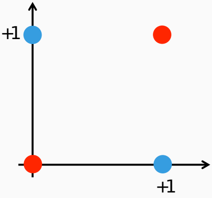
A perceptron can only classify data points that are linearly separable:



Linearly separable



Nonlinearly separable



The xor function

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

(Source: Vincent Lepetit)

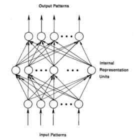
Artificial neural network

1958 Perceptron

1969
Perceptrons
book

Perceptron criticized

~1980
Multilayer
network



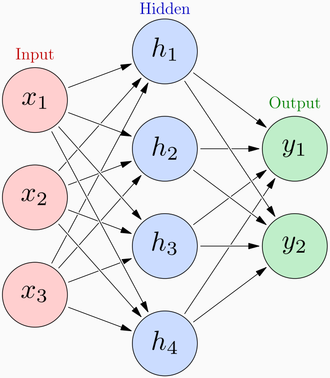
1989
Universal
Approximation
Theorem

(Source: Lucas Masuch & Vincent Lepetit)

Artificial neural network

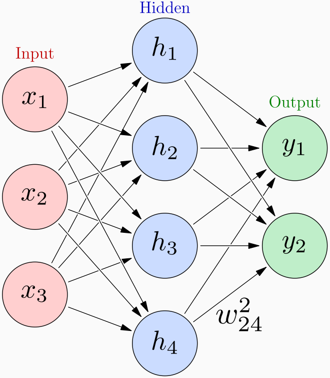
- Supervised learning method initially inspired by the behavior of the human brain.
- Consists of the inter-connection of several small units (just like in the human brain).
- Introduced in the late 50s, very popular in the 90s, reappeared in the 2010s with deep learning.
- Also referred to as Multi-Layer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

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.

Artificial neural network / Multilayer perceptron / NeuralNet



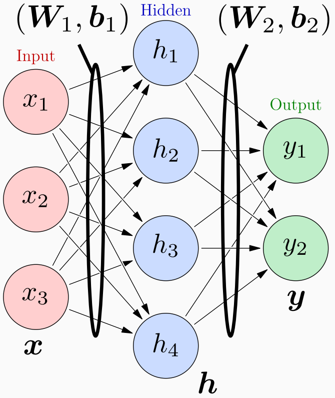
$$h_1 = g_1\left(w^1_{11}x_1 + w^1_{12}x_2 + w^1_{13}x_3 + b^1_1\right)$$
$$h_2 = g_1\left(w^1_{21}x_1 + w^1_{22}x_2 + w^1_{23}x_3 + b^1_2\right)$$
$$h_3 = g_1\left(w^1_{31}x_1 + w^1_{32}x_2 + w^1_{33}x_3 + b^1_3\right)$$
$$h_4 = g_1\left(w^1_{41}x_1 + w^1_{42}x_2 + w^1_{43}x_3 + b^1_4\right)$$

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

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



$$h_1 = g_1\left(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1\right)$$
$$h_2 = g_1\left(w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1\right)$$
$$h_3 = g_1\left(w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1\right)$$
$$h_4 = g_1\left(w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1\right)$$
$$\boldsymbol{h} = g_1\left(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1\right)$$

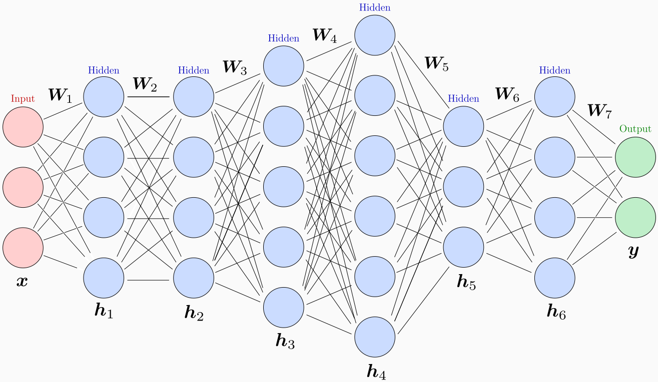
$$y_1 = g_2\left(w_{11}^2 h_1 + w_{12}^2 h_2 + w_{13}^2 h_3 + w_{14}^2 h_4 + b_1^2\right)$$
$$y_2 = g_2\left(w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2\right)$$
$$\boldsymbol{y} = g_2\left(\boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2\right)$$

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.

Activation functions

Linear units: $g(a) = a$

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

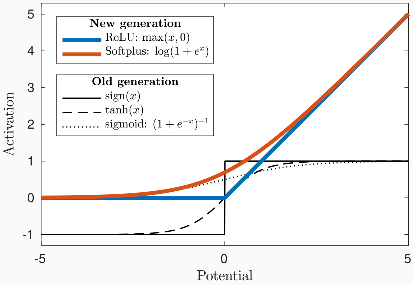
We can always find an equivalent network without hidden units, because compositions of affine functions are affine.

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

"Modern" units:

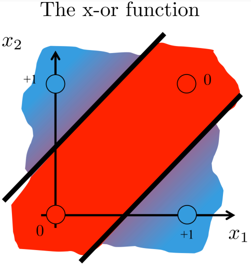
$$\underbrace{g(a) = \max(a, 0)}_{\text{ReLU}} \quad \text{or} \quad \underbrace{g(a) = \log(1 + e^a)}_{\text{Softplus}}$$

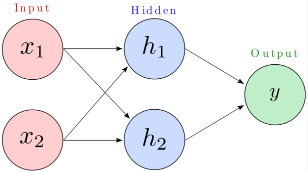


Most neural networks use ReLU (Rectifier linear unit) – $\max(a, 0)$ – nowadays for hidden layers, since it trains much faster, is more expressive than logistic function and prevents the gradient vanishing problem.

(Source: Lucas Masuch)

Neural networks solve non-linear separable problems

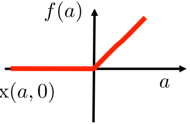




$$\boldsymbol{h} = g(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)$$

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

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



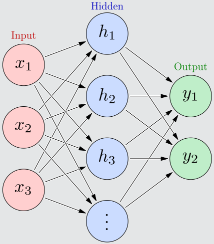
$$\boldsymbol{W}_1 = \begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix}, \; \boldsymbol{b}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \; \boldsymbol{w}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \; b_2 = 0$$

(Source: Vincent Lepetit)

Universal Approximation Theorem

(Hornik et al, 1989; Cybenko, 1989)

Any continuous function $f : K \subset \mathbb{R}^N \to \mathbb{R}^K$ 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.



$$\boldsymbol{h} = g(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$$
$$\boldsymbol{y} = \boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2$$

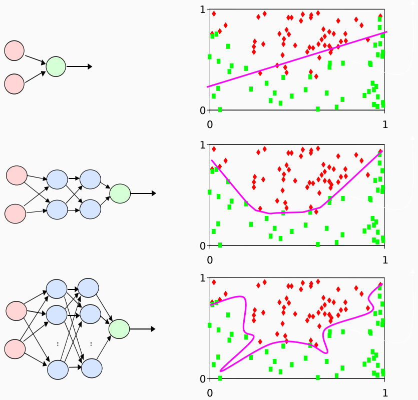
- Works if and only if g is not polynomial (and thus non linear).
- The theorem does not say how large the network needs to be.
- No guarantee that the training algorithm will be able to train the network.

The architecture of the network defines the shape of the separator

1 neuron

2+2+1 neurons

10+10+1 neurons



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.

Optimization

ANN – Optimization

- The parameters of the neural network are

$$\boldsymbol{\theta} = (\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_L)$$

- The parameters of the neural network are

$$\boldsymbol{\theta} = (\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2, \dots, \boldsymbol{W}_L, \boldsymbol{b}_L)$$

- Training the network = minimizing the training loss $E(\boldsymbol{\theta})$

Objective: $\min_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$

$$\Rightarrow \quad \nabla E(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{W}_1} & \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{b}_1} & \dots & \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{W}_L} & \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{b}_L} \end{pmatrix}^T = 0$$

ANN – Optimization

- The parameters of the neural network are

$$\boldsymbol{\theta} = (\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_L)$$

- Training the network = minimizing the training loss $E(\boldsymbol{\theta})$

Objective: $\min_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$

$$\Rightarrow \quad \nabla E(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{W}_1} & \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{b}_1} & \ldots & \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{W}_L} & \frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{b}_L} \end{pmatrix}^T = 0$$

- **Solution:** no closed-form solutions ⇒ use (stochastic) gradient descent.

The Stochastic Gradient Descent (SGD)

Denoting $\mathcal{T}$ the training dataset, the loss functions are of the form

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} L(f_\theta(\boldsymbol{x}^i); \boldsymbol{d}^i)$$

where f_θ is the neural network.

Example: $E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} \|f_\theta(\boldsymbol{x}^i) - \boldsymbol{d}^i)\|^2$

The Stochastic Gradient Descent (SGD)

Denoting $\mathcal{T}$ the training dataset, the loss functions are of the form

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} L(f_\theta(\boldsymbol{x}^i); \boldsymbol{d}^i)$$

where f_θ is the neural network.

Example: $E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} \|f_\theta(\boldsymbol{x}^i) - \boldsymbol{d}^i)\|^2$

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

- Initialize $\boldsymbol{\theta}_0$ randomly
- For $0 \leqslant k \leqslant N$,
    - For all $(\boldsymbol{x}, \boldsymbol{d}) \in \mathcal{T}$ (or a random subset $\mathcal{T}' \subset \mathcal{T}$)
        - Update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} L(f_\theta(\boldsymbol{x}); \boldsymbol{d})$

The Stochastic Gradient Descent (SGD)

Denoting $\mathcal{T}$ the training dataset, the loss functions are of the form

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} L(f_\theta(\boldsymbol{x}^i); \boldsymbol{d}^i)$$

where f_θ is the neural network.

Example: $E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} \|f_\theta(\boldsymbol{x}^i) - \boldsymbol{d}^i)\|^2$

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

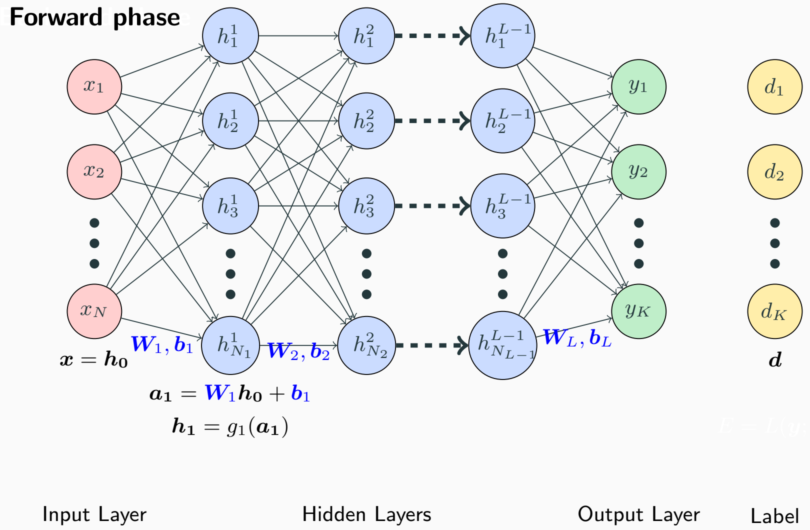
- Initialize $\boldsymbol{\theta}_0$ randomly
- For $0 \leqslant k \leqslant N$,
    - For all $(\boldsymbol{x}, \boldsymbol{d}) \in \mathcal{T}$ (or a random subset $\mathcal{T}' \subset \mathcal{T}$)
        - Update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} L(f_\theta(\boldsymbol{x}); \boldsymbol{d})$

An iteration overall the dataset is called an epoch.

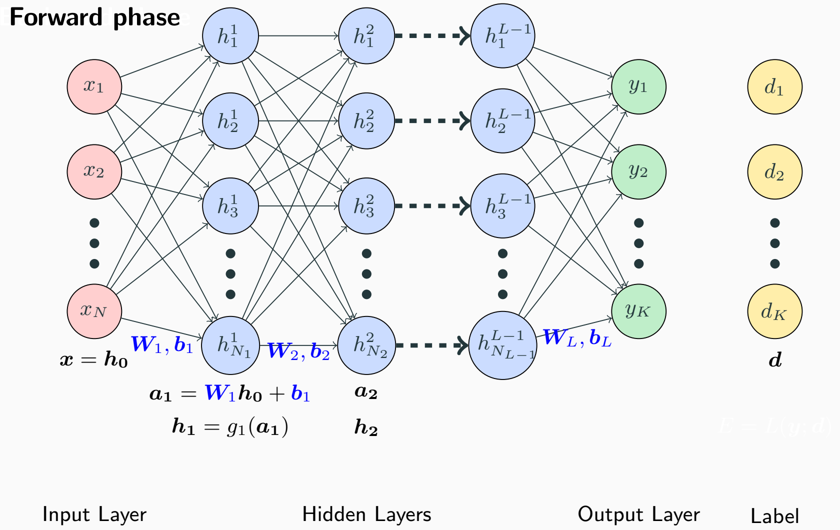
How is computed the gradient for the SGD ?

How is computed the gradient for the SGD ?

Error backpropagation

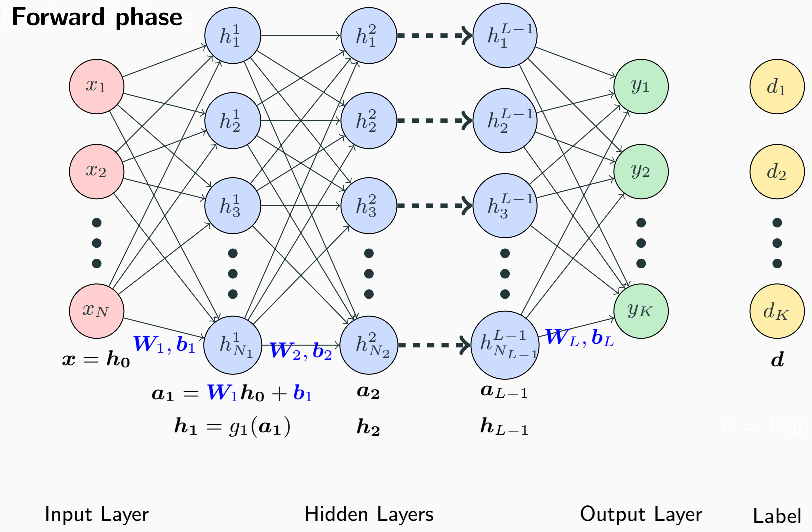


Error backpropagation

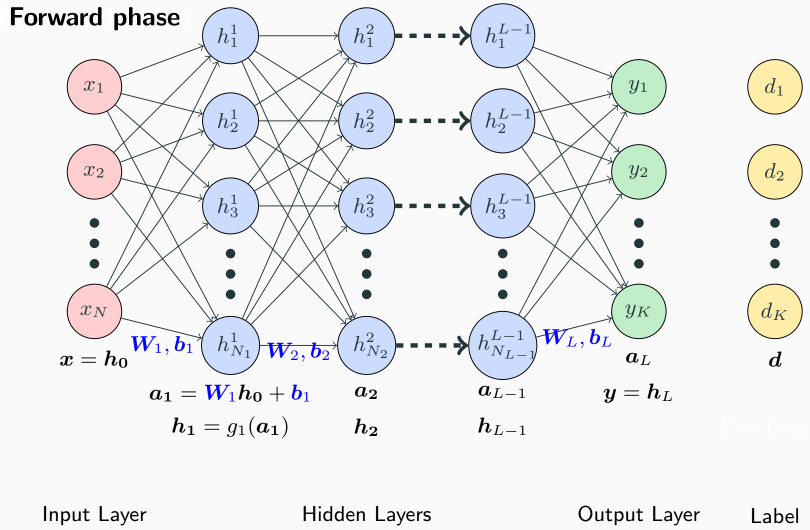


How is computed the gradient for the SGD ?

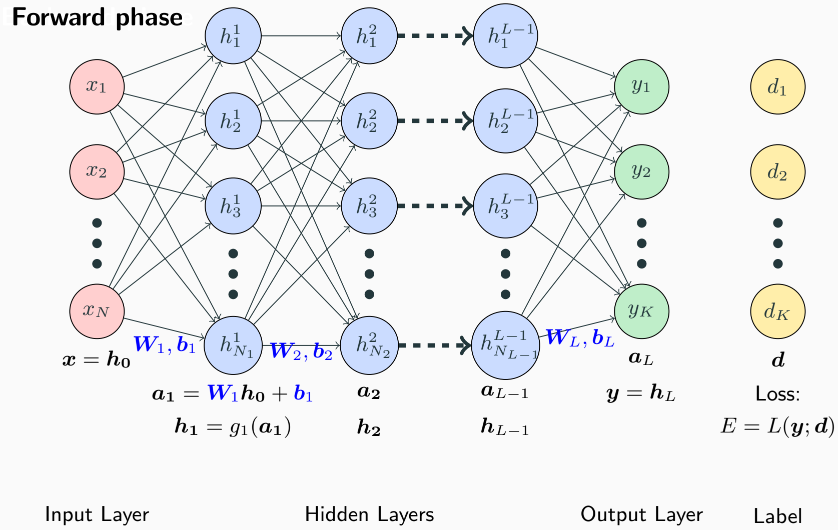
Error backpropagation



Error backpropagation

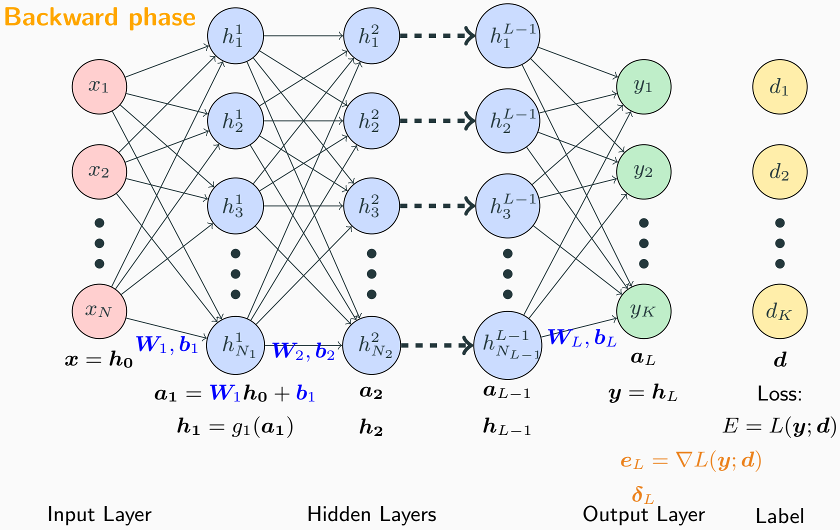


Error backpropagation



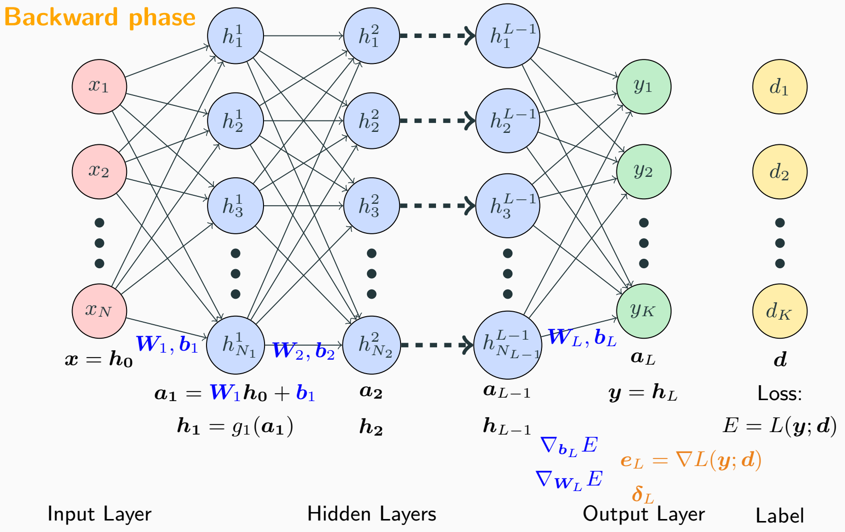
How is computed the gradient for the SGD ?

Error backpropagation

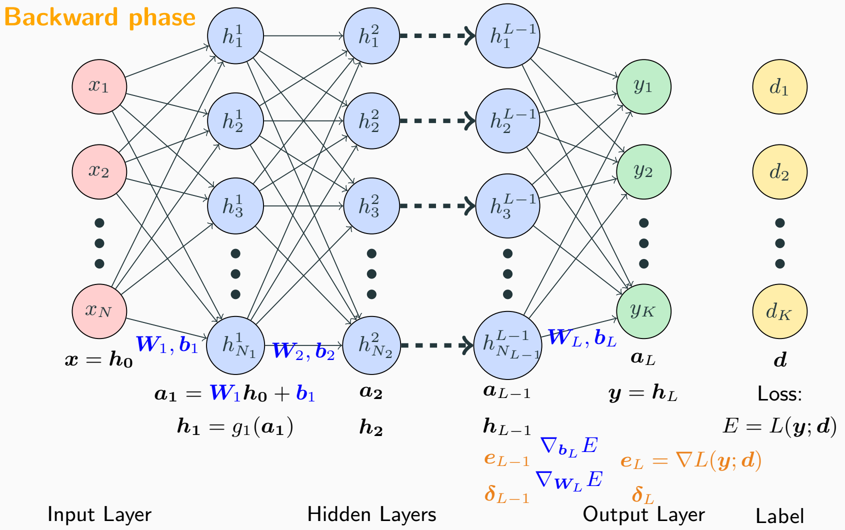


How is computed the gradient for the SGD ?

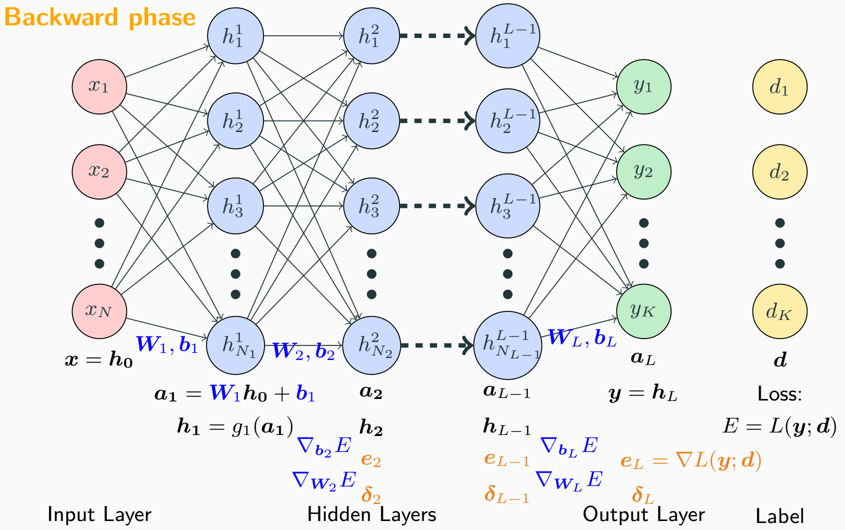
Error backpropagation



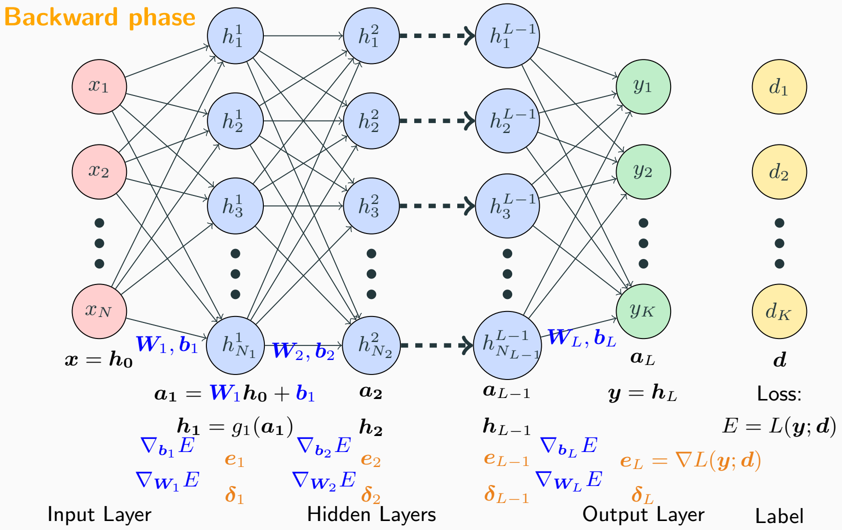
Error backpropagation



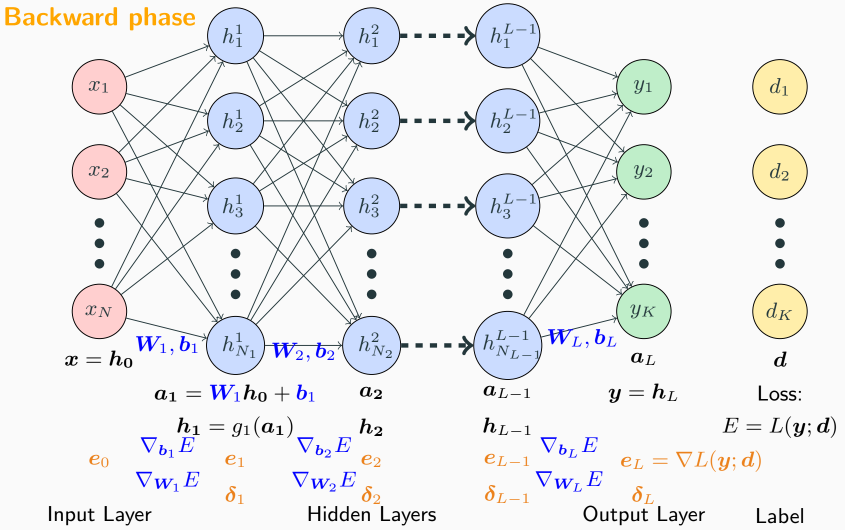
Error backpropagation



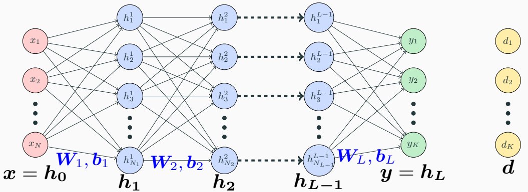
Error backpropagation



Error backpropagation



How is computed the gradient for the SGD ?



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Forward pass

Initialization:

$\boldsymbol{h}_0 = \boldsymbol{x}$

for layer $k = 1$ **to** L **do**

Linear unit:

$\boldsymbol{a}_k = \boldsymbol{W}_k \boldsymbol{h}_{k-1} + \boldsymbol{b}_k$

Componentwise non-linear activation:

$\boldsymbol{h}_k = g_k(\boldsymbol{a}_k)$

end

Output layer:

$\boldsymbol{y} = \boldsymbol{h}_L$

Compute loss:

$E = L(\boldsymbol{y}; \boldsymbol{d})$

Backward pass

Initialization: Gradient of output layer:

$\nabla_{\boldsymbol{h}_L} E = \nabla L(\boldsymbol{y}; \boldsymbol{d})$

for layer $k = L$ **to** 1 **do**

Componentwise gain of error:

$\boldsymbol{\delta}_k = \nabla_{\boldsymbol{a}_k} E = \nabla_{\boldsymbol{h}_k} E \odot g'_k(\boldsymbol{a}_k)$

Gradient of layer bias:

$\nabla_{\boldsymbol{b}_k} E = \boldsymbol{\delta}_k$

Gradient of weights:

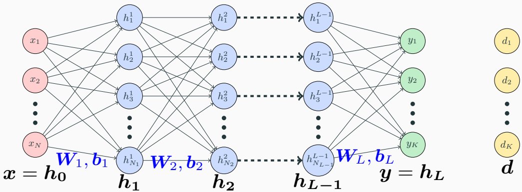
$\nabla_{\boldsymbol{W}_k} E = \boldsymbol{\delta}_k \boldsymbol{h}^T_{k-1}$

Gradient of previous hidden layer:

$\nabla_{\boldsymbol{h}_{k-1}} E = \boldsymbol{W}^T_k \boldsymbol{\delta}_k$

end

How is computed the gradient for the SGD ?



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Forward pass

Initialization:

$\boldsymbol{h}_0 = \boldsymbol{x}$

for layer $k = 1$ **to** L **do**

Linear unit:

$\boldsymbol{a}_k = \boldsymbol{W}_k \boldsymbol{h}_{k-1} + \boldsymbol{b}_k$ **(stored)**

Componentwise non-linear activation:

$\boldsymbol{h}_k = g_k(\boldsymbol{a}_k)$ **(stored)**

end

Output layer:

$\boldsymbol{y} = \boldsymbol{h}_L$

Compute loss:

$E = L(\boldsymbol{y}; \boldsymbol{d})$

Backward pass

Initialization: Gradient of output layer:

$\nabla_{\boldsymbol{h}_L} E = \nabla L(\boldsymbol{y}; \boldsymbol{d})$

for layer $k = L$ **to** 1 **do**

Componentwise gain of error:

$\boldsymbol{\delta}_k = \nabla_{\boldsymbol{a}_k} E = \nabla_{\boldsymbol{h}_k} E \odot g_k'(\boldsymbol{a}_k)$

Gradient of layer bias:

$\nabla_{\boldsymbol{b}_k} E = \boldsymbol{\delta}_k$

Gradient of weights:

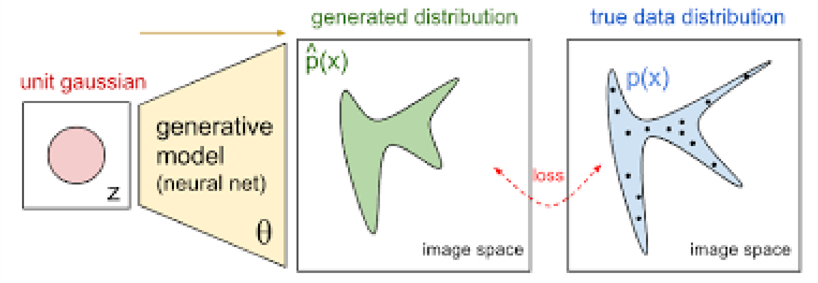
$\nabla_{\boldsymbol{W}_k} E = \boldsymbol{\delta}_k \boldsymbol{h}_{k-1}^T$

Gradient of previous hidden layer:

$\nabla_{\boldsymbol{h}_{k-1}} E = \boldsymbol{W}_k^T \boldsymbol{\delta}_k$

end

To the generative models



Questions?

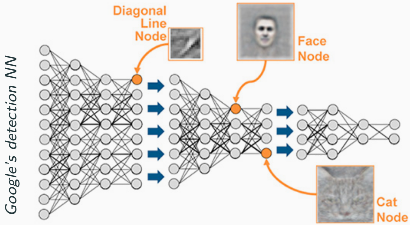
Sources, images courtesy and acknowledgment

Charles Deledalle **V. Lepetit** **L. Masuch**

Convolutional Neural Network (CNN)

What is deep learning?

- Representation learning using artificial neural networks
  - $\rightarrow$ Learning good features automatically from raw data.
  - $\rightarrow$ Exceptionally effective at learning patterns.
- Learning representations of data with multiple levels of abstraction
  - $\rightarrow$ hierarchy of layers that mimic the neural networks of our brain,
  - $\rightarrow$ cascade of non-linear transforms.



- If you provide the system with tons of information, it begins to understand it and responds in useful ways.

(Source: Caner Hazırbaş & Lucas Masuch)

Deep learning

How to teach a machine?

(or any other **hand-crafted** features)

Good representations are often very complex to define.

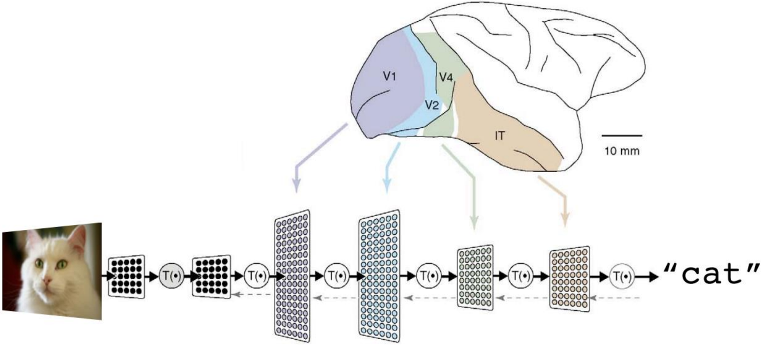
(Source: Caner Hazırbaş)

Inspired by the Brain

- The first hierarchy of neurons that receives information in the visual cortex are sensitive to specific edges while brain regions further down the visual pipeline are sensitive to more complex structures such as faces.
- Our brain has lots of neurons connected together and the strength of the connections between neurons represents long term knowledge.
- One learning algorithm hypothesis: all significant mental algorithms are learned except for the learning and reward machinery itself.

(Source: Lucas Masuch)

Deep learning – Basic architecture



A deep neural network consists of a hierarchy of layers, whereby each layer transforms the input data into more abstract representations (e.g. edge -> nose -> face). The output layer combines those features to make predictions.

(Source: Lucas Masuch)

Trainable feature hierarchy

- Hierarchy of representations with increasing levels of abstraction.
- Each stage is a kind of trainable feature transform.

Image recognition

- Pixel → edge → texton → motif → part → object

Text

- Character → word → word group → clause → sentence → story

Speech

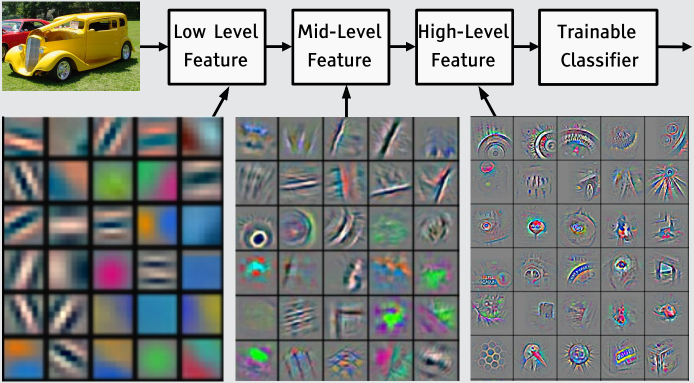
- Sample → spectral band → sound → ... → phone → phoneme → word

Deep Learning addresses the problem of learning hierarchical representations.

(Source: Yann LeCun & Marc'Aurelio Ranzato)

Deep learning – Feature hierarchy

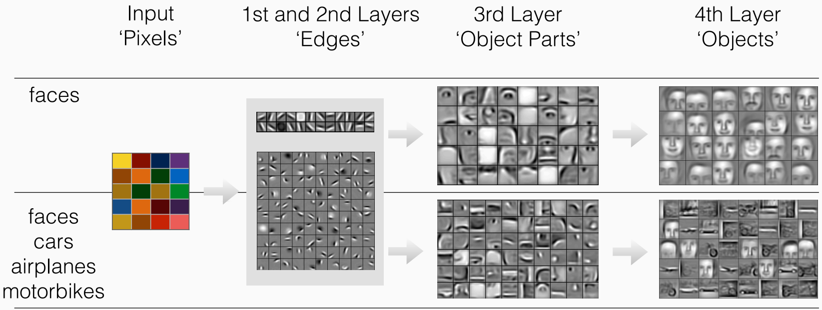
- It's deep if it has more than one stage of non-linear feature transformation



Feature visualization of convolutional net
trained on ImageNet from (Zeiler & Fergus, 2013)

(Source: Yann LeCun & Marc'Aurelio Ranzato)

Deep learning – Feature hierarchy



Each layer progressively extracts higher level features of the input until the final layer essentially makes a decision about what the input shows. The more layers the network has, the higher level features it will learn.

(Source: Andrew Ng & Lucas Masuch & Caner Hazırbaş)

Deep learning – Training

Today's trend: make it deeper and deeper

- 2012: 8 layers (AlexNet – Krizhevsky *et al.*, 2012)
- 2014: 19 layers (VGG Net – Simonyan & Zisserman, 2014)
- 2014: 22 layers (GoogLeNet – Szegedy *et al.*, 2014)
- 2015: 152 layers (ResNet – He *et al.*, 2015)
- 2016: 201 layers (DenseNet – Huang *et al.*, 2017)

But remember, with back-propagation:

- We got stuck at local optima or saddle points
- The learning time does not scale well
  - it is very slow for deep networks and can be unstable.

How did networks get so deep? First, why does backprop fail?

Deep learning – Gradient vanishing problems

Back-propagation and gradient vanishing problems

Update: $\boldsymbol{W}_k = \boldsymbol{W}_k - \gamma \nabla_{\boldsymbol{W}_k} E$ with $\nabla_{\boldsymbol{W}_k} E = \boldsymbol{\delta}_k \boldsymbol{h}_{k-1}^T$

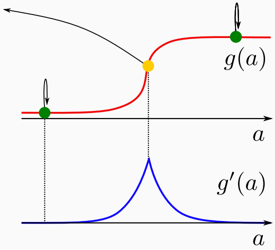
where $\boldsymbol{\delta}_k = \nabla_{\boldsymbol{a}_k} E = \nabla_{\boldsymbol{h}_k} E \odot g'_k(\boldsymbol{a}_k)$.

- With deep networks, the gradient vanishes quickly.
- Unfortunately, this arises even though we are far from a solution.
- The updates become insignificant, which leads to slow training rates.
- This strongly depends on the shape of $g'(a)$.
- The gradient may also explode leading to instabilities:
  $\rightarrow$ gradient exploding problem.

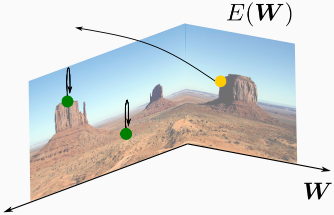
Deep learning – Gradient vanishing problem

As the network gets deeper, the landscape of E becomes:

- very hilly $\rightarrow$ lots of stationary points,
- with large plateaus $\rightarrow$ gradient vanishing problem,
- and delimited by cliffs $\rightarrow$ gradient exploding problem.



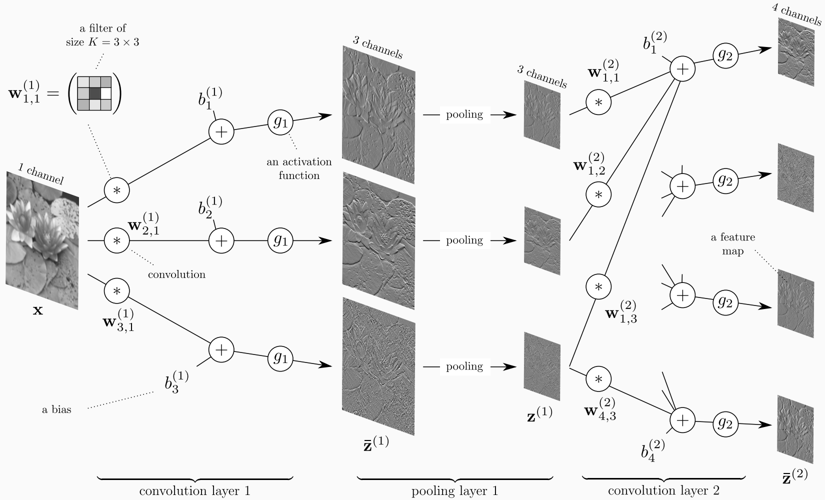
Activation function



Cost landscape

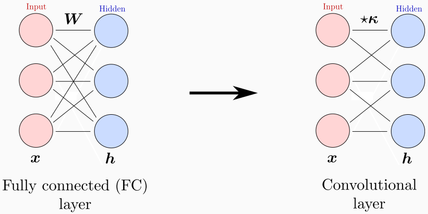
So, what has changed? (see later for recipes...)

CNN for image processing



What are CNNs?

- Essentially neural networks that use convolution in place of general matrix multiplications at least for the first layers.



- CNNs are designed to process the data in the form of multidimensional arrays/tensors (*e.g.*, 2D images, 3D video/volumetric images).
- Composed of series of stages: convolutional layers and pooling layers.
- Units connected to local regions in the feature maps of the previous layer.
- Do not only mimic the brain connectivity but also the visual cortex.

Convolutional neural networks

CNNs are composed of three main ingredients:

1. Local receptive fields
   - hidden units connected only to a small region of their input,
2. Shared weights
   - same weights and biases for all units of a hidden layer,
3. Pooling
   - condensing hidden layers.

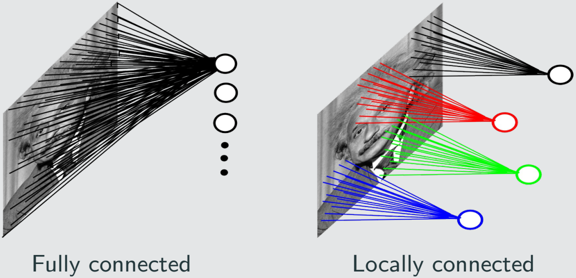
but also

4. Redundancy: more units in a hidden layer than inputs,
5. Sparsity: units should not all fire for the same stimulus.

All take inspiration from the visual cortex.

Local receptive fields → Locally connected layer

- Each unit in a hidden layer can see only a small neighborhood of its input,
- Captures the concept of spatiality.

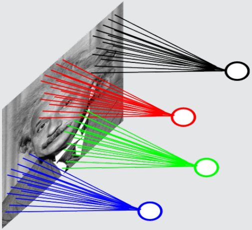


For a 200×200 image and 40,000 hidden units

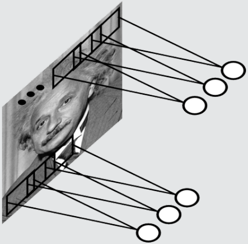
- Fully connected: 1.6 billion parameters,
- Locally connected (10×10 fields): 4 million parameters.

Self-similar receptive fields → Shared weights

- Detect features regardless of position (translation invariance),
- Use convolutions to learn simple input patterns.



Locally connected



Shared weights

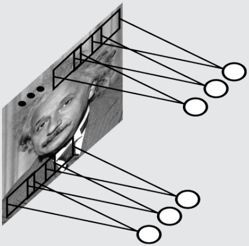
For a 200×200 image and 40,000 hidden units

- Locally connected (10×10 fields): 4 million parameters,
- & Shared weights: 100 parameters (independent of image size).

Convolutional neural networks

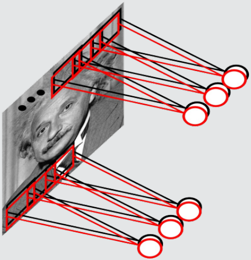
Specialized cells → Filter bank

- Use a filter bank to detect multiple patterns at each location,
- Multiple convolutions with different kernels,
- Result is a 3d array, where each slice is a feature map.



Shared weights

(1 input → 1 feature map)



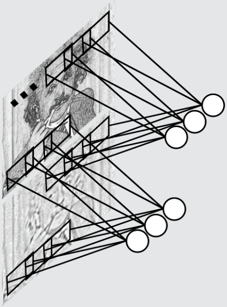
Filter bank

(1 input → 2 feature maps)

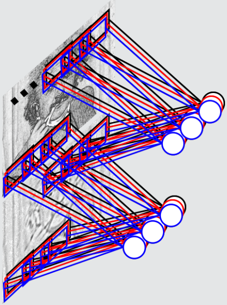
- 10×10 fields & 10 output features: 1,000 parameters.

Hierarchy → inputs of deep layers are themselves 3d arrays

- Learn to filter each channel such that their sum detects a relevant feature,
- Repeat as many times as the desired number of output features should be.



Multi-input filter
(2 inputs → 1 feature map)

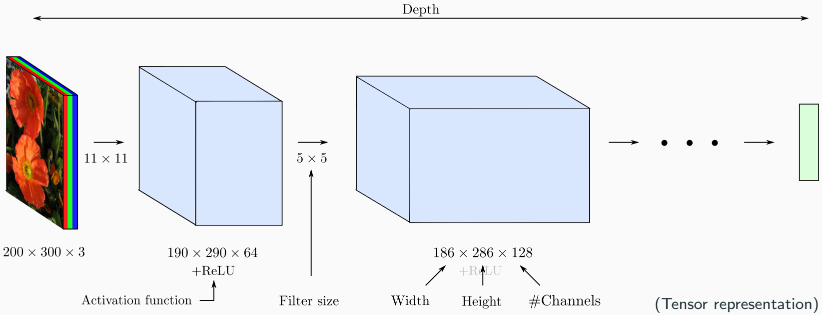


Multi-input filter bank
(2 inputs → 3 feature maps)

- **Remark:** these are not 3d convolutions, but sums of 2d convolutions.

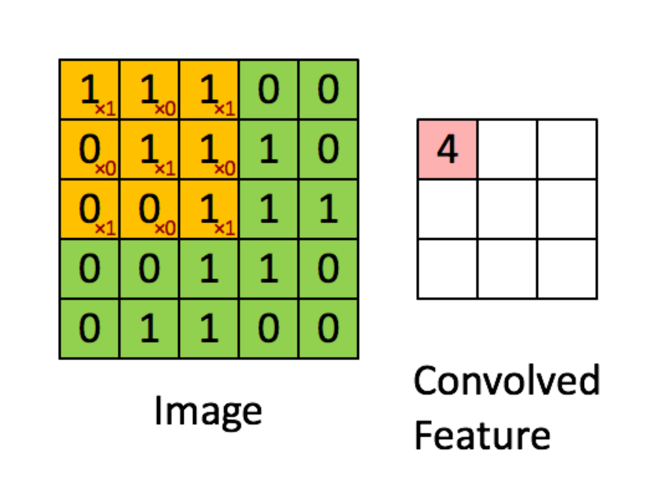
- 10×10 fields & 10 inputs & 10 outputs: 10,000 parameters.

Overcomplete → increase the number of channels

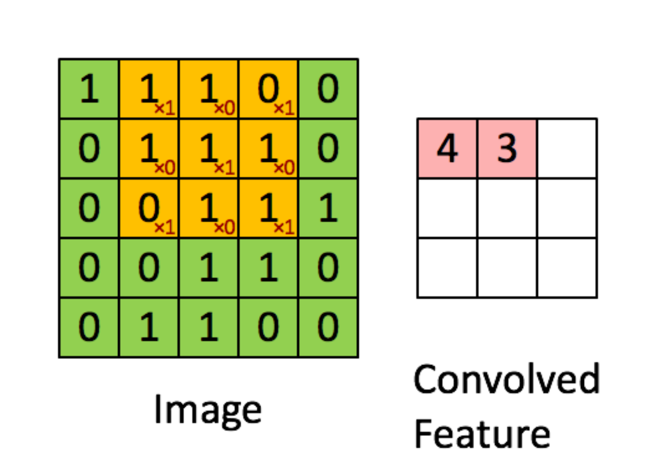


- **Redundancy**: increase the number of channels between layers.
- **Padding**: $n \times n$ conv + *valid* → width and height decrease by $n - 1$.
- Can we control even more the number of simple cells?

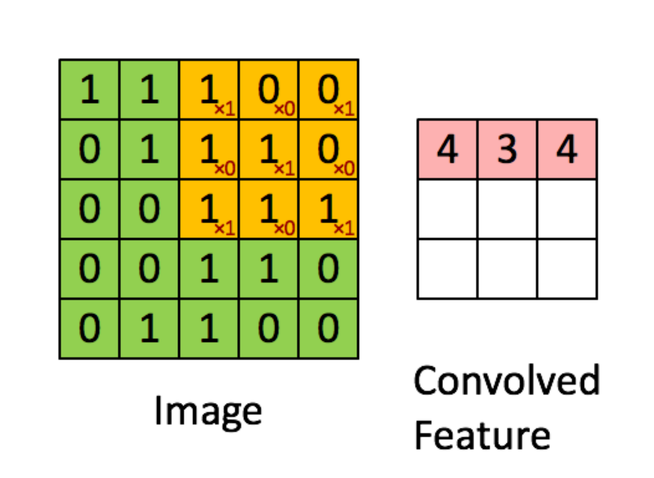
Convolutions



+bias

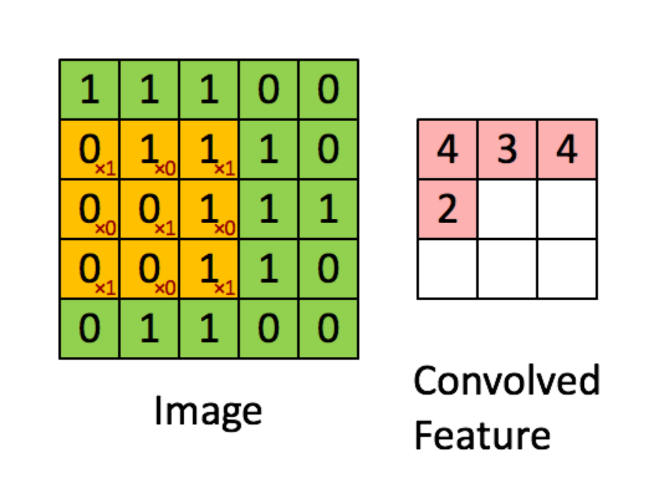

1
1 ×1
1 ×0
0 ×1
0
0
1 ×0
1 ×1
1 ×0
0
0
0 ×1
1 ×0
1 ×1
1
0
0
1
1
0
0
1
1
0
0
Image
4
3
Convolved
Feature

+bias

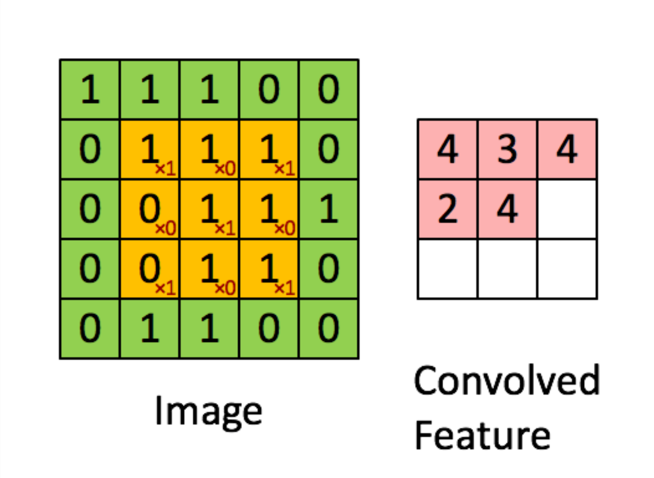

1
1
1 ×1
0 ×0
0 ×1
0
1
1 ×0
1 ×1
0 ×0
0
0
1 ×1
1 ×0
1 ×1
0
0
1
1
0
0
1
1
0
0
Image
4
3
4
Convolved Feature

+bias

Convolutions

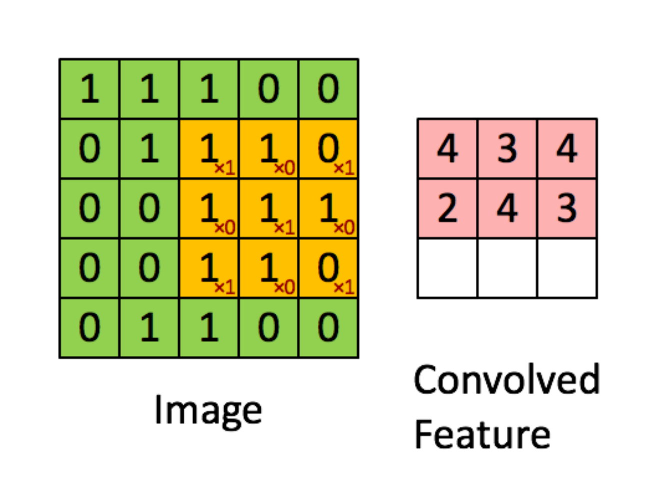


+bias


1 1 1 0 0
0 1×1 1×0 1×1 0
0 0×0 1×1 1×0 1
0 0×1 1×0 1×1 0
0 1 1 0 0
Image
4 3 4
2 4
Convolved Feature

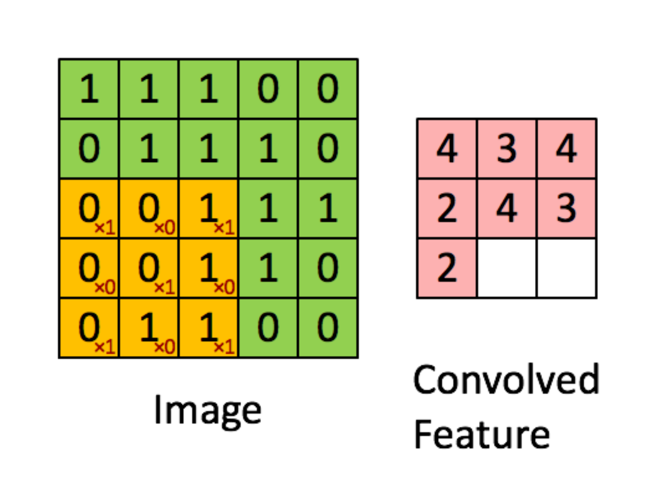
+bias

Convolutions



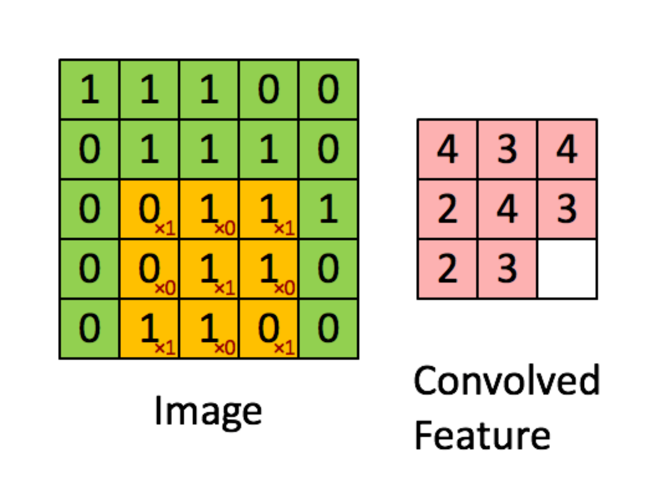
+bias

Convolutions

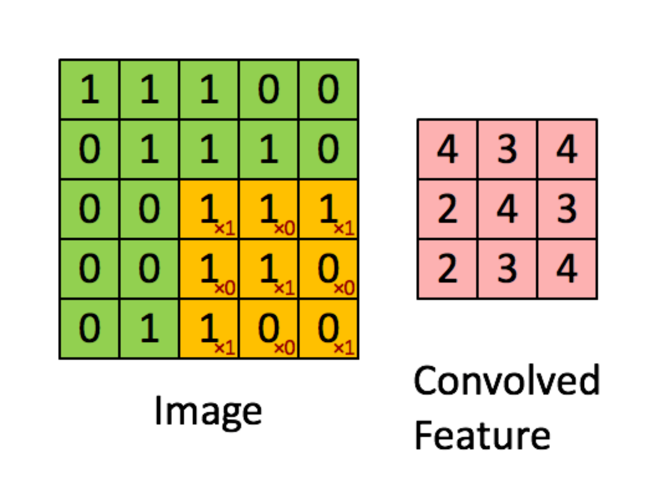


+bias

Convolutions

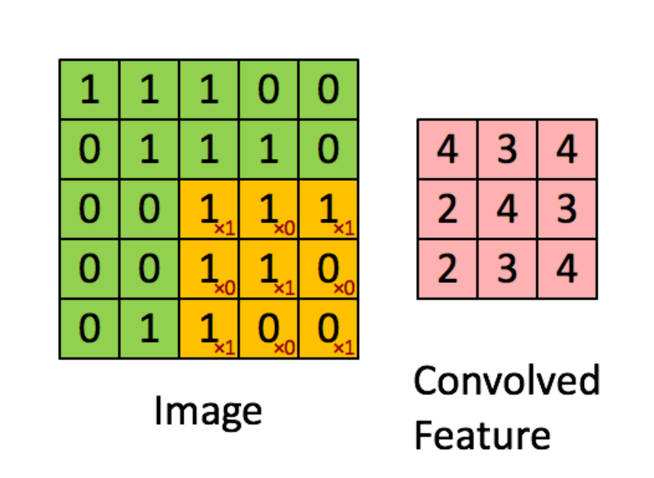


+bias


1 1 1 0 0
0 1 1 1 0
0 0 1 ×1 1 ×0 1 ×1
0 0 1 ×0 1 ×1 0 ×0
0 1 1 ×1 0 ×0 0 ×1
Image
4 3 4
2 4 3
2 3 4
Convolved Feature

$+$bias

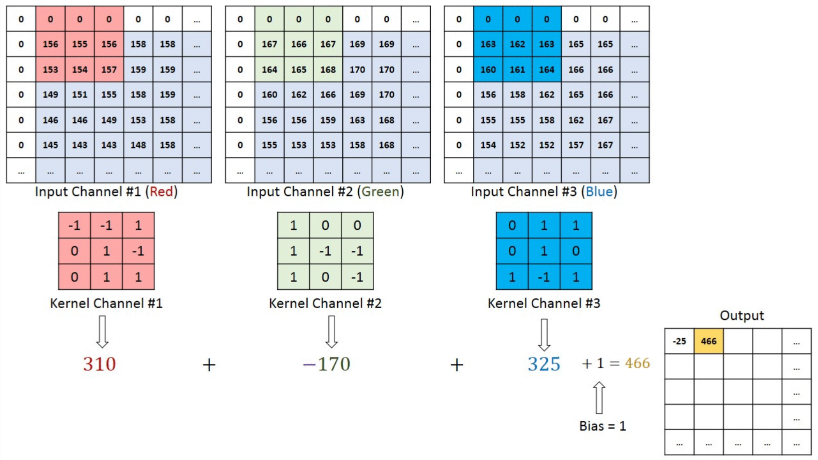
Convolutions



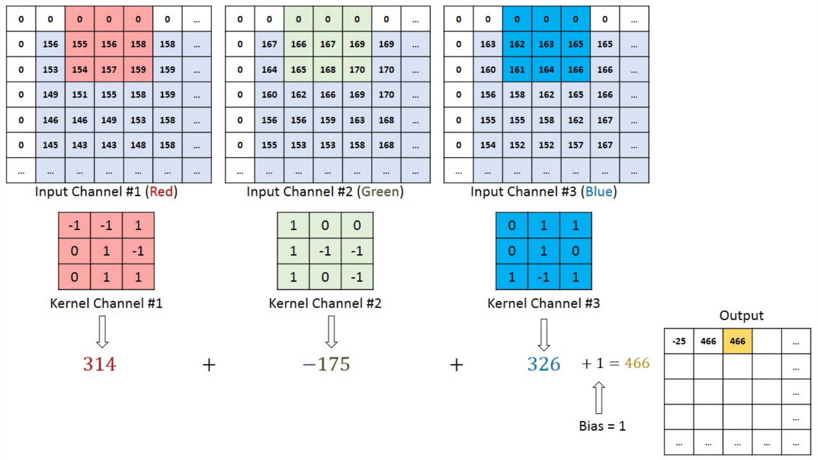
+bias

→ For an input size (M, N), the output size of convolution by a kernel of size $2k+1$ is $(M-2k, N-2k)$

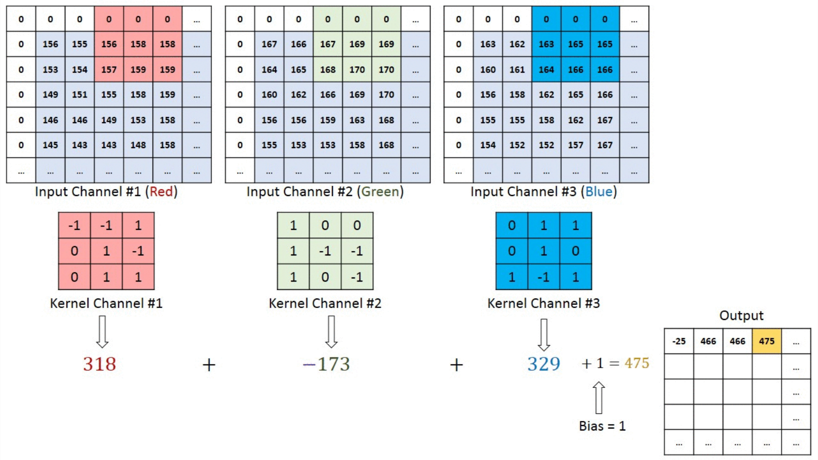
Convolutions with channels



Convolutions with channels



Convolutions with channels



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

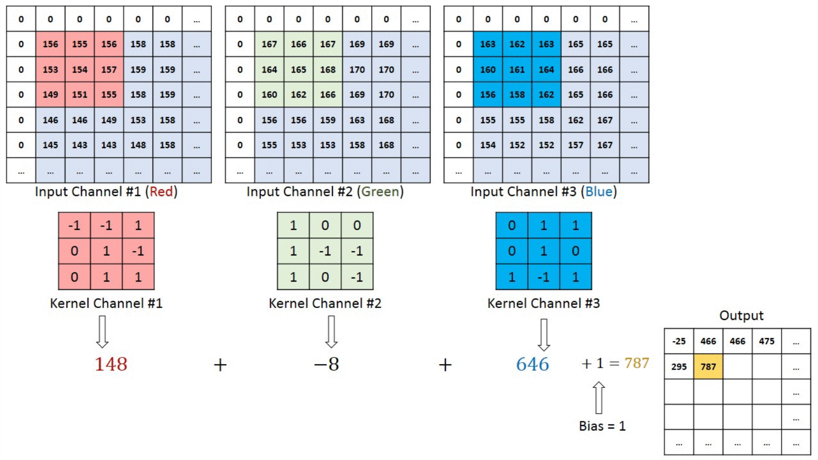
$$298 \quad + \quad -491 \quad + \quad 487 \quad +1 = 295$$

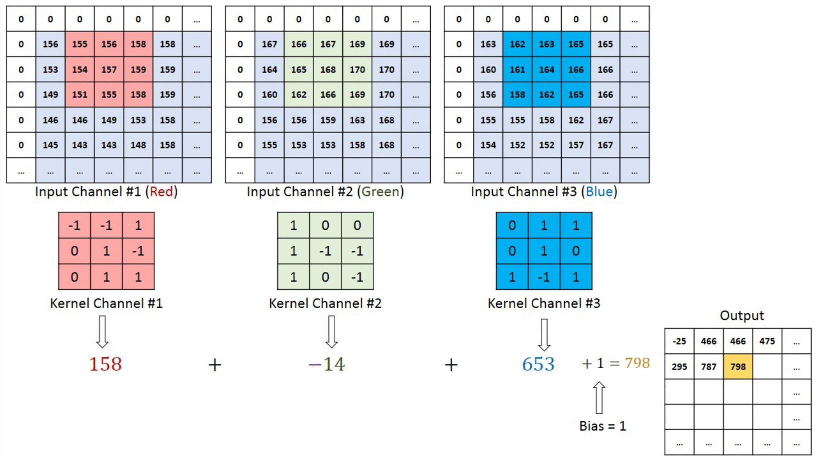
Bias = 1

Output

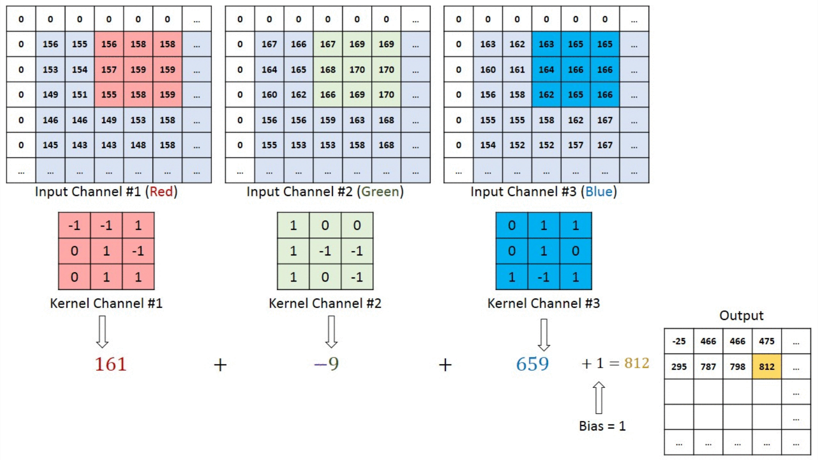
-25	466	466	475	...
295				...
				...
				...
...	...	...	...	...

Convolutions with channels

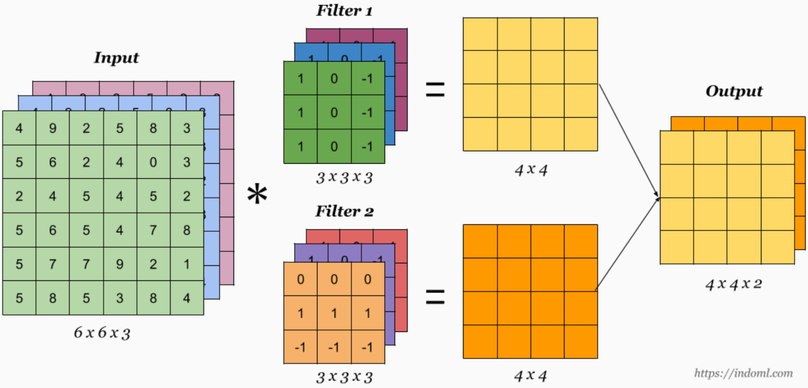



Input Channel #1 (Red)
Input Channel #2 (Green)
Input Channel #3 (Blue)
Kernel Channel #1
Kernel Channel #2
Kernel Channel #3
158
+
−14
+
653
+ 1 = 798
Bias = 1
Output

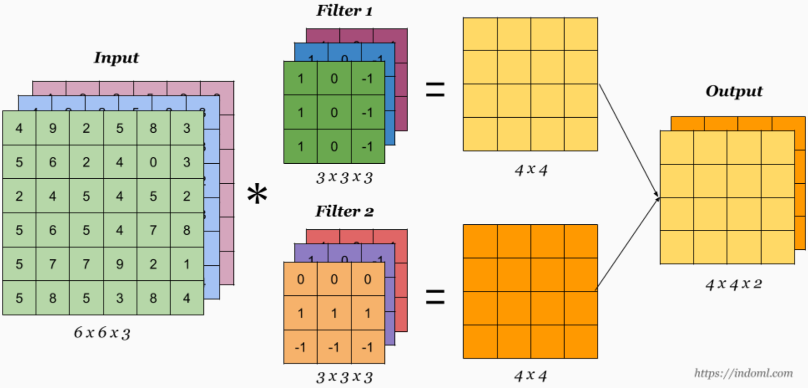
Convolutions with channels



Convolutions with several out channels



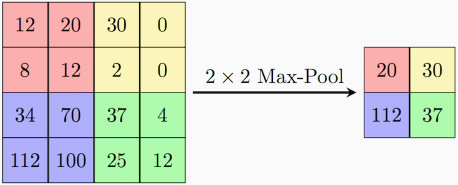
Convolutions with several out channels



These convolutions are of the form $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ but the number of parameters is the size of the filters/kernels.

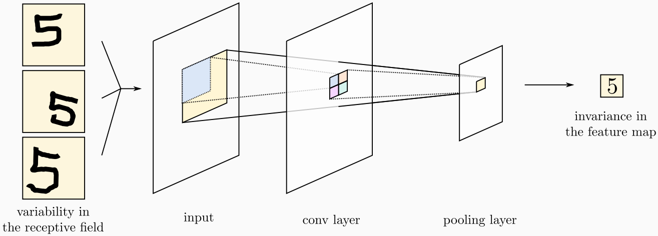
Pooling layer

- Used after each convolution layer to mimic complex cells,
- Unlike striding, reduce the size by aggregating inputs:
  - Partition the image in a grid of $z \times z$ windows (usually $z = 2$),
  - max-pooling: take the $\max$ in the window



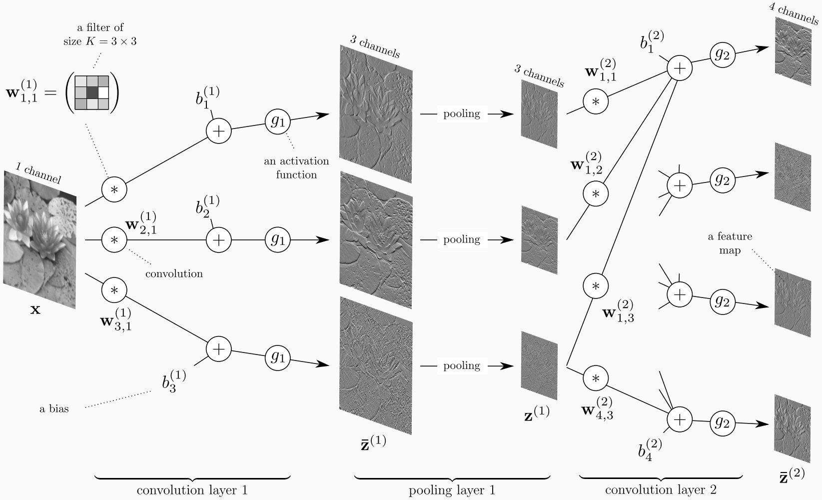
  - no parameters to learn

Pooling layer

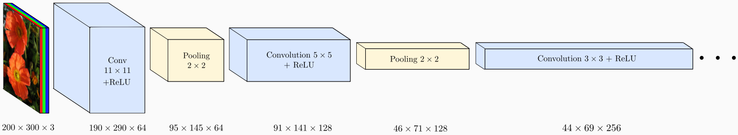


- Makes the output unchanged even if the input is a little bit changed,
- Allows some invariance/robustness with respect to the exact position,
- Simplifies/Condenses/Summarizes the output from hidden layers,
- Increases the effective receptive fields (with respect to the first layer.)

All concepts together



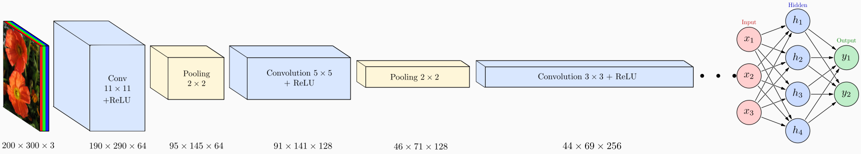
All concepts together with tensor representation



CNN: Alternate:

Conv + ReLU + pooling

All concepts together with tensor representation



CNN: Alternate:

Conv + ReLU + pooling

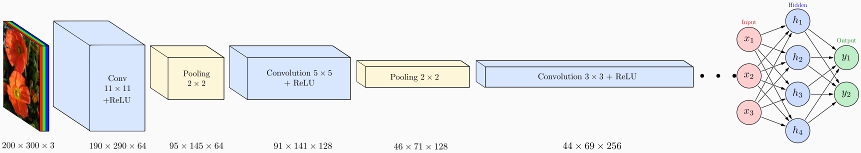
End of network:

Plug a standard neural network:

Fully connected hidden layers

(linear) + ReLU

All concepts together with tensor representation



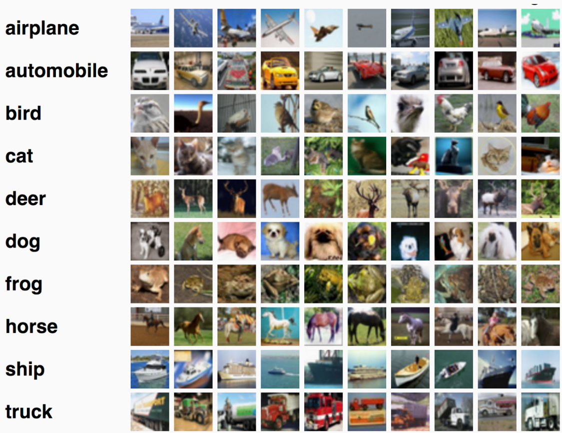
CNN: Alternate:
Conv + ReLU + pooling

End of network:
Plug a standard neural network:
Fully connected hidden layers (linear) + ReLU

Full network:

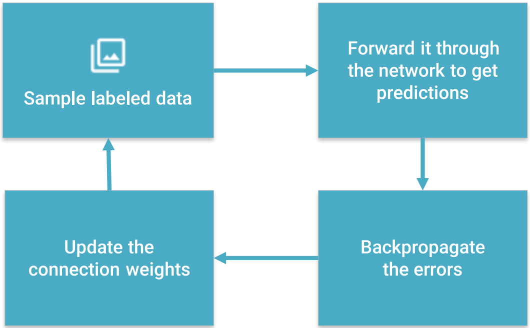
- **CNN:** Extract features specific to spatial data
- **Fully connected part:** Use CNN features for specific regression/classification task
- **Training:** Learn regression/classification and feature extraction **jointly**

Our goal: Image classification



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

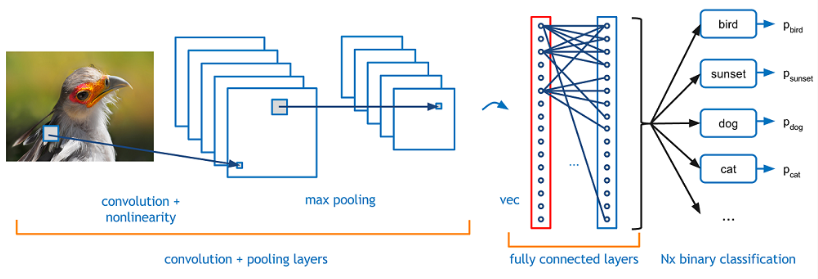
Training process



Learns by generating an error signal that measures the difference between the predictions of the network and the desired values and then using this error signal to change the weights (or parameters) so that predictions get more accurate.

(Source: Lucas Masuch)

Our goal



convolution +
nonlinearity
max pooling
vec
...
bird
sunset
dog
cat
...
p bird
p sunset
p dog
p cat
convolution + pooling layers
fully connected layers
Nx binary classification

Our goal

We want a CNN f_θ such that for a given input image $\boldsymbol{x}$:

$$f_\theta(\boldsymbol{x}) = \begin{pmatrix} \mathbb{P}\left(\boldsymbol{x} \in C_1\right) \\ \mathbb{P}\left(\boldsymbol{x} \in C_2\right) \\ \vdots \\ \mathbb{P}\left(\boldsymbol{x} \in C_n\right) \end{pmatrix}$$

Our goal

We want a CNN f_θ such that for a given input image $\boldsymbol{x}$:

$$f_\theta(\boldsymbol{x}) = \begin{pmatrix} \mathbb{P}\left(\boldsymbol{x} \in C_1\right) \\ \mathbb{P}\left(\boldsymbol{x} \in C_2\right) \\ \vdots \\ \mathbb{P}\left(\boldsymbol{x} \in C_n\right) \end{pmatrix}$$

The last layer will have an activation function softmax such that:

$$\text{softmax}\left(z_i\right) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \approx \mathbb{P}(x \in C_i)$$

Our goal

We want a CNN f_θ such that for a given input image $\boldsymbol{x}$:

$$f_\theta(\boldsymbol{x}) = \begin{pmatrix} \mathbb{P}\left(\boldsymbol{x} \in C_1\right) \\ \mathbb{P}\left(\boldsymbol{x} \in C_2\right) \\ \vdots \\ \mathbb{P}\left(\boldsymbol{x} \in C_n\right) \end{pmatrix}$$

The last layer will have an activation function softmax such that:

$$\text{softmax}\left(z_i\right) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \approx \mathbb{P}(x \in C_i)$$

The loss "Cross-entropy" is built to make that the training leads to this output. (see *multivariate regression* for explaination)

Conclusion: How to perform image classification ?

- Consider a labeled training dataset

Conclusion: How to perform image classification ?

- Consider a labeled training dataset
- Consider a CNN followed by a standard neural network (as shown before)

Conclusion: How to perform image classification ?

- Consider a labeled training dataset
- Consider a CNN followed by a standard neural network (as shown before)
- Train it to minimize the Cross-entropy loss via stochastic gradient descent (SGD).

Conclusion: How to perform image classification ?

- Consider a labeled training dataset
- Consider a CNN followed by a standard neural network (as shown before)
- Train it to minimize the Cross-entropy loss via stochastic gradient descent (SGD).
- Enjoy !
- Link to the tutorial

Questions?

Sources, images courtesy and acknowledgment

Charles Deledalle **V. Lepetit** **L. Masuch**