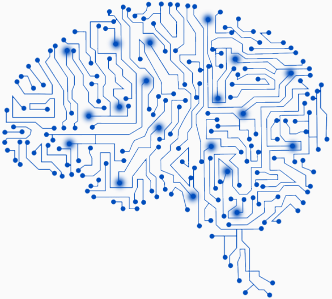
Collège doctoral Centre-Val de Loire

Introduction to Deep Learning

Course I – Introduction to Artificial Neural Networks

Bruno Galerne

2024-2025



Credits

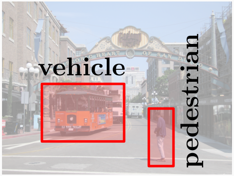
Most of the slides from **Charles Deledalle's** course "UCSD ECE285 Machine learning for image processing" (30 $\times$ 50 minutes course)

www.charles-deledalle.fr/

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

Computer Vision and Machine Learning

$\longrightarrow$



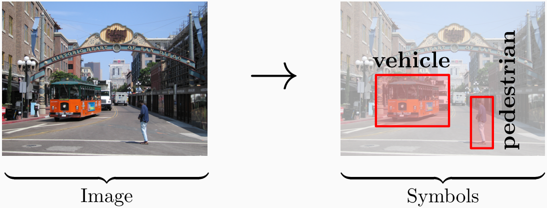
Image

Symbols

Computer vision – Artificial Intelligence – Machine Learning

Definition (The British Machine Vision Association)

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



CV is a subfield of Artificial Intelligence.

Definition (Oxford dictionary)

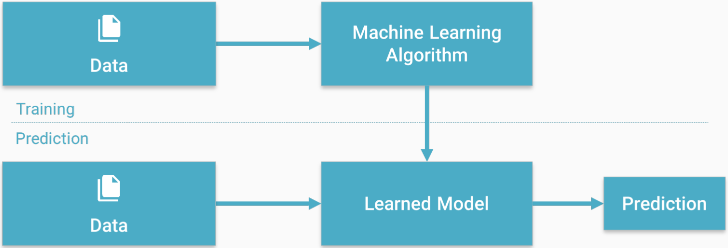
Artificial Intelligence, *noun*: the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation.

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.

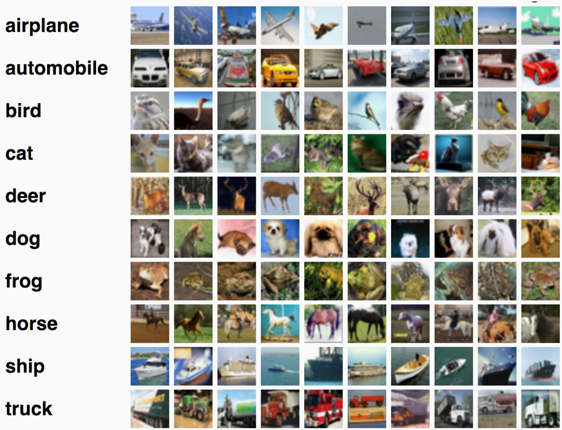
Definition

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:

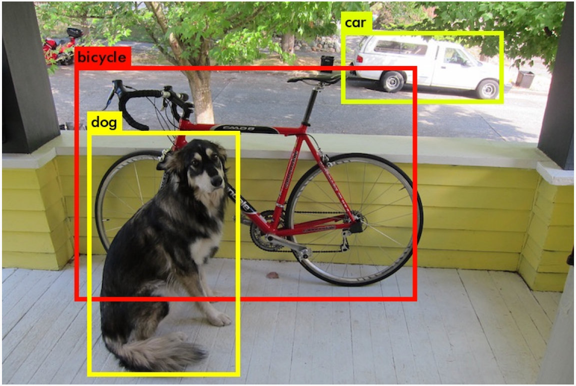


Computer vision – Image classification



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."

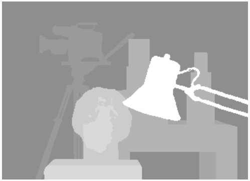
"woman is holding bunch of bananas."

"black cat is sitting on top of suitcase."

(Karpathy, Fei-Fei, CVPR, 2015)

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

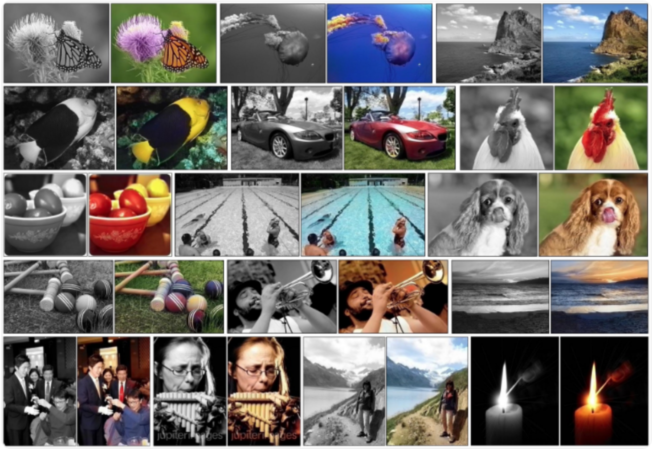
Computer vision – Depth estimation

 →  → 

(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)

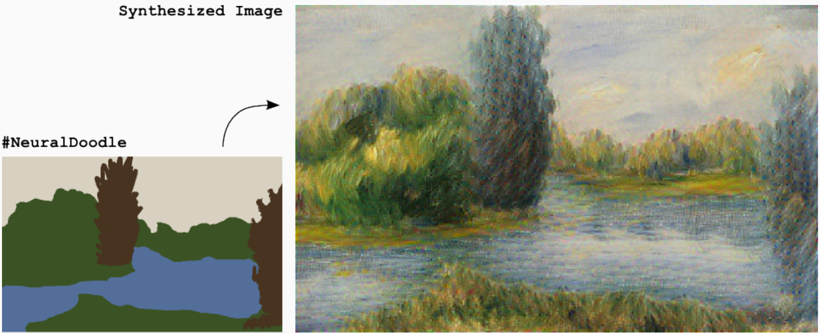
Goal: to add color to grayscale photographs.

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, ...*

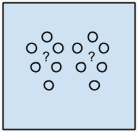
Training set: Set of data used to discover potentially predictive relationships.

Validation set: Set used to adjust the model hyperparameters.

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

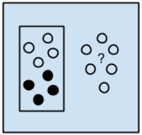
Label (Output): The class or outcome assigned to a sample. The actual prediction is often denoted by $\boldsymbol{y}$ and the desired/targeted class by $\boldsymbol{d}$ or $\boldsymbol{t}$. *Example: man/woman, wealth, education level, ...*

Learning approaches



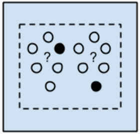
Unsupervised Learning Algorithms

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.*



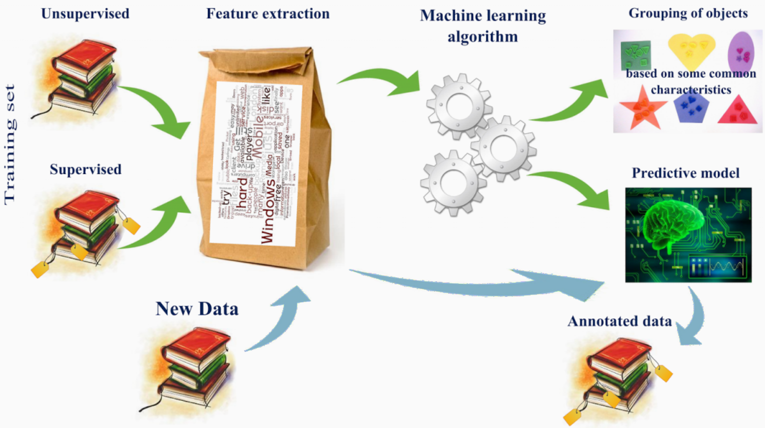
Semi-supervised Learning Algorithms

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)

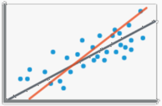
Machine learning workflow



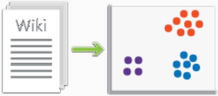
(Source: Michael Walker)

Problem types

Classification
(supervised – predictive)



Regression
(supervised – predictive)



Clustering
(unsupervised – descriptive)

Anomaly Detection
(unsupervised– descriptive)

(Source: Lucas Masuch)

What is deep learning?

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

(Source: Lucas Masuch)

Deep learning: Academic actors

- Popularized by Hinton in 2006 with Restricted Boltzmann Machines

Geoffrey Hinton: University of Toronto & Google

- Developed by different actors:

Yann LeCun: New York University & Facebook

Andrew Ng: Stanford & Baidu

Yoshua Bengio: University of Montreal

Jürgen Schmidhuber: Swiss AI Lab & NNAISENSE

and many others...

- Popularized by Hinton in 2006 with Restricted Boltzmann Machines

Geoffrey Hinton: University of Toronto & Google

- Developed by different actors:

Yann LeCun: New York University & Facebook

Andrew Ng: Stanford & Baidu

Yoshua Bengio: University of Montreal

Jürgen Schmidhuber: Swiss AI Lab & NNAISENSE

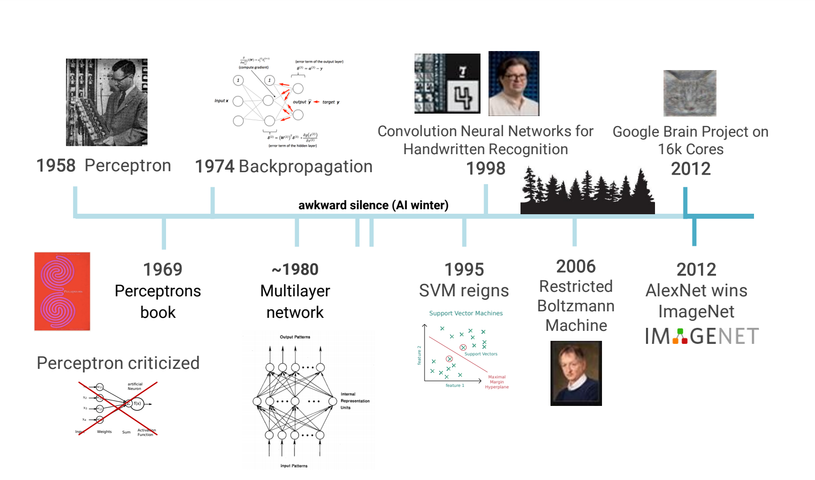
and many others...

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

Actors and applications

- Very active technology developed by big actors: Facebook/Meta (PyTorch), Google (Tensorflow, Kerras, JAX),...
- Success story for many different academic problems
    - Image processing
    - Computer vision
    - Speech recognition
    - Natural language processing
    - Translation
    - etc
- Today all industries wonder if AI/DL can improve their process.

Timeline of (deep) learning



(Source: Lucas Masuch & Vincent Lepetit)

Plan of the course

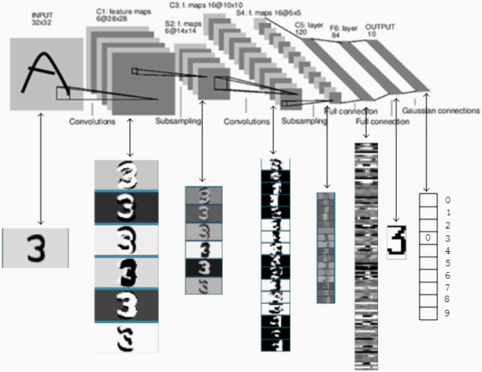
1. Introduction to neural networks
2. Convolutional neural networks for image classification
3. Deep CNN for image classification, transfer learning
4. Convolutional neural networks for image segmentation and image processing
5. Deep generative models
6. Transformers (if time allows)
7. Diffusion models (if time allows)

Software: Python + PyTorch using Google Colab.

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

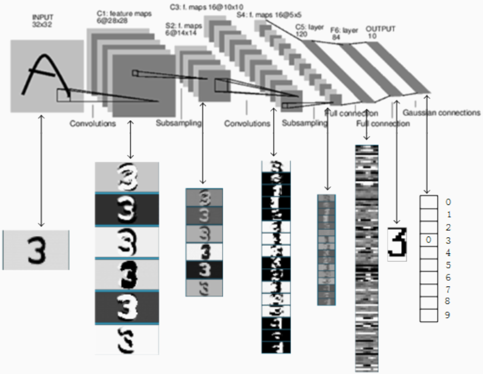
- Signal processing, speech recognition,...
- Text processing
- Graph processing (discrete geometry, social networks,...)
- Physics, chemistry,. . .

Neural networks for image classification



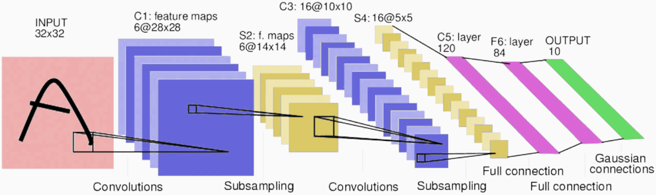
- **Goal:** Train a convolutional neural network for image classification

Neural networks for image classification

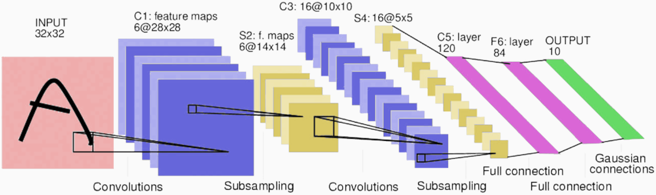


- **Goal:** Train a convolutional neural network for image classification
- **Goal: Understand the training** of a convolutional neural network for image classification

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

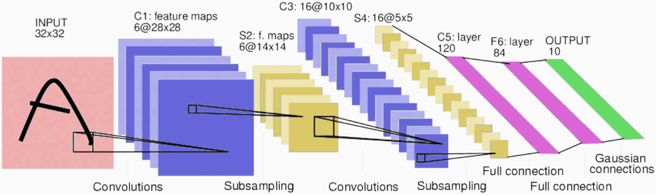


Understand the training of a convolutional neural network for image classification: **A lot of notions:** going backwards...



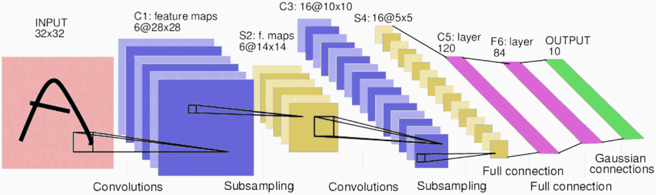
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.

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



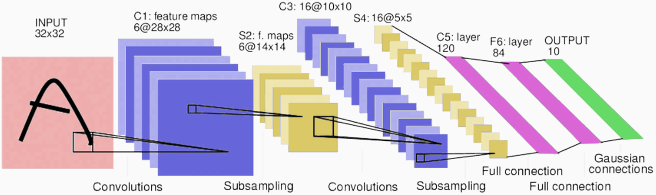
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.

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



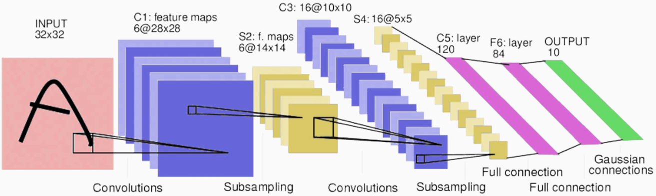
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done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

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



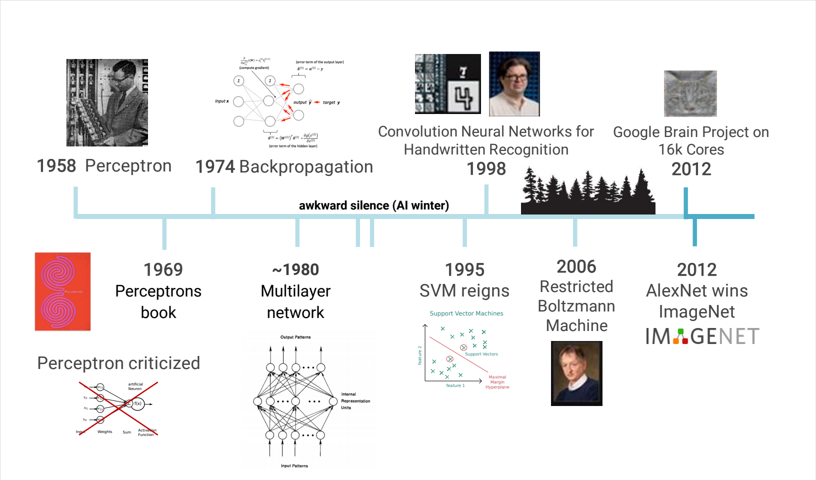
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done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

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



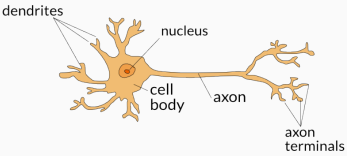
- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done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

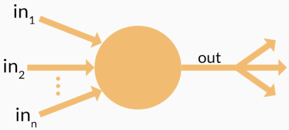
Timeline of (deep) learning



(Source: Lucas Masuch & Vincent Lepetit)

Perceptron


dendrites
nucleus
cell
body
axon
axon
terminals

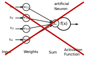

in_1
in_2
in_n
out

Perceptron

1958 Perceptron

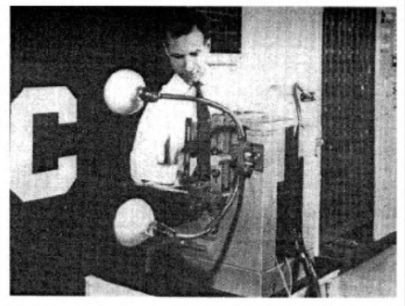
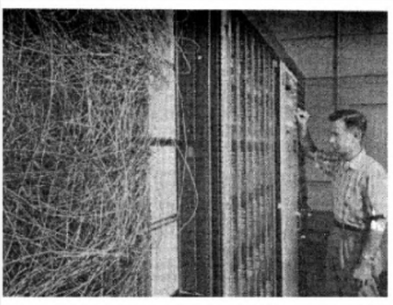
1969
Perceptrons
book

Perceptron criticized



(Source: Lucas Masuch & Vincent Lepetit)

Perceptron (Frank Rosenblatt, 1958)

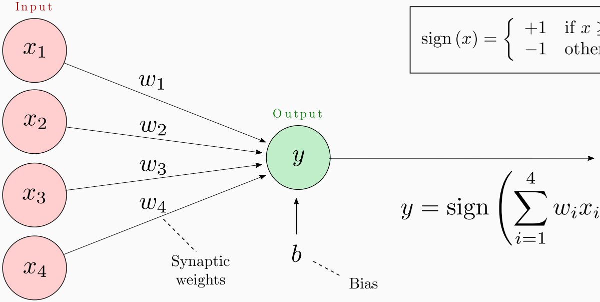
 

First binary classifier based on supervised learning (discrimination).

Foundation of modern artificial neural networks.

At that time: technological, scientific and philosophical challenges.

Representation of the Perceptron



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

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

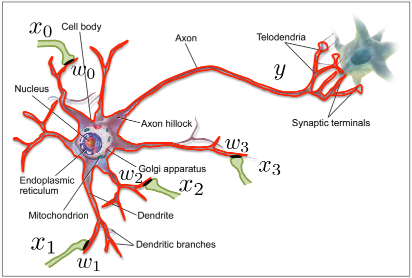
Parameters of the perceptron

$$\left.\begin{array}{l} \bullet\ \boldsymbol{w}_k\text{: synaptic weights} \\ \bullet\ b\text{: bias} \end{array}\right\} \longleftarrow \text{real parameters to be estimated.}$$

Training = adjusting the weights and biases

The origin of the Perceptron

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

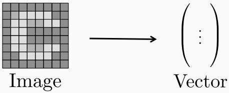


- When a neuron receives a stimulus with high enough voltage, it emits an action potential (aka, nerve impulse or spike). It is said to fire.
- The perceptron mimics this activation effect: it fires only when

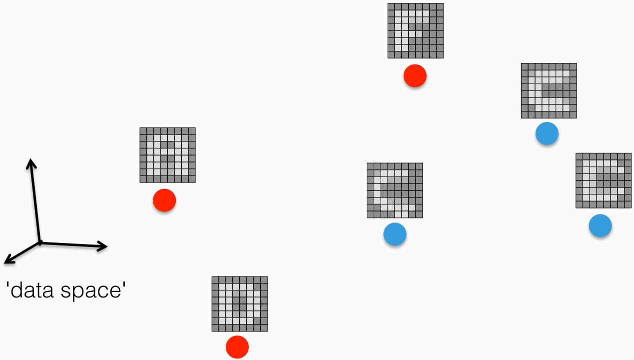
$$\sum_i w_i x_i + b > 0$$

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

1. Data are represented as vectors:



2. Collect training data with positive and negative examples:



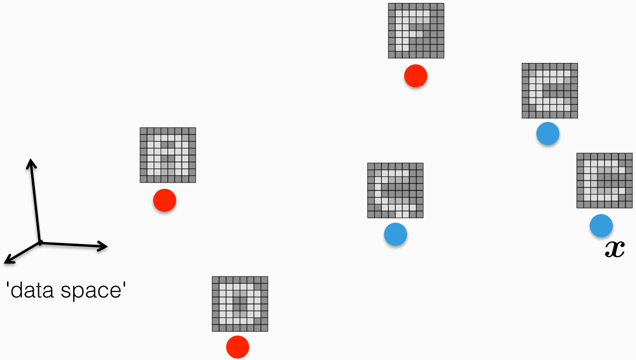
(Source: Vincent Lepetit)

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:

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



(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 $\boldsymbol{x}$.

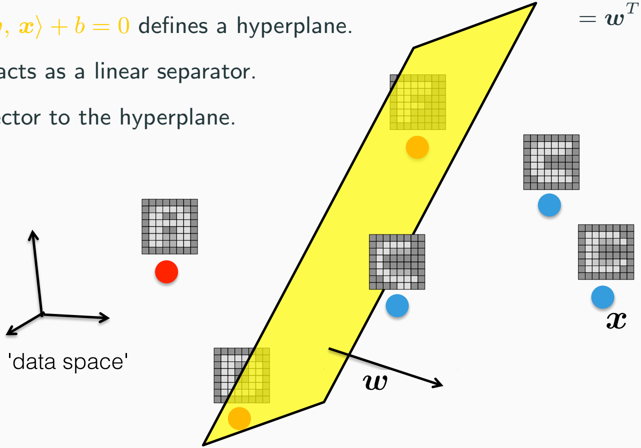
Dot product:

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

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

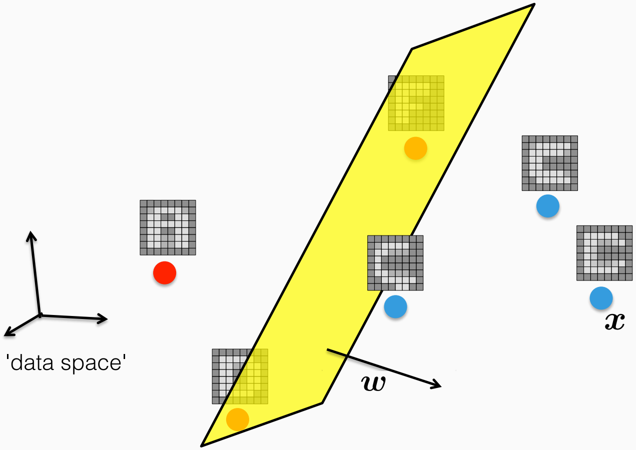
The hyperplane acts as a linear separator.

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



(Source: Vincent Lepetit)

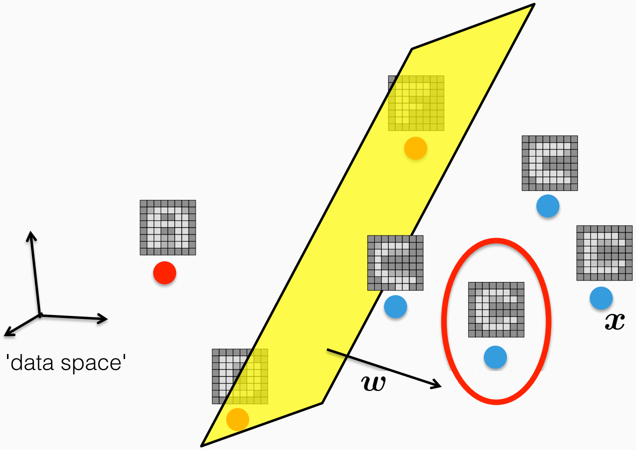
❹ **Testing:** the perceptron can now classify new examples.



(Source: Vincent Lepetit)

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

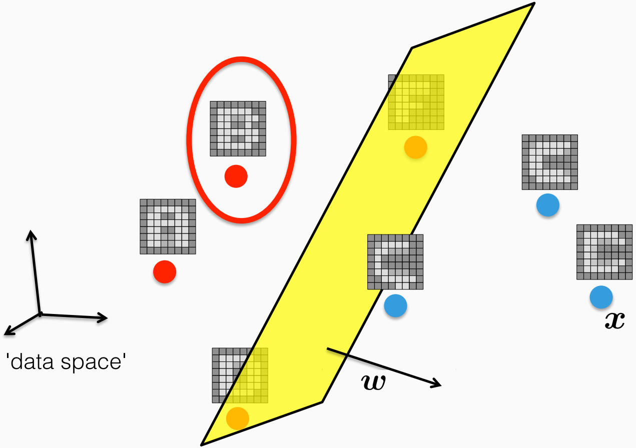
- A new example $\boldsymbol{x}$ is classified positive if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is positive,



(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.



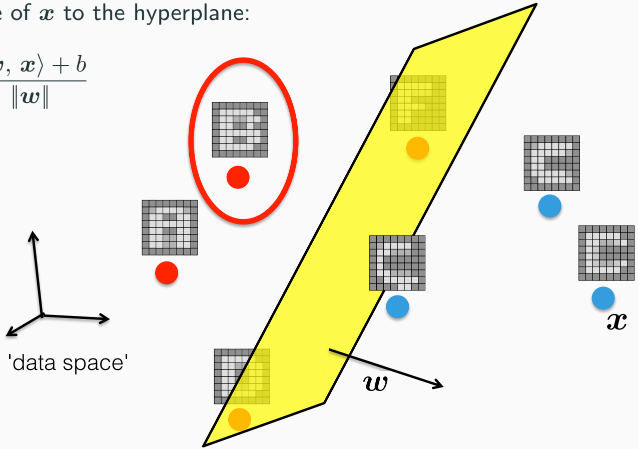
(Source: Vincent Lepetit)

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.

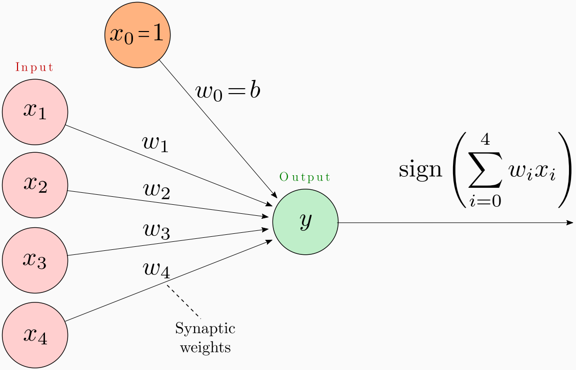
(signed) distance of $\boldsymbol{x}$ to the hyperplane:

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



(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 $\mathcal{T}$

$$\mathcal{T} = \{(\underbrace{\boldsymbol{x}^i}_{\substack{i\text{-th training}\\ \text{example}}}, \underbrace{d^i}_{\substack{\text{desired output}\\ \text{for sample } i\\ \{-1,+1\}}})\}_{\underbrace{i=1..N}_{\substack{\text{number of}\\ \text{training samples}}}}$$

How: minimize classification errors

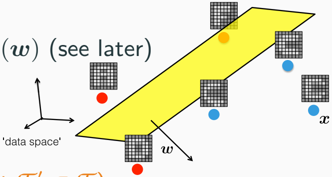
$$\min_{\boldsymbol{w}} E(\boldsymbol{w}) = -\sum_{\substack{(\boldsymbol{x},d)\in\mathcal{T}\\ \text{st } y\neq d}} d \times \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{(\boldsymbol{x},d)\in\mathcal{T}} \max(-d \times \langle \boldsymbol{w}, \boldsymbol{x} \rangle, 0)$$

- 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 $y \neq d$:

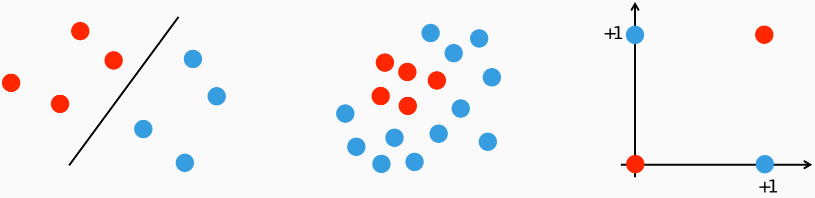
      Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma d \boldsymbol{x}$

- 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

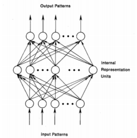
Artificial neural network

1958 Perceptron

1969
Perceptrons
book

Perceptron criticized

~1980
Multilayer
network



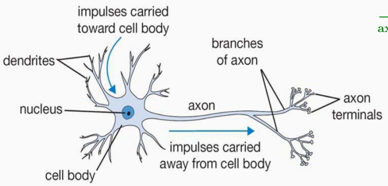
1989
Universal
Approximation
Theorem

(Source: Lucas Masuch & Vincent Lepetit)

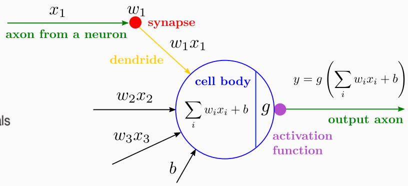
Artificial neural network

- 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.

Artificial neuron (McCulloch & Pitts, 1943)



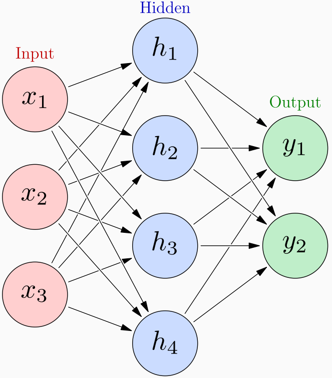
Biological neuron



Artificial neuron

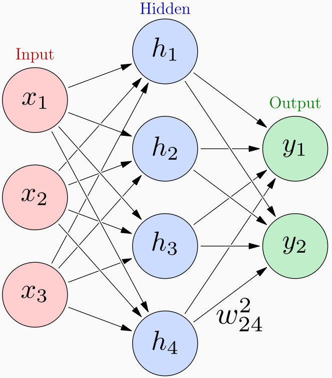
- An artificial neuron contains several incoming weighted connections, an outgoing connection and has a nonlinear activation function g.
- Neurons are trained to filter and detect specific features or patterns (e.g. edge, nose) by receiving weighted input, transforming it with the activation function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



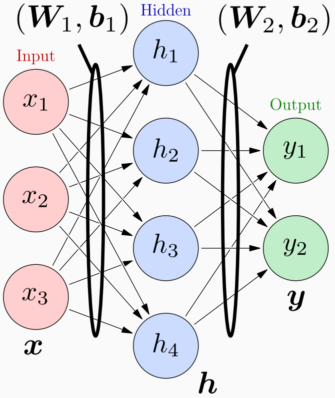
$$h_1 = g_1\left(w^1_{11}x_1 + w^1_{12}x_2 + w^1_{13}x_3 + b^1_1\right)$$
$$h_2 = g_1\left(w^1_{21}x_1 + w^1_{22}x_2 + w^1_{23}x_3 + b^1_2\right)$$
$$h_3 = g_1\left(w^1_{31}x_1 + w^1_{32}x_2 + w^1_{33}x_3 + b^1_3\right)$$
$$h_4 = g_1\left(w^1_{41}x_1 + w^1_{42}x_2 + w^1_{43}x_3 + b^1_4\right)$$

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

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



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

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

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

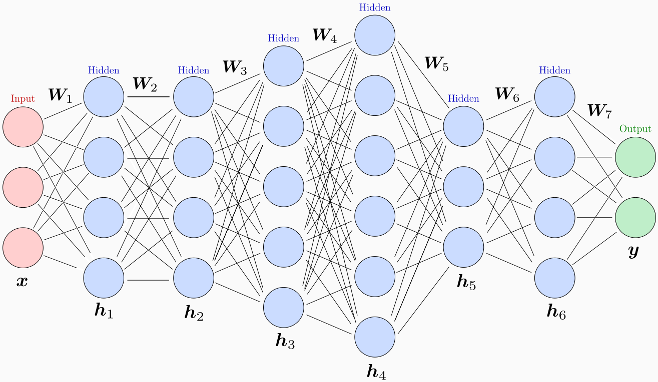
$$\boldsymbol{y} = g_2\left(\boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2\right)$$

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.

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: $g(a) = a$

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

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

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.

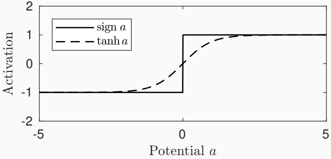
Activation functions

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 :

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

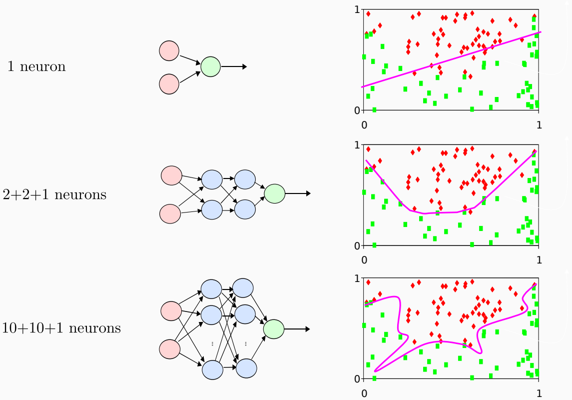
- Differentiable approximations of the sign and step functions, respectively.
- Act as threshold units for large values of $|a|$ and as linear for small values.

Machine learning – ANN

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



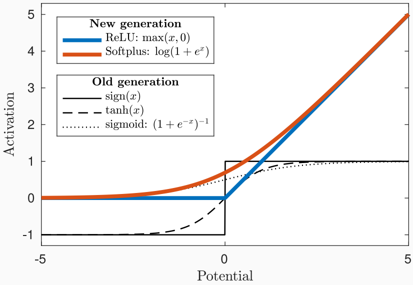
Separation
$\{\boldsymbol{x} \text{ s.t. } P(C_1|\boldsymbol{x}) = P(C_2|\boldsymbol{x})\}$

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

Activation functions

"Modern" units:

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

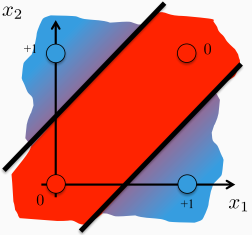


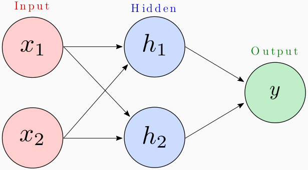
Most neural networks use ReLU (Rectifier linear unit) – $\max(a, 0)$ – nowadays for hidden layers, since it trains much faster, is more expressive than logistic function and prevents the gradient vanishing problem.

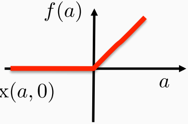
(Source: Lucas Masuch)

Neural networks solve non-linear separable problems

The x-or function







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

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

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

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

(Source: Vincent Lepetit)

Tasks, architectures and loss functions

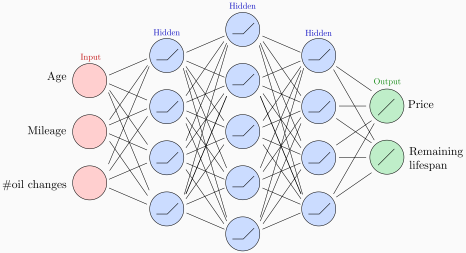
Approximation – Least square regression

- **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:

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

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

- **Typical architecture:**



- Hidden layer:

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

- Linear output:

$$g(a) = a$$

Approximation – Least square regression

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

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

  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)

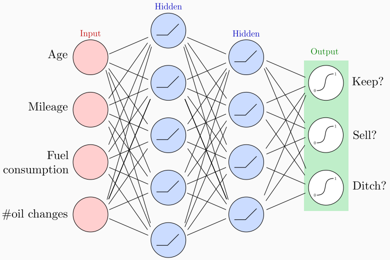
- **Goal:** Classify an object $\boldsymbol{x}$ into one among K classes $C_1, \ldots, C_K$.
- **How:** Estimate the coefficients $\boldsymbol{W}$ of a multivariate function

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

  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)

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

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

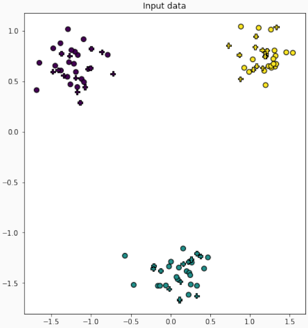
and look for $\boldsymbol{W}^*$ such that $\nabla E(\boldsymbol{W}^*) = 0$.

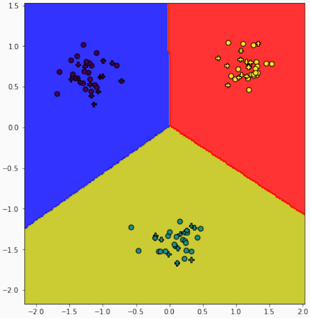
- **Solution:** Provided the network has enough flexibility and the size of the training set grows to infinity

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

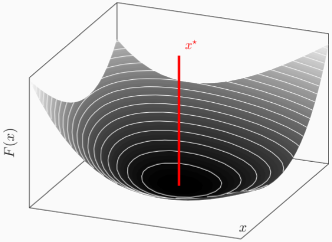
Multiclass classification – Multivariate logistic regression

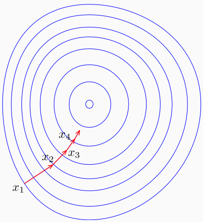
- If there is just one layer (no hidden layer), we get linear separation for multiple classes: Each class region is the intersection of half-spaces regions.

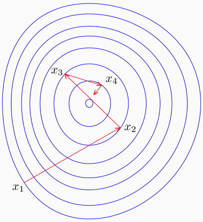




Gradient descent







- The parameters of the neural networks are obtained by minimizing the training loss.
- This is done using (variants of) the standard optimization algorithm: **Gradient descent**.
- Recall that the **gradient** of function $F : \mathbb{R}^d \to \mathbb{R}$ is the vector of all its partial derivatives:

$$\nabla F(x) = \begin{pmatrix} \frac{\partial F}{\partial x_1}(x_1, \dots, x_d) \\ \frac{\partial F}{\partial x_2}(x_1, \dots, x_d) \\ \vdots \\ \frac{\partial F}{\partial x_d}(x_1, \dots, x_d) \end{pmatrix}$$

- It gives the steepest direction (local direction towards maximal increase of F values).
- Gradient descent consists in moving in the opposite direction $-\nabla F(x)$.

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

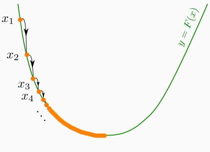
$$x^{(n+1)} = x^{(n)} \underbrace{- \gamma \nabla F(x^{(n)})}_{\text{direction of greatest descent}},$$

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

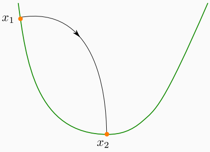
$$\nabla F(x^\star) = 0 \ .$$

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

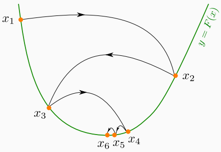
One dimension



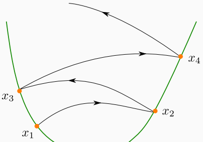
Small step size
Slow convergence



Good step size
Fast convergence

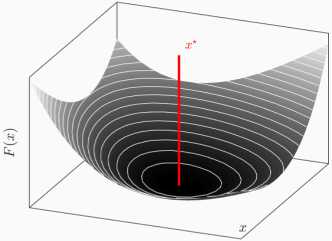


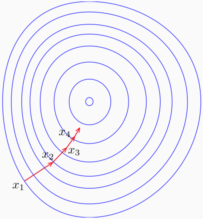
Large step size
Slow convergence

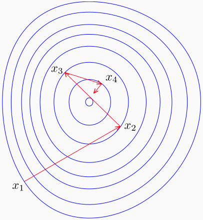


Too large step size
Divergence

Two dimensions

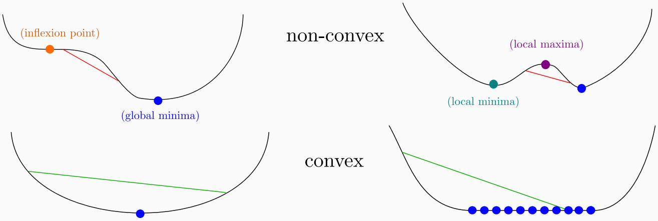




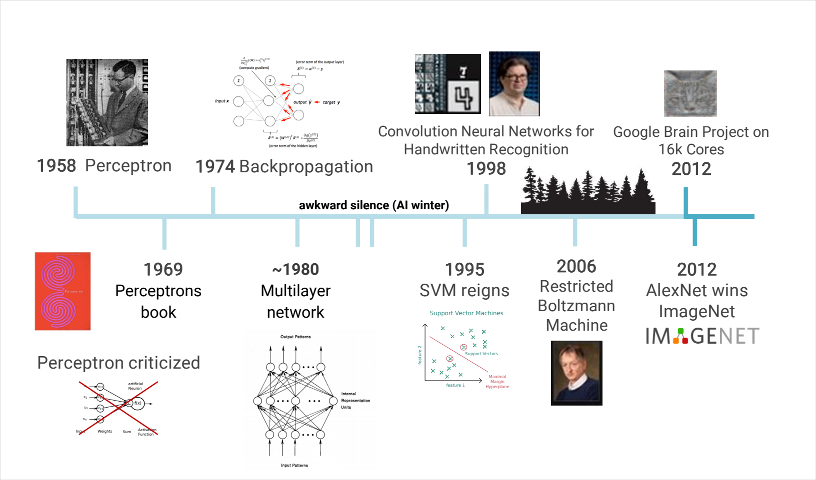


Non convexity in machine learning

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

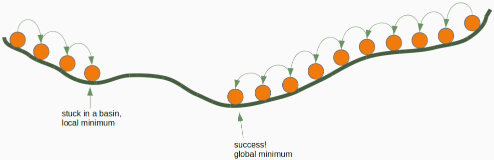


Timeline of (deep) learning

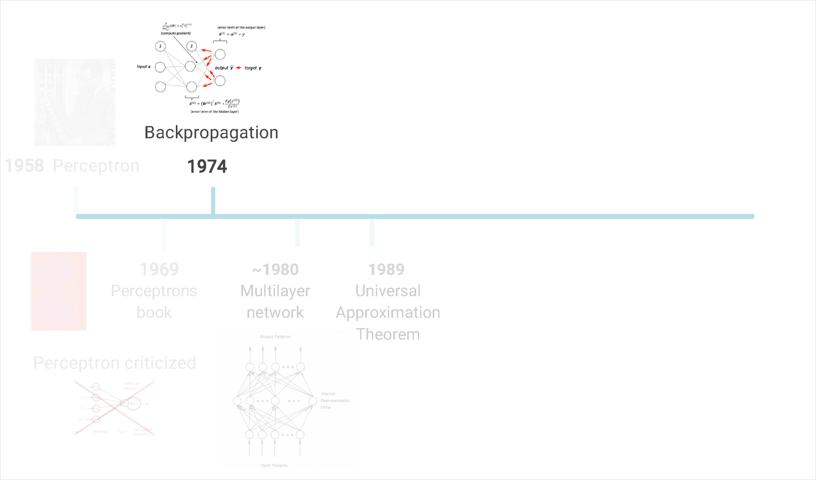


(Source: Lucas Masuch & Vincent Lepetit)

Backpropagation

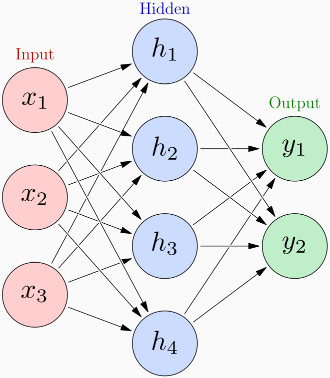


Learning with backpropagation



(Source: *Lucas Masuch & Vincent Lepetit*)

Artificial neural network / Multilayer perceptron / NeuralNet



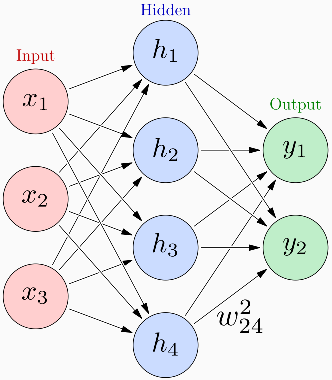
$$h_1 = g_1\left(w^1_{11}x_1 + w^1_{12}x_2 + w^1_{13}x_3 + b^1_1\right)$$
$$h_2 = g_1\left(w^1_{21}x_1 + w^1_{22}x_2 + w^1_{23}x_3 + b^1_2\right)$$
$$h_3 = g_1\left(w^1_{31}x_1 + w^1_{32}x_2 + w^1_{33}x_3 + b^1_3\right)$$
$$h_4 = g_1\left(w^1_{41}x_1 + w^1_{42}x_2 + w^1_{43}x_3 + b^1_4\right)$$

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

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



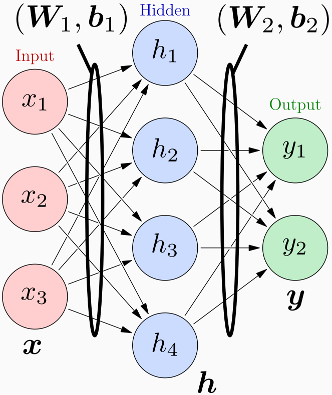
$$h_1 = g_1\left(w^1_{11}x_1 + w^1_{12}x_2 + w^1_{13}x_3 + b^1_1\right)$$
$$h_2 = g_1\left(w^1_{21}x_1 + w^1_{22}x_2 + w^1_{23}x_3 + b^1_2\right)$$
$$h_3 = g_1\left(w^1_{31}x_1 + w^1_{32}x_2 + w^1_{33}x_3 + b^1_3\right)$$
$$h_4 = g_1\left(w^1_{41}x_1 + w^1_{42}x_2 + w^1_{43}x_3 + b^1_4\right)$$

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

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



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

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

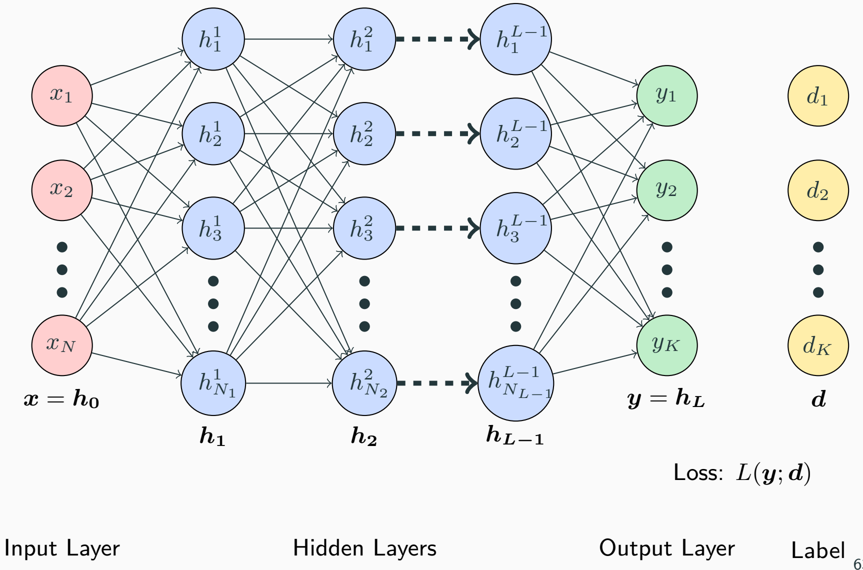
w_{ij}^k synaptic weight between previous node j and next node i at layer k.

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

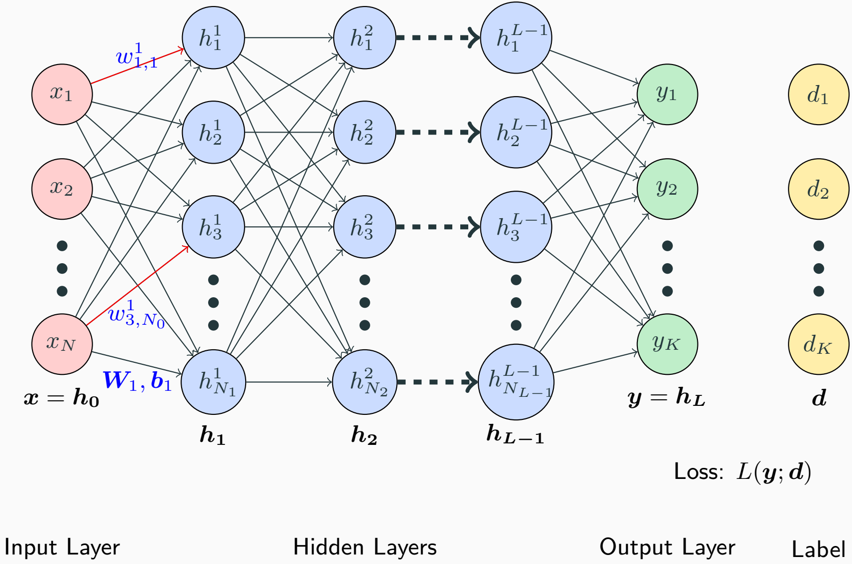
The matrices $\boldsymbol{W_k}$ and biases $\boldsymbol{b}_k$ are learned from labeled training data.

Feedforward Artificial Neural Network

Recall the feedforward structure

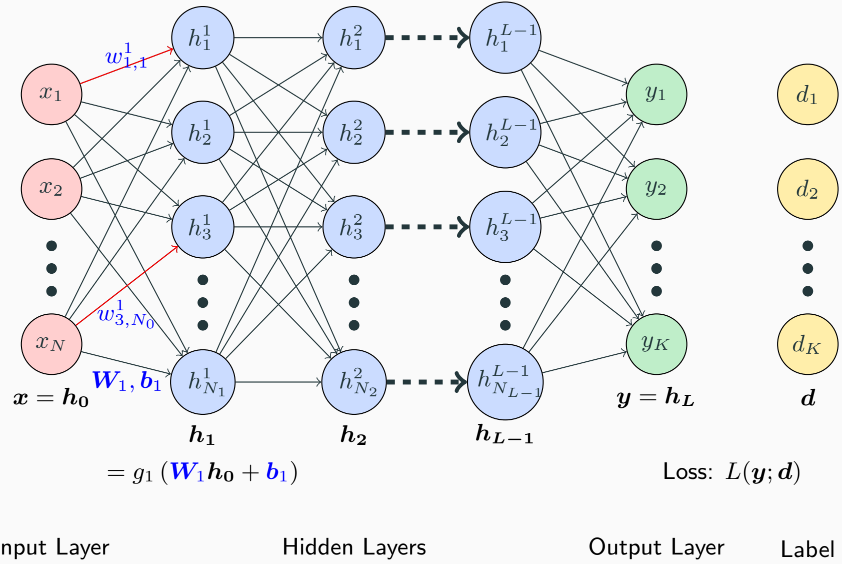


Recall the feedforward structure

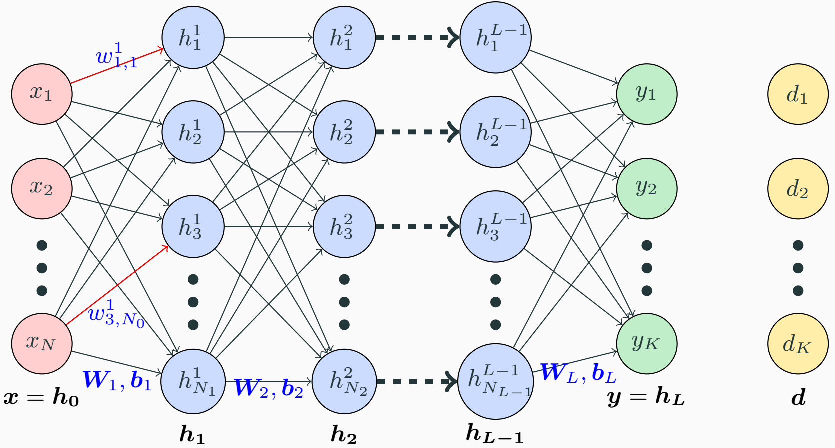


Feedforward Artificial Neural Network

Recall the feedforward structure



Recall the feedforward structure

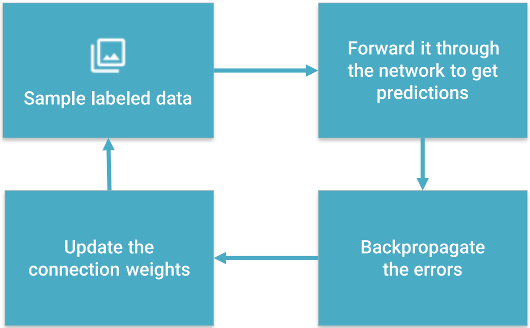


$= g_1\left(\boldsymbol{W}_1 \boldsymbol{h_0} + \boldsymbol{b}_1\right)$

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

Input Layer Hidden Layers Output Layer Label

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)

- The parameters of the neural network are

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

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

Objective: $\min_{\boldsymbol{W}} E(\boldsymbol{W})$ where $E(W) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{y}^i; \boldsymbol{d}^i)$

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

- **Solution:** no closed-form solutions ⇒ use (stochastic) gradient descent.
- $\frac{\partial E(\boldsymbol{W})}{\partial \boldsymbol{W}_k}$ not really rigorous, we will use the notation

$$\nabla_{\boldsymbol{W}_k} E(\boldsymbol{W}) \quad \text{and} \quad \nabla_{\boldsymbol{b}_k} E(\boldsymbol{W}).$$

Minimizing training loss

For multilayer neural networks $\boldsymbol{W} \mapsto E(\boldsymbol{W})$ is non-convex
$\Rightarrow$ No guarantee of convergence.
Even if convergence occurs, the solution depends on the initialization and the step size/learning rate γ.

Nevertheless, really good minima or saddle points are reached in practice by

$$\boldsymbol{W}^{t+1} \leftarrow \boldsymbol{W}^t - \gamma \nabla E(\boldsymbol{W}^t), \quad \gamma > 0$$

Gradient descent can be expressed coordinate by coordinate as:

$$w_{i,j}^{k,t+1} \leftarrow w_{i,j}^{k,t} - \gamma \frac{\partial E(\boldsymbol{W}^t)}{\partial w_{i,j}^k}$$

for all weights $w_{i,j}^k$ linking a node j to a node i in the next layer k.

$\Rightarrow$ The algorithm to compute $\dfrac{\partial E(\boldsymbol{W})}{\partial w_{i,j}^k}$ for ANNs is called **backpropagation**.

ANN – Optimization

- In practice we only use **stochastic** gradient descent with batch of training set.
- For some random small subset (e.g. batch) $\mathcal{S} \subset \mathcal{T}$, consider

$$E(\boldsymbol{W}; \mathcal{S}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{S}} L(\boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Our **goal** is to compute the noisy gradient

$$\nabla_{\boldsymbol{W}_k} E(\boldsymbol{W}; \mathcal{S}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{S}} \nabla_{\boldsymbol{W}_k} L(\boldsymbol{y}^i; \boldsymbol{d}^i).$$

ANN – Optimization

- In practice we only use **stochastic** gradient descent with batch of training set.
- For some random small subset (e.g. batch) $\mathcal{S} \subset \mathcal{T}$, consider

$$E(\boldsymbol{W};\mathcal{S}) = \sum_{(\boldsymbol{x}^i,\boldsymbol{d}^i)\in\mathcal{S}} L(\boldsymbol{y}^i;\boldsymbol{d}^i)$$

- Our **goal** is to compute the noisy gradient

$$\nabla_{\boldsymbol{W}_k} E(\boldsymbol{W};\mathcal{S}) = \sum_{(\boldsymbol{x}^i,\boldsymbol{d}^i)\in\mathcal{S}} \nabla_{\boldsymbol{W}_k} L(\boldsymbol{y}^i;\boldsymbol{d}^i).$$

- Why is this relevant to minimize $E(\boldsymbol{W}) = E(\boldsymbol{W};\mathcal{T})$?

ANN – Optimization

- **Stochastic** gradient descent: For some random small subset (e.g. batch) $\mathcal{S} \subset \mathcal{T}$, our **goal** is to compute the noisy gradient

$$\nabla_{\boldsymbol{W}_k} E(\boldsymbol{W}; \mathcal{S}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{S}} \nabla_{\boldsymbol{W}_k} L(\boldsymbol{y}^i; \boldsymbol{d}^i).$$

- **Unbiased approximation:** As soon as $\mathcal{S}$ spans uniformly the whole training set $\mathcal{T}$,

$$\begin{aligned}
\mathbb{E}_{\mathcal{S}} \left(\nabla_{\boldsymbol{W}_k} E(\boldsymbol{W}; \mathcal{S}) \right) &= \mathbb{E}_{\mathcal{S}} \left(\sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{S}} \nabla_{\boldsymbol{W}_k} L(\boldsymbol{y}^i; \boldsymbol{d}^i) \right) \\
&= \mathbb{E}_{\mathcal{S}} \left(\sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \mathbf{1}_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{S}} \nabla_{\boldsymbol{W}_k} L(\boldsymbol{y}^i; \boldsymbol{d}^i) \right) \\
&= \frac{|\mathcal{S}|}{|\mathcal{T}|} \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \nabla_{\boldsymbol{W}_k} L(\boldsymbol{y}^i; \boldsymbol{d}^i) = \frac{|\mathcal{S}|}{|\mathcal{T}|} \nabla_{\boldsymbol{W}_k} E(\boldsymbol{W}).
\end{aligned}$$

- Conclusion: In expectation the noisy gradient is equal to the gradient using the whole training dataset (unbiased estimator).

ANN – Optimization

Loss functions: Classical loss functions are:

For regression: $\boldsymbol{d}^i \in \mathbb{R}^K$

- Square error

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i,\boldsymbol{d}^i)\in\mathcal{T}} \frac{1}{2}\|\boldsymbol{y}^i - \boldsymbol{d}^i\|_2^2 = \sum_{(\boldsymbol{x}^i,\boldsymbol{d}^i)\in\mathcal{T}} \frac{1}{2}\sum_k (y_k^i - d_k^i)^2$$

For multi-class classification: $d^i \in \{1, \dots, K\}$, coded by $\boldsymbol{d}^i \in \{0,1\}^K$,

- Cross-entropy with softmax as the last layer

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

- Cross-entropy with softmax included in loss (PyTorch convention): $\boldsymbol{y}^i = \boldsymbol{a}^i$ is the output of the last linear layer:

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

ANN – Optimization

- The loss functions are of the form

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

- By linearity,

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

- There the neural net output $\boldsymbol{y}^i = f(\boldsymbol{x}^i; \boldsymbol{W})$ is a function of the input data $\boldsymbol{x}^i$ and the neural weights $\boldsymbol{W}$.
- We know the gradient of $L(\boldsymbol{y}^i; \boldsymbol{d}^i)$ with respect to the variable $\boldsymbol{y}$
  - Regression/Square error:

$$L(\boldsymbol{y}; \boldsymbol{d}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{d}\|_2^2 \quad \Rightarrow \quad \nabla_{\boldsymbol{y}} L(\boldsymbol{y}; \boldsymbol{d}) = \boldsymbol{y} - \boldsymbol{d}$$

  - Multi-class classification/cross-entropy:

$$L(\boldsymbol{y}; \boldsymbol{d}) = -y_d + \log\left(\sum_{k=1}^{K} \exp(y_k)\right) \quad \Rightarrow \quad \nabla_{\boldsymbol{y}} L(\boldsymbol{y}; \boldsymbol{d}) = \mathrm{softmax}(\boldsymbol{y}) - \boldsymbol{d}.$$

- The loss functions are of the form

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

- By linearity,

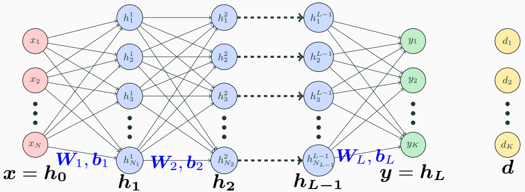
$$\nabla E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} \nabla L(\boldsymbol{y}^i; \boldsymbol{d}^i)$$

- There the neural net output $\boldsymbol{y}^i = f(\boldsymbol{x}^i; \boldsymbol{W})$ is a function of the input data $\boldsymbol{x}^i$ and the neural weights $\boldsymbol{W}$.
- We know the gradient of $L(\boldsymbol{y}^i; \boldsymbol{d}^i)$ with respect to the variable $\boldsymbol{y}$
- We still need to compute

$$\nabla_{\boldsymbol{W}_k} L(\boldsymbol{y}; \boldsymbol{d}) \quad \text{and} \quad \nabla_{\boldsymbol{b}_k} L(\boldsymbol{y}; \boldsymbol{d}) \quad \text{for } k = 0, \ldots, L.$$

- For simplicity above we will use the notation $E = L(\boldsymbol{y}; \boldsymbol{d})$, that is considering only one point.

ANN – Backpropagation



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:

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

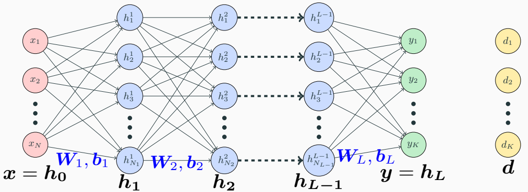
end

Output layer:

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

Compute loss:

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



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

Goal: Compute the gradient with respect to all parameters

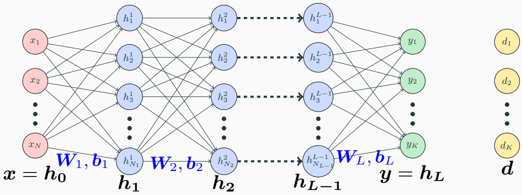
$$\frac{\partial E}{\partial w_{i,j}^k} = ? \qquad \frac{\partial E}{\partial b_i^k} = ?$$

for all

$k \in \{1, \ldots, L\}$,

$i \in \{1, \ldots, N_k\}$,

$j \in \{1, \ldots, N_{k-1}\}$.

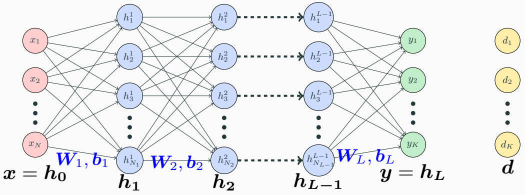


Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Going backward

- We know how to compute the loss function and its gradient:

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



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

Gradient with respect to last linear unit output $\boldsymbol{a}_L$

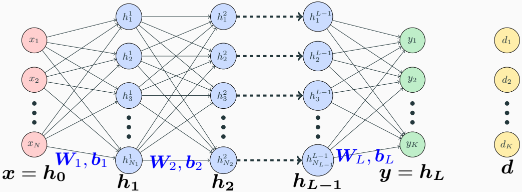
$$\boldsymbol{h}_L = g_L(\boldsymbol{a}_L)$$

That is for all $i \in \{1, \dots, N_L\}$, $h_i^L = g_L(a_i^L)$. By the chain rule,

$$\frac{\partial E}{\partial a_i^L} = \frac{\partial E}{\partial h_i^L} \frac{\partial h_i^L}{\partial a_i^L} = [\nabla_{\boldsymbol{h}_L} E]_i \, g_L'(a_i^L)$$

Vector formula: $\nabla_{\boldsymbol{a}_L} E = \nabla_{\boldsymbol{h}_L} E \odot g_L'(\boldsymbol{a}_L)$

where $\odot$ is the componentwise product between vectors, ie Hadamard product.



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

Gradient with respect to bias of last linear unit b_L

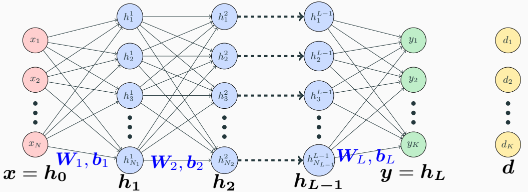
$$\boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$$

That is for all $i \in \{1, \ldots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$.

By the chain rule, for all $i \in \{1, \ldots, N_L\}$,

$$\frac{\partial E}{\partial b_i^L} = \frac{\partial E}{\partial a_i^L} \underbrace{\frac{\partial a_i^L}{\partial b_i^L}}_{=1} = \frac{\partial E}{\partial a_i^L} = [\nabla_{\boldsymbol{a}_L} E]_i$$

Vector formula: $\nabla_{\boldsymbol{b}_L} E = \nabla_{\boldsymbol{a}_L} E$



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

Gradient with respect to weights of last linear unit W_L

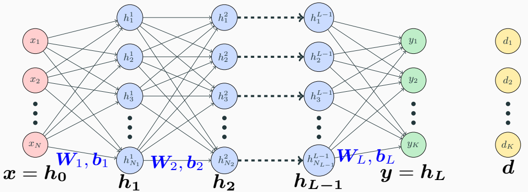
$$\boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$$

That is for all $i \in \{1, \ldots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$.

By the chain rule, for all $i \in \{1, \ldots, N_L\}$ and $j \in \{1, \ldots, N_{L-1}\}$

$$\frac{\partial E}{\partial w_{i,j}^L} = \frac{\partial E}{\partial a_i^L} \underbrace{\frac{\partial a_i^L}{\partial w_{i,j}^L}}_{=h_j^{L-1}} = \frac{\partial E}{\partial a_i^L} h_j^{L-1} = \left[\nabla_{\boldsymbol{a}_L} E\right]_i \left[\boldsymbol{h}_{L-1}\right]_j$$

Matrix formula: $\nabla_{\boldsymbol{W}_L} E = \nabla_{\boldsymbol{a}_L} E \, \boldsymbol{h}_{L-1}^T$

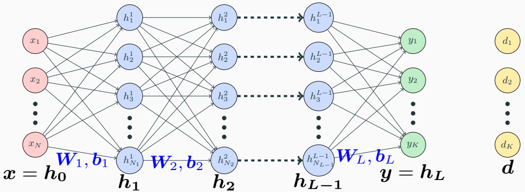


Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradients for last layer parameters

Given the gradient with respect to the output layer $\nabla_{\boldsymbol{h}_L} E$, so far we can compute:

- $\nabla_{\boldsymbol{a}_L} E = \nabla_{\boldsymbol{h}_L} E \odot g'_L(\boldsymbol{a}_L)$
- $\nabla_{\boldsymbol{b}_L} E = \nabla_{\boldsymbol{a}_L} E$
- $\nabla_{\boldsymbol{W}_L} E = \nabla_{\boldsymbol{a}_L} E \, \boldsymbol{h}_{L-1}^T$



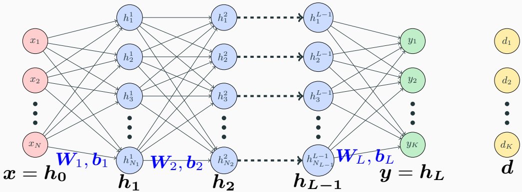
Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradients for last layer parameters

Given the gradient with respect to the output layer $\nabla_{\boldsymbol{h}_L} E$, so far we can compute:

- $\nabla_{\boldsymbol{a}_L} E = \nabla_{\boldsymbol{h}_L} E \odot g'_L(\boldsymbol{a}_L)$
- $\nabla_{\boldsymbol{b}_L} E = \nabla_{\boldsymbol{a}_L} E$
- $\nabla_{\boldsymbol{W}_L} E = \nabla_{\boldsymbol{a}_L} E \, \boldsymbol{h}_{L-1}^T$

How can we compute the gradients for the parameters of layer $L-1$?



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

Gradients for last layer parameters

Given the gradient with respect to the output layer $\nabla_{\boldsymbol{h}_L} E$, so far we can compute:

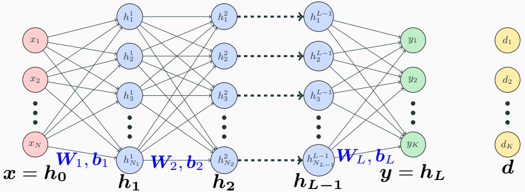
- $\nabla_{\boldsymbol{a}_L} E = \nabla_{\boldsymbol{h}_L} E \odot g'_L(\boldsymbol{a}_L)$
- $\nabla_{\boldsymbol{b}_L} E = \nabla_{\boldsymbol{a}_L} E$
- $\nabla_{\boldsymbol{W}_L} E = \nabla_{\boldsymbol{a}_L} E \, \boldsymbol{h}_{L-1}^T$

How can we compute the gradients for the parameters of layer $L-1$?

We need the expression of the gradient with respect to the last but one hidden layer $\boldsymbol{h}_{L-1}$... and then the same formulas apply!

$$\nabla_{\boldsymbol{h}_{L-1}} E = ?$$

ANN – Backpropagation



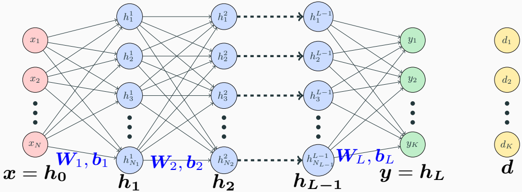
Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient with respect to the last but one hidden layer $\boldsymbol{h}_{L-1}$

Here, even to compute the scalar partial derivative $\dfrac{\partial E}{\partial h_j^{L-1}}$, we need to use differential calculus for multivariate functions since h_j^{L-1} appears in each component of $\boldsymbol{a}_L$:

For all $i \in \{1, \ldots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$.

ANN – Backpropagation



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

Gradient with respect to the last but one hidden layer $\boldsymbol{h}_{L-1}$

Let us recall the derivative rule for composition with affine maps:

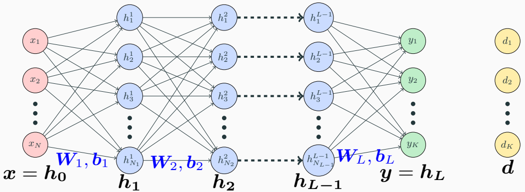
$$\text{For} \quad \varphi(\boldsymbol{x}) = f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \quad \text{one has} \quad \nabla\varphi(\boldsymbol{x}) = \boldsymbol{A}^T \nabla f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}).$$

Using the decomposition

$$\begin{array}{ccccc} \mathbb{R}^{N_{L-1}} & \to & \mathbb{R}^{N_L} & \to & \mathbb{R} \\ \boldsymbol{h}_{L-1} & \mapsto & \boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L & \mapsto & E \end{array}$$

Vector formula: $\nabla_{\boldsymbol{h}_{L-1}} E = {\boldsymbol{W}_L}^T \nabla_{\boldsymbol{a}_L} E$

ANN – Backpropagation



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:

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

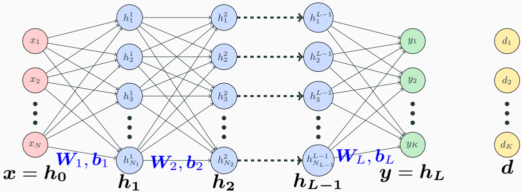
Gradient of weights:

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

Gradient of previous hidden layer:

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

end



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:

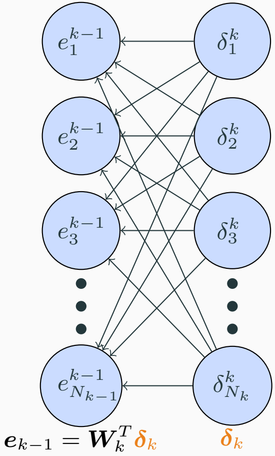
$\nabla_{\boldsymbol{W}_k} E = \boldsymbol{\delta}_k \boldsymbol{h}_{k-1}^T$

Gradient of previous hidden layer:

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

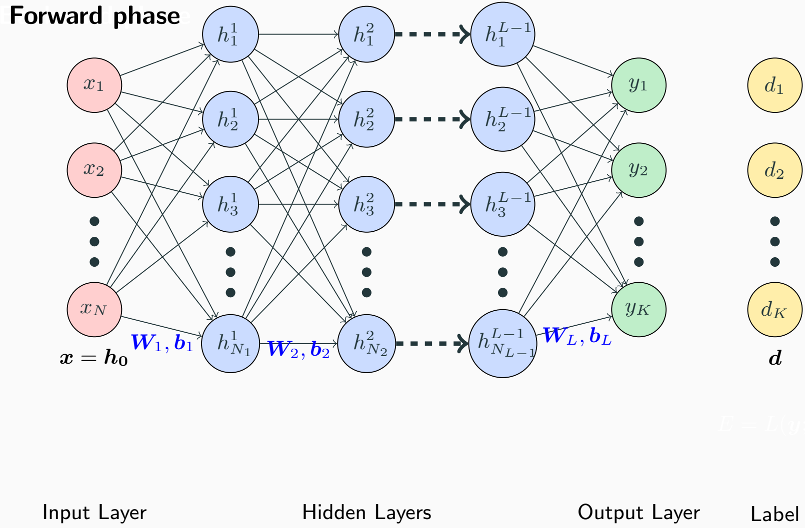
end

Error backpropagation



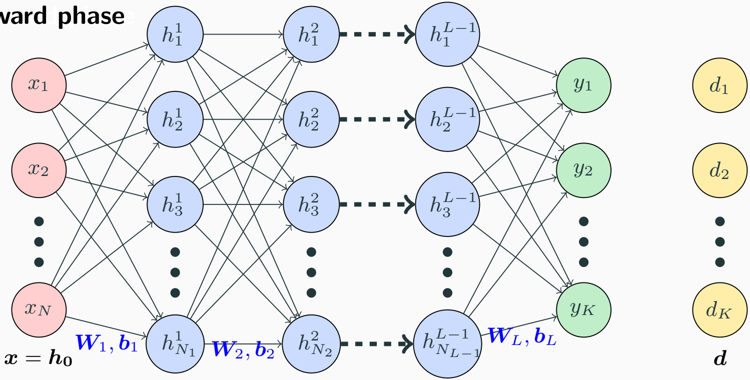
- Gradient of previous hidden layer: $\boldsymbol{e}_{k-1} = \nabla_{\boldsymbol{h}_{k-1}} E = \boldsymbol{W}_k^T \boldsymbol{\delta}_k$
- Multiplying by $\boldsymbol{W}_k^T$ corresponds to passing to the linear layer in reverse order.
- The error is backpropagated layer by layer to compute the gradient with respect to each layer parameters.

Error backpropagation



Error backpropagation

Forward phase

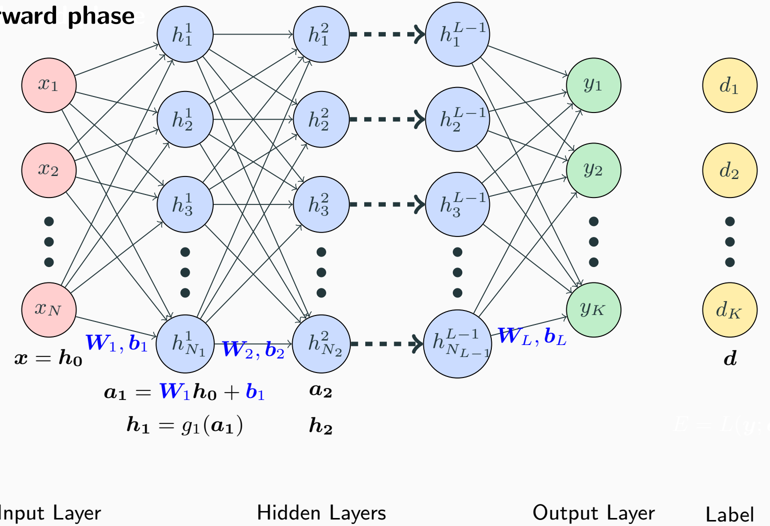


$$\boldsymbol{a_1} = \boldsymbol{W}_1 \boldsymbol{h_0} + \boldsymbol{b}_1$$

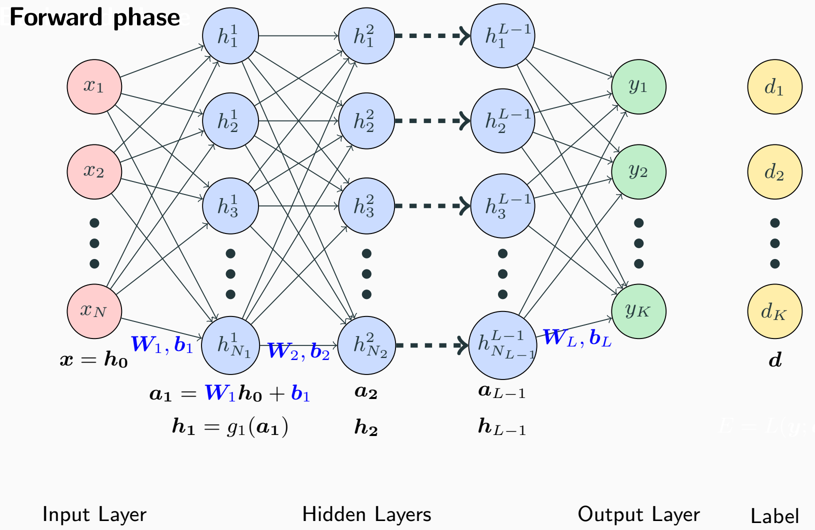
$$\boldsymbol{h_1} = g_1(\boldsymbol{a_1})$$

Input Layer | Hidden Layers | Output Layer | Label

Error backpropagation

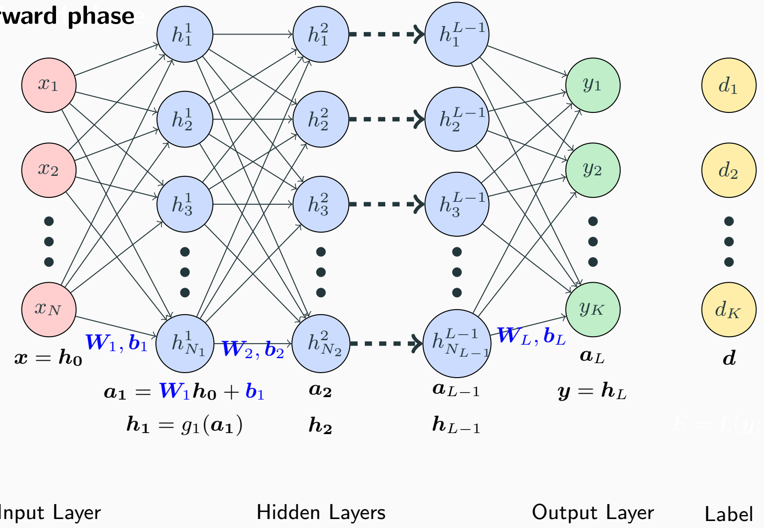


Error backpropagation

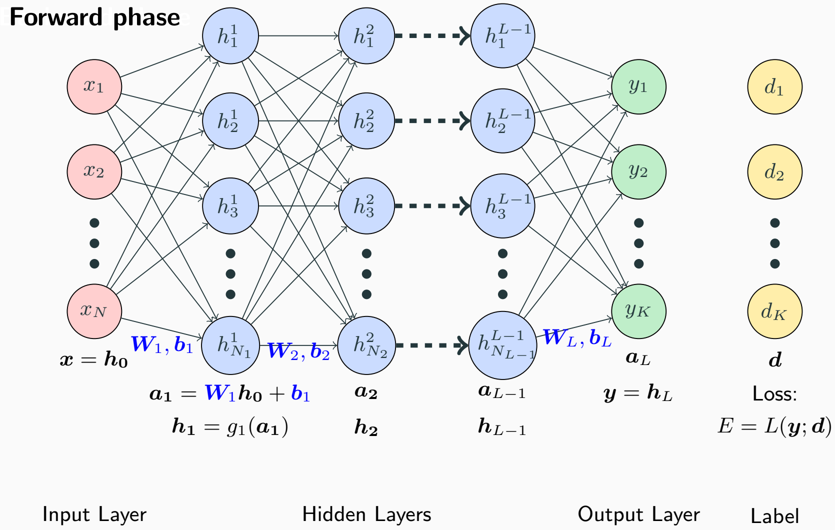


Error backpropagation

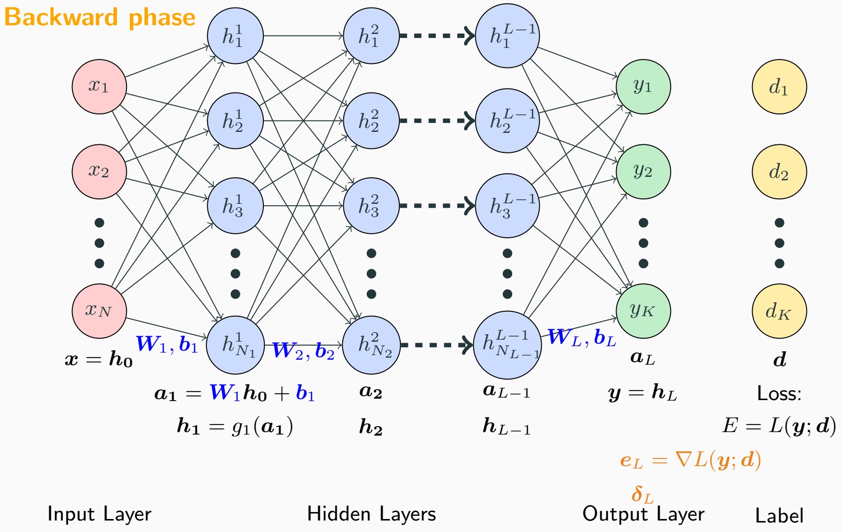
Forward phase



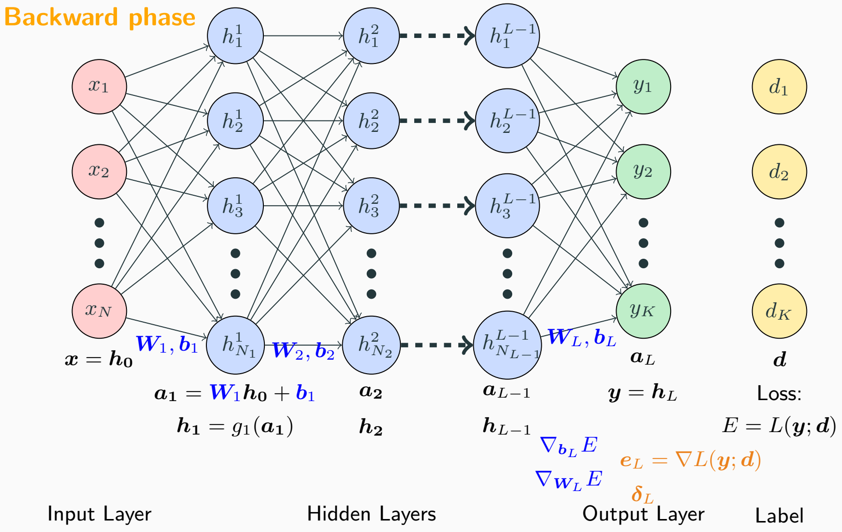
Error backpropagation



Error backpropagation

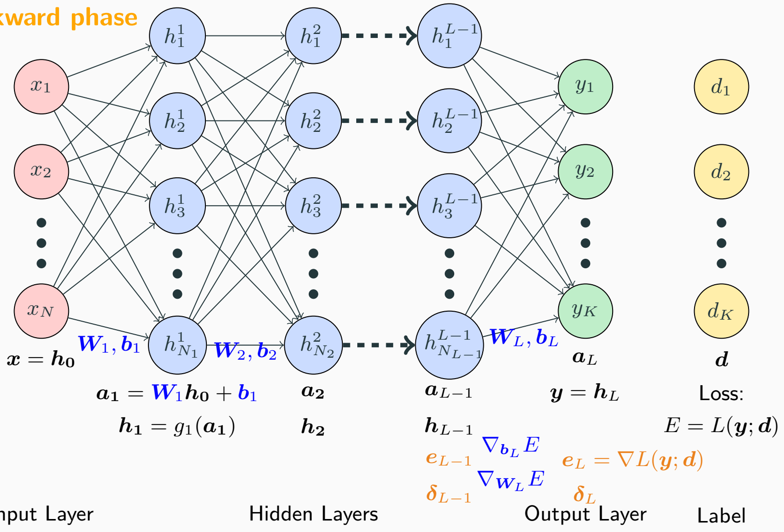


Error backpropagation



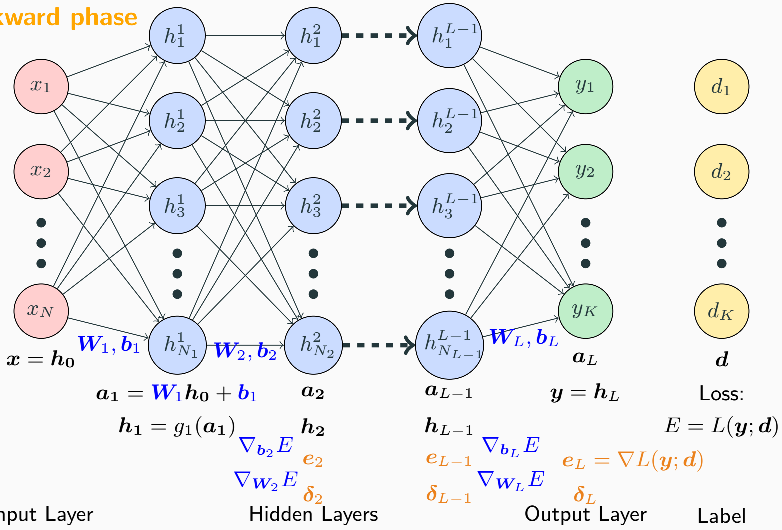
Error backpropagation

Backward phase

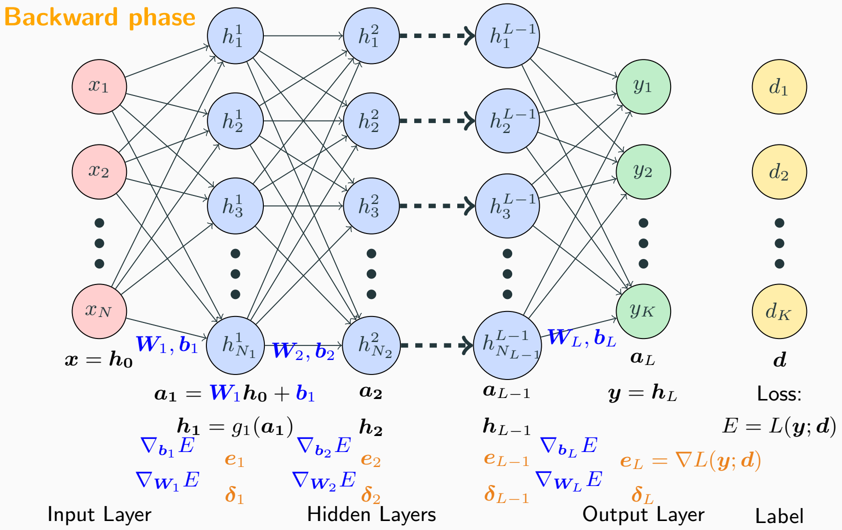


Error backpropagation

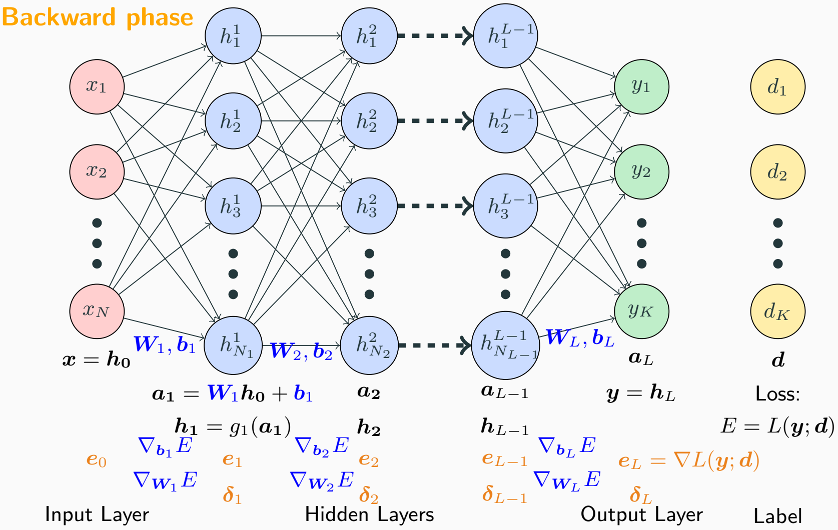
Backward phase



Error backpropagation



Error backpropagation



Error backpropagation in practice

Training loss:

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{y}^i; \boldsymbol{d}^i)$$

- The backpropagation procedure computes $\nabla_{\boldsymbol{W}} L(\boldsymbol{y}^i; \boldsymbol{d}^i) = \nabla_{\boldsymbol{W}} L(f(\boldsymbol{x}^i; \boldsymbol{W}); \boldsymbol{d}^i)$.
- This has to be done for each data point $\boldsymbol{x}^i \in \mathcal{T}$.
- By linearity, the final gradient $\nabla E(\boldsymbol{W})$ is the sum of all individual gradients $\nabla_{\boldsymbol{W}} L(\boldsymbol{y}^i; \boldsymbol{d}^i)$.
- These gradients are summed sequentially (no need to store each individual gradients).
- In general we do not compute the exact gradient...

Error backpropagation in practice

Batch loss:

$$E(\boldsymbol{W}) \approx \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{S}} L(\boldsymbol{y}^i; \boldsymbol{d}^i), \quad \text{with} \quad \mathcal{S} \subset \mathcal{T}$$

- The backpropagation has to be done for each visited data point $\boldsymbol{x}^i \in \mathcal{S}$ of the batch.
- The gradient for each point $\boldsymbol{x}^i$ is added to the running gradient = current gradient estimation.
- Once the noisy estimated gradient is used as a gradient step, one needs to set the gradients to zero: See PyTorch `torch.zero_grad()` procedure.

Questions?

Sources, images courtesy and acknowledgment

Charles Deledalle

V. Lepetit

L. Masuch