

Deep learning – What is deep learning?

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)

Machine learning – Brief history

Brief history

- **First wave**

- 1943. McCulloch and Pitts proposed the first neural model,
- 1958. Rosenblath introduced the Perceptron,
- 1969. Minsky and Papert's book demonstrated the limitation of single layer perceptrons, and almost the whole field went into hibernation.

- **Second wave**

- 1986. Backpropagation learning algorithm was rediscovered,
- 1989. Yann LeCun (re)introduced Convolutional Neural Networks,
- 1998. Breakthrough of CNNs in recognizing hand-written digits.

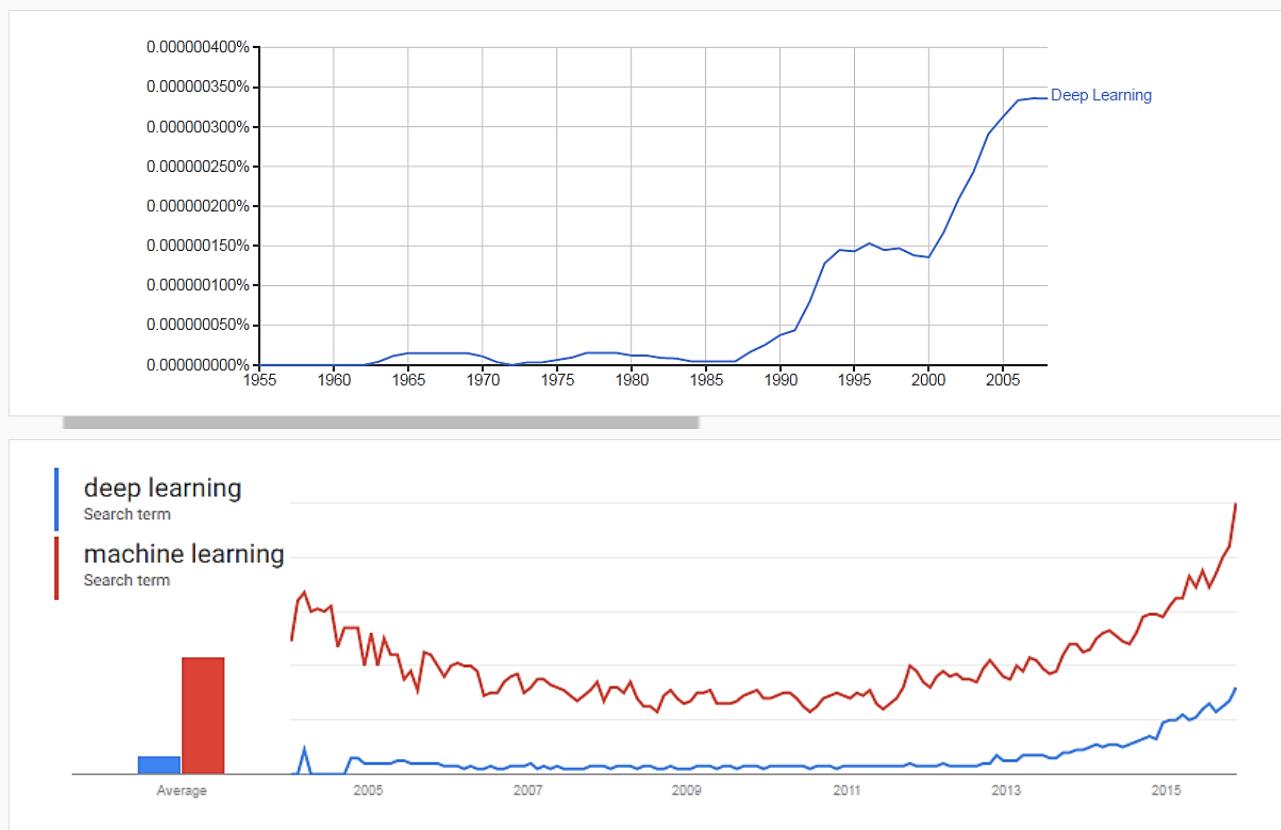
- **Third wave**

- 2006. Deep (neural network) learning gains popularity,
- 2012. Made significant breakthroughs in many applications,
- 2015. AlphaGo first program to beat a professional Go player.

(Source: Jun Wang)

Deep learning – Interest

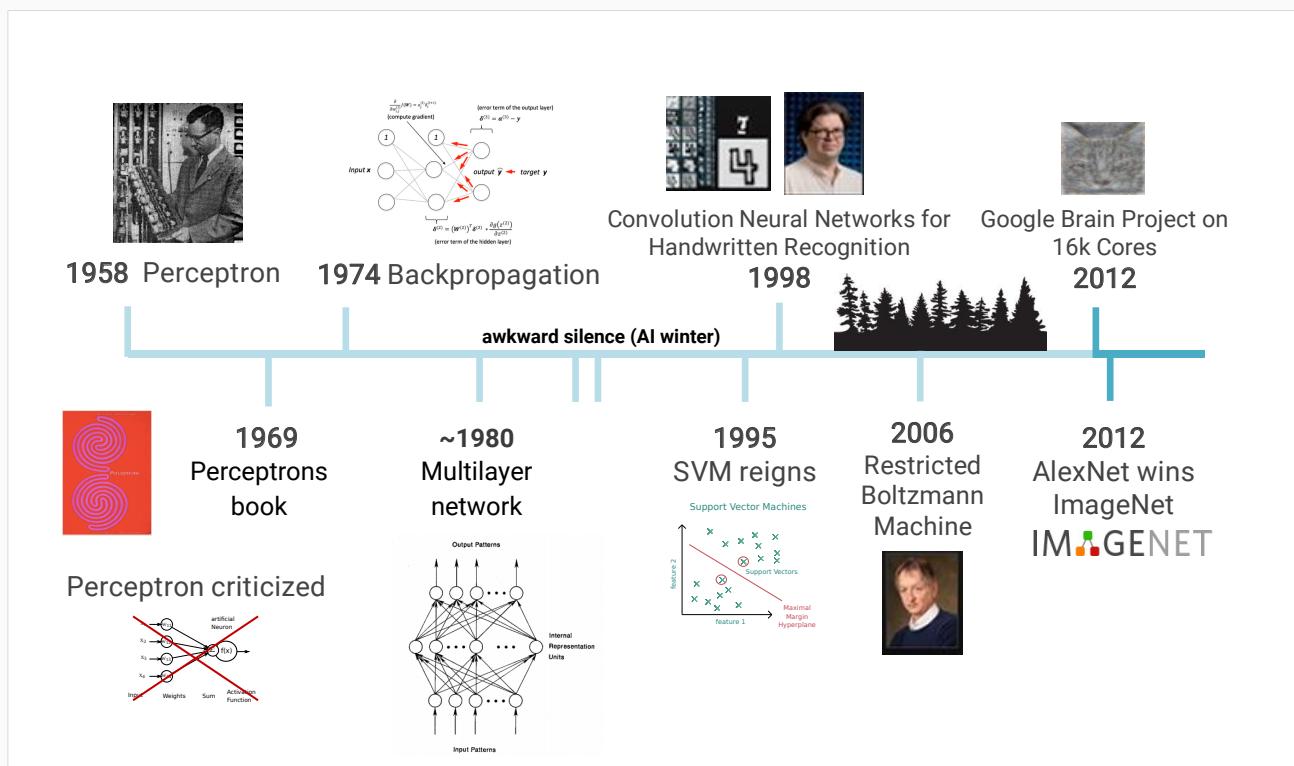
Google NGRAM & Google Trends



(Source: Lucas Masuch)

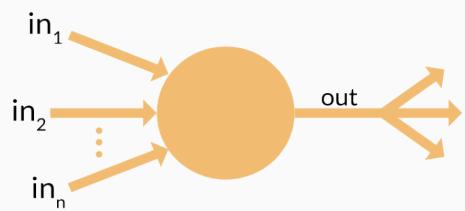
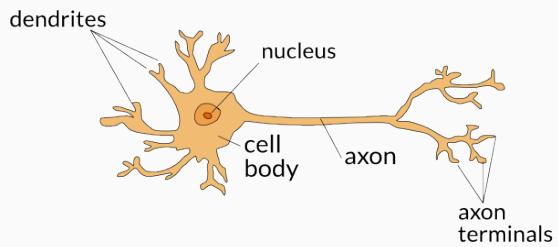
Machine learning – Timeline

Timeline of (deep) learning



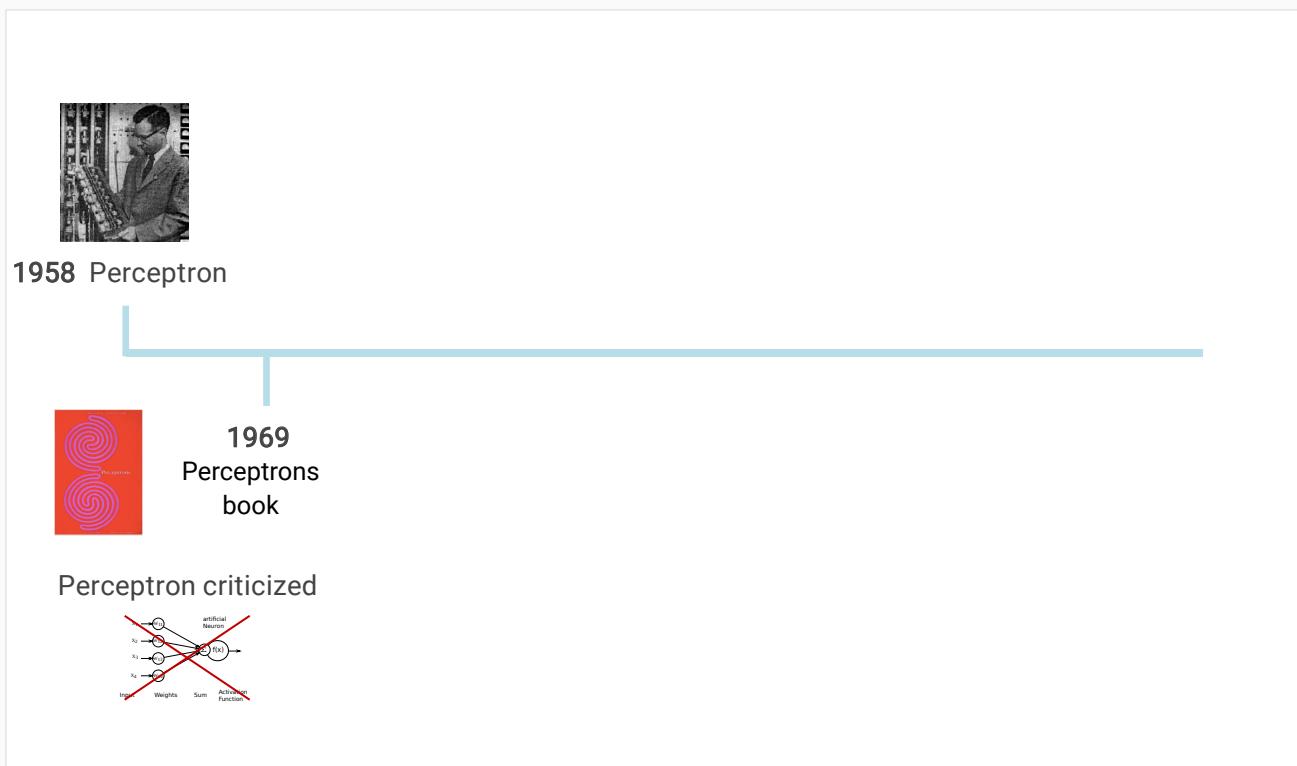
(Source: Lucas Masuch & Vincent Lepetit)

Perceptron



Machine learning – ANN - Backpropagation

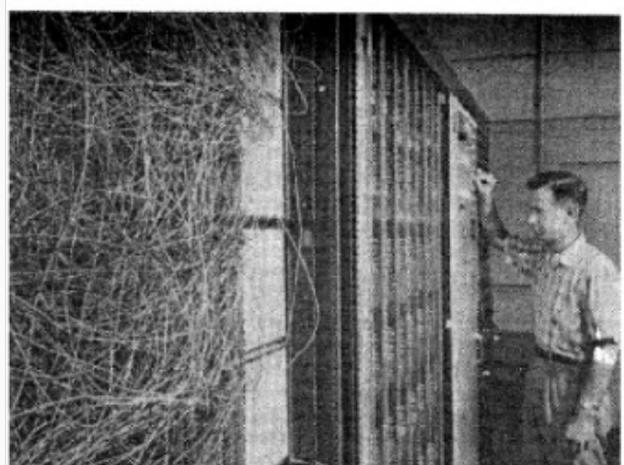
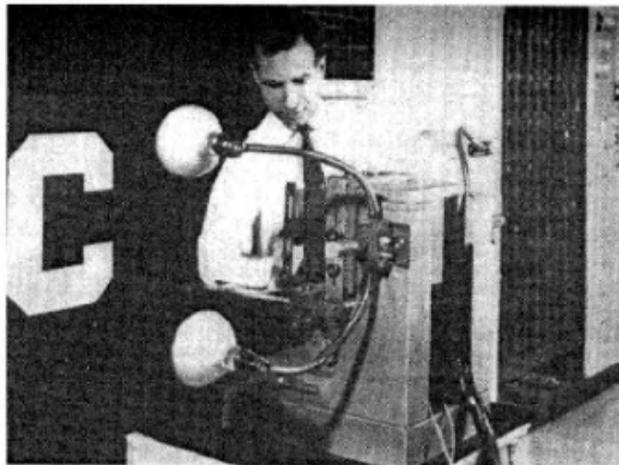
Perceptron



(Source: Lucas Masuch & Vincent Lepetit)

Machine learning – Perceptron

Perceptron (Frank Rosenblatt, 1958)



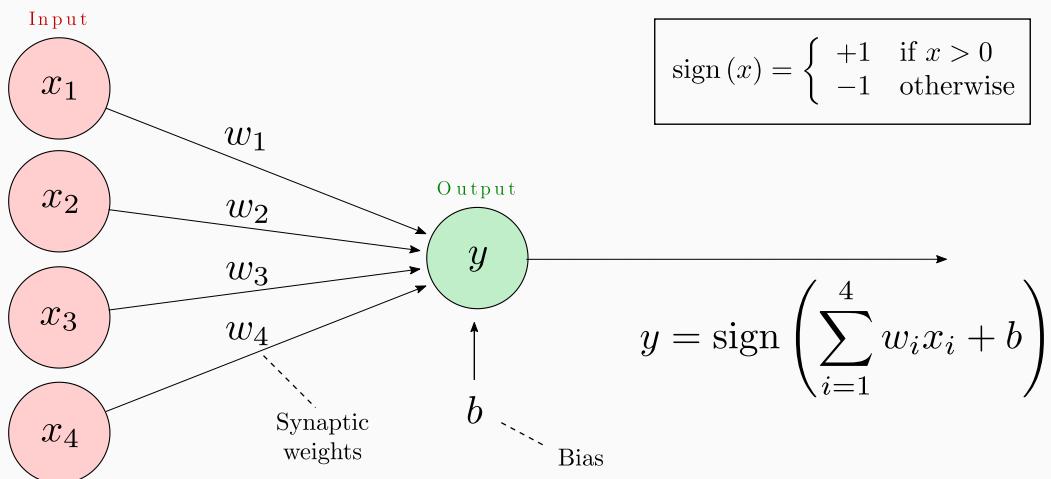
First binary classifier based on supervised learning (discrimination).

Foundation of modern artificial neural networks.

At that time: technological, scientific and philosophical challenges.

Machine learning – Perceptron – Representation

Representation of the Perceptron



Parameters of the perceptron

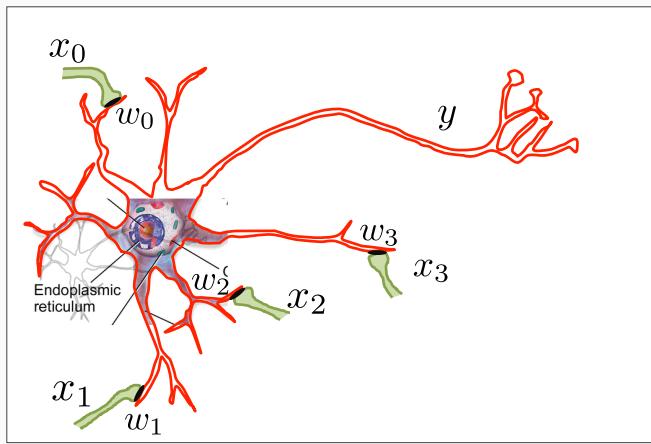
- w_k : synaptic weights
 - b : bias
- } ← real parameters to be estimated.

Training = adjusting the weights and biases

Machine learning – Perceptron – Inspiration

The origin of the Perceptron

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



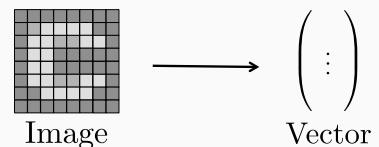
- When a neuron receives a stimulus with high enough voltage, it emits an **action potential** (aka, nerve impulse or spike). It is said to **fire**.
- The perceptron mimics this activation effect: it fires only when

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

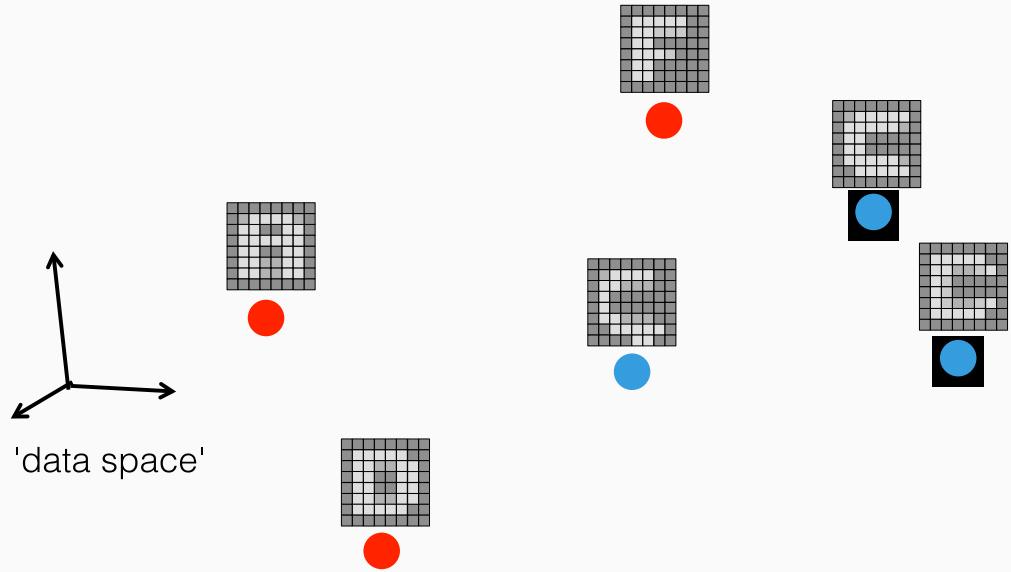
$$y = \underbrace{\text{sign}(w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + b)}_{f(\mathbf{x}; \mathbf{w})} = \begin{cases} +1 & \text{for the first class} \\ -1 & \text{for the second class} \end{cases}$$

Machine learning – Perceptron – Principle

- ① Data are represented as vectors:



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



(Source: Vincent Lepetit)

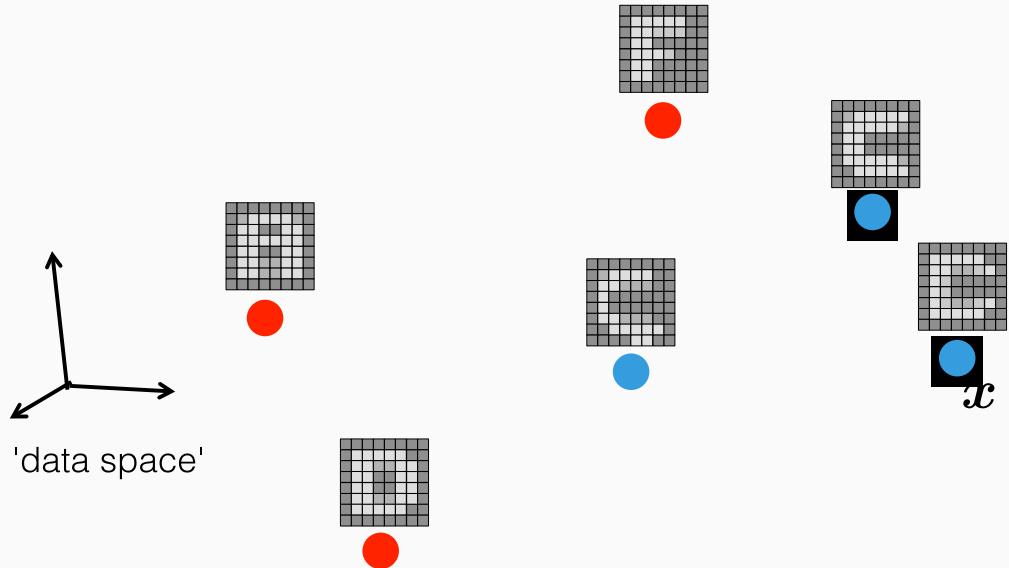
Machine learning – Perceptron – Principle

③ **Training:** find w and b so that:

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

Dot product:

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



(Source: Vincent Lepetit)

Machine learning – Perceptron – Principle

③ **Training:** find w and b so that:

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

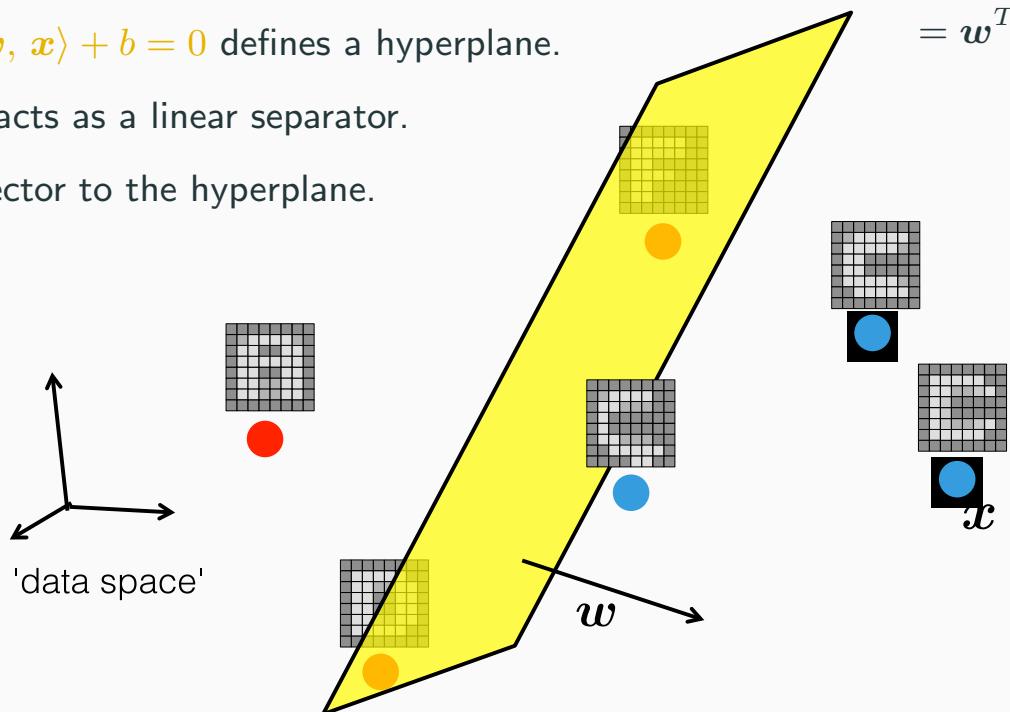
The equation $\langle w, x \rangle + b = 0$ defines a hyperplane.

The hyperplane acts as a linear separator.

w is a normal vector to the hyperplane.

Dot product:

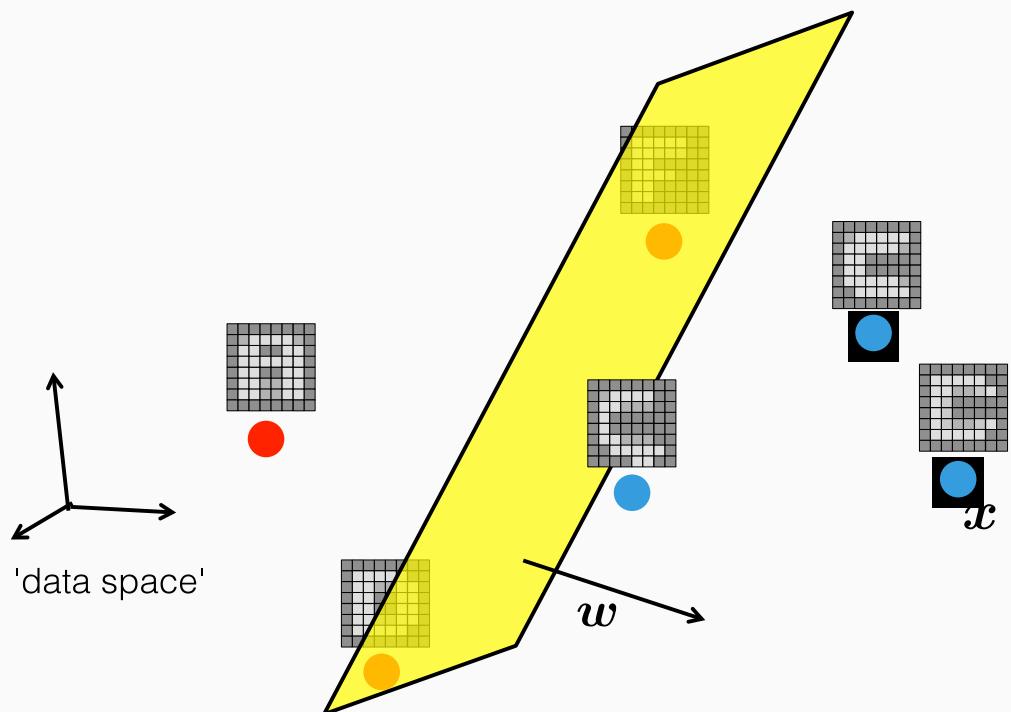
$$\begin{aligned}\langle w, x \rangle &= \sum_{i=1}^d w_i x_i \\ &= w^T x\end{aligned}$$



(Source: Vincent Lepetit)

Machine learning – Perceptron – Principle

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

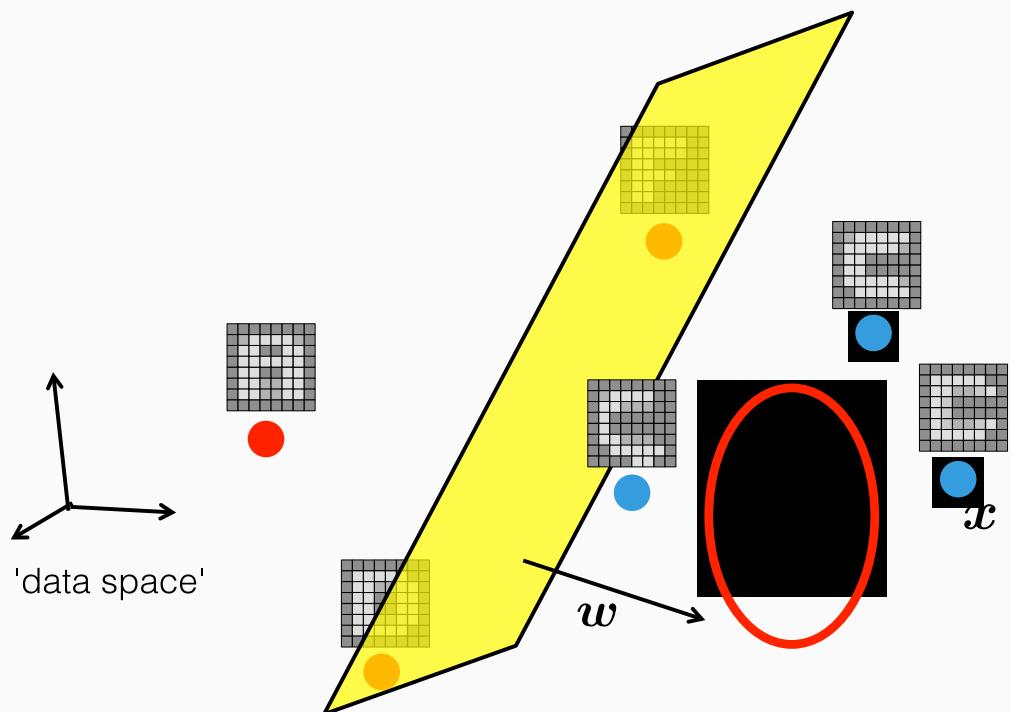


(Source: Vincent Lepetit)

Machine learning – Perceptron – Principle

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

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

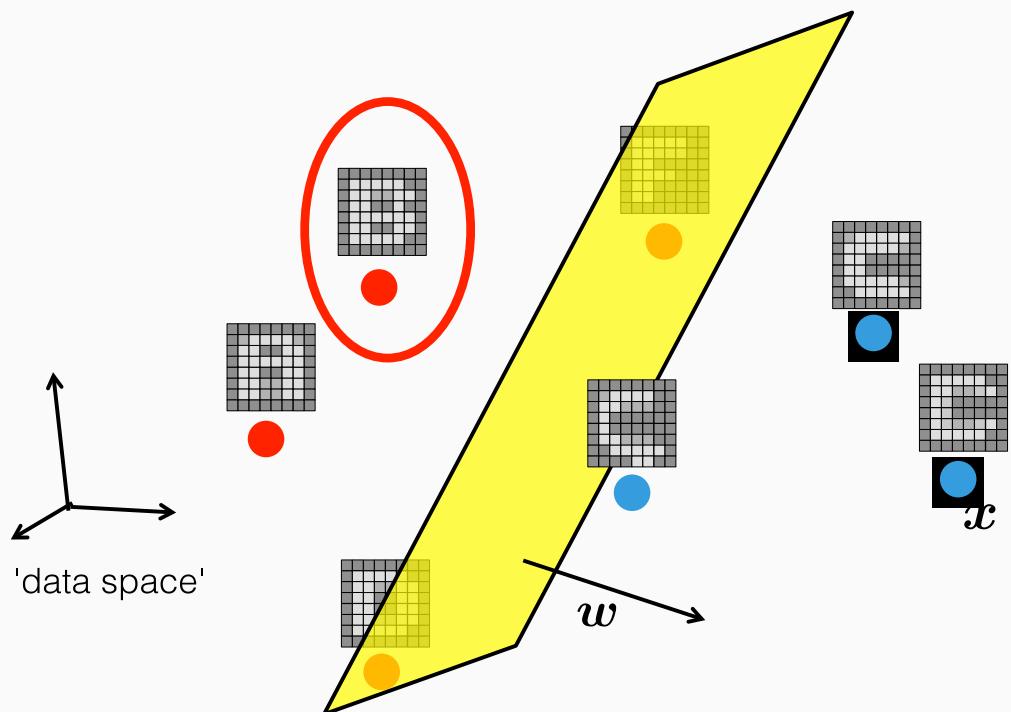


(Source: Vincent Lepetit)

Machine learning – Perceptron – Principle

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

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



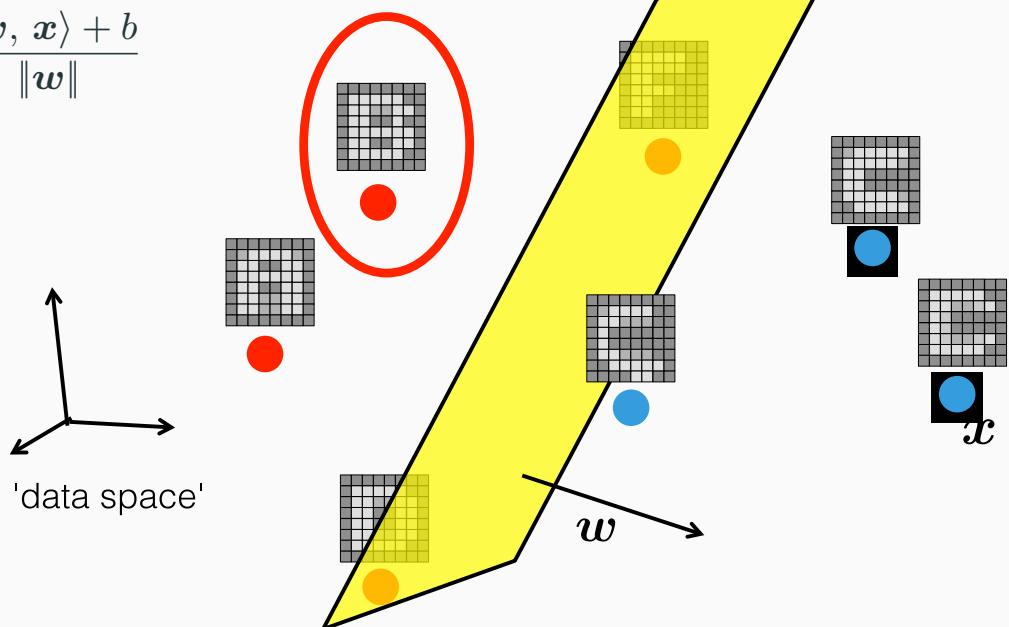
Machine learning – Perceptron – Principle

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

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

(signed) distance of x to the hyperplane:

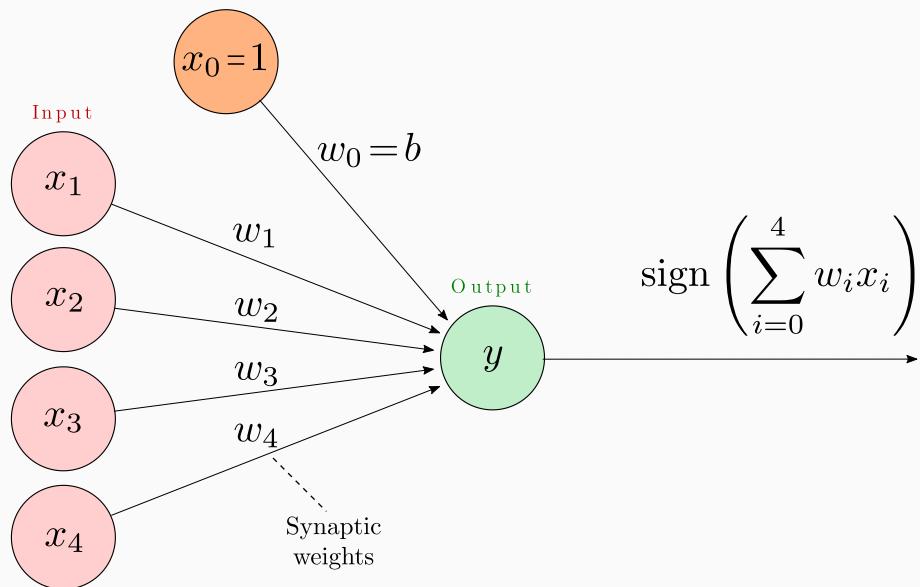
$$r = \frac{\langle w, x \rangle + b}{\|w\|}$$



(Source: Vincent Lepetit)

Machine learning – Perceptron – Representation

Alternative representation



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

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

Machine learning – Perceptron – Training

Perceptron algorithm

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

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

\mathbf{x}^i – i -th training example
 d^i – desired output for sample i
 N – number of training samples
 $\{-1, +1\}$

How: minimize classification errors

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

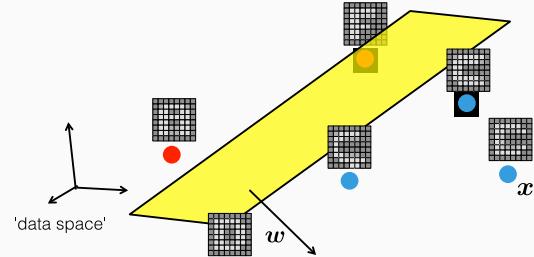
- penalize only misclassified samples ($y \neq d$),
- zero if all samples are correctly classified,
- proportional to the absolute distance to the hyperplane ($d \times \langle \mathbf{w}, \mathbf{x} \rangle < 0$)

Machine learning – Perceptron – Training

Perceptron algorithm

Algorithm:

- Initialize \mathbf{w} randomly
- Repeat until convergence
 - For all $(\mathbf{x}, d) \in \mathcal{T}$
 - Compute: $y = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$
 - If $y \neq d$:
Update: $\mathbf{w} \leftarrow \mathbf{w} + d\mathbf{x}$



- Converges to some solution if the training data are **linearly separable**,
- Corresponds to stochastic gradient descent for $E(\mathbf{w})$ (see later),
- But may pick any of many solutions of varying quality.
 \Rightarrow Poor generalization error.

Machine learning – Perceptron – Training

Variant: ADALINE (Adaptive Linear Neuron) algorithm

(Widrow & Hoff, 1960).

Loss: minimize instead the least square error with the prediction without sign

$$\min_{\mathbf{w}} E(\mathbf{w}) = \sum_{(\mathbf{x}, d) \in \mathcal{T}} (\langle \mathbf{w}, \mathbf{x} \rangle - d)^2$$

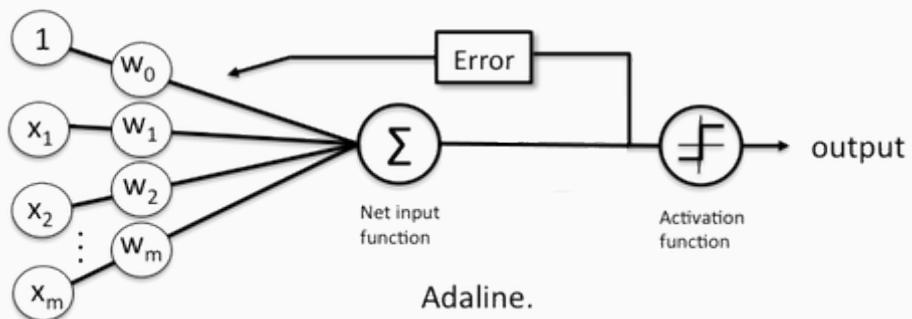
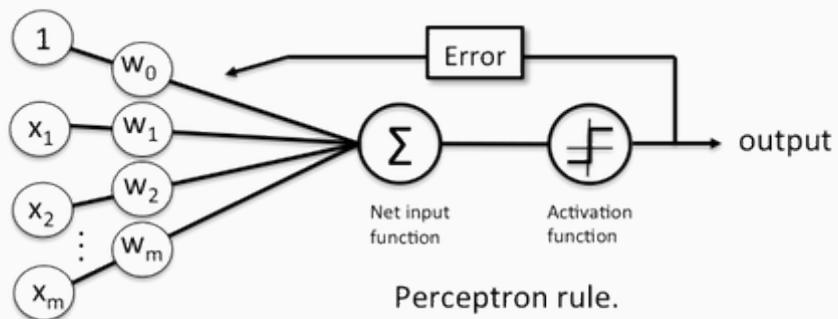
Algorithm:

- Initialize \mathbf{w} randomly
- Repeat until convergence
 - For all $(\mathbf{x}, d) \in \mathcal{T}$
 - Compute: $y = \langle \mathbf{w}, \mathbf{x} \rangle$
 - Update: $\mathbf{w} \leftarrow \mathbf{w} + \gamma(d - y)\mathbf{x}$, $\gamma > 0$

Also corresponds to stochastic gradient descent for $E(\mathbf{w})$ (see later).

Machine learning – Perceptron – Training

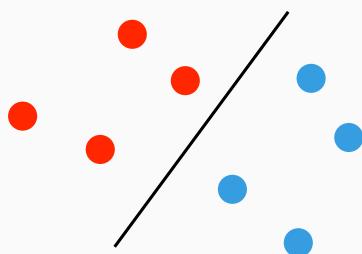
Perceptron vs ADALINE algorithm



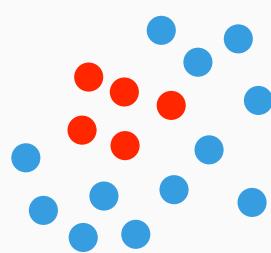
Machine learning – Perceptron – Perceptrons book

Perceptrons book (Minsky and Papert, 1969)

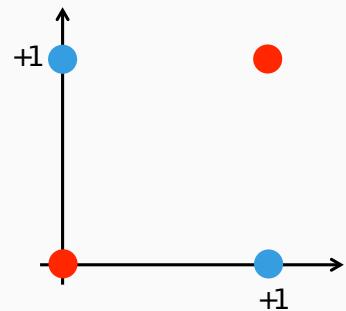
A perceptron can only classify data points that are linearly separable:



Linearly separable



Nonlinearly separable



The xor function

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

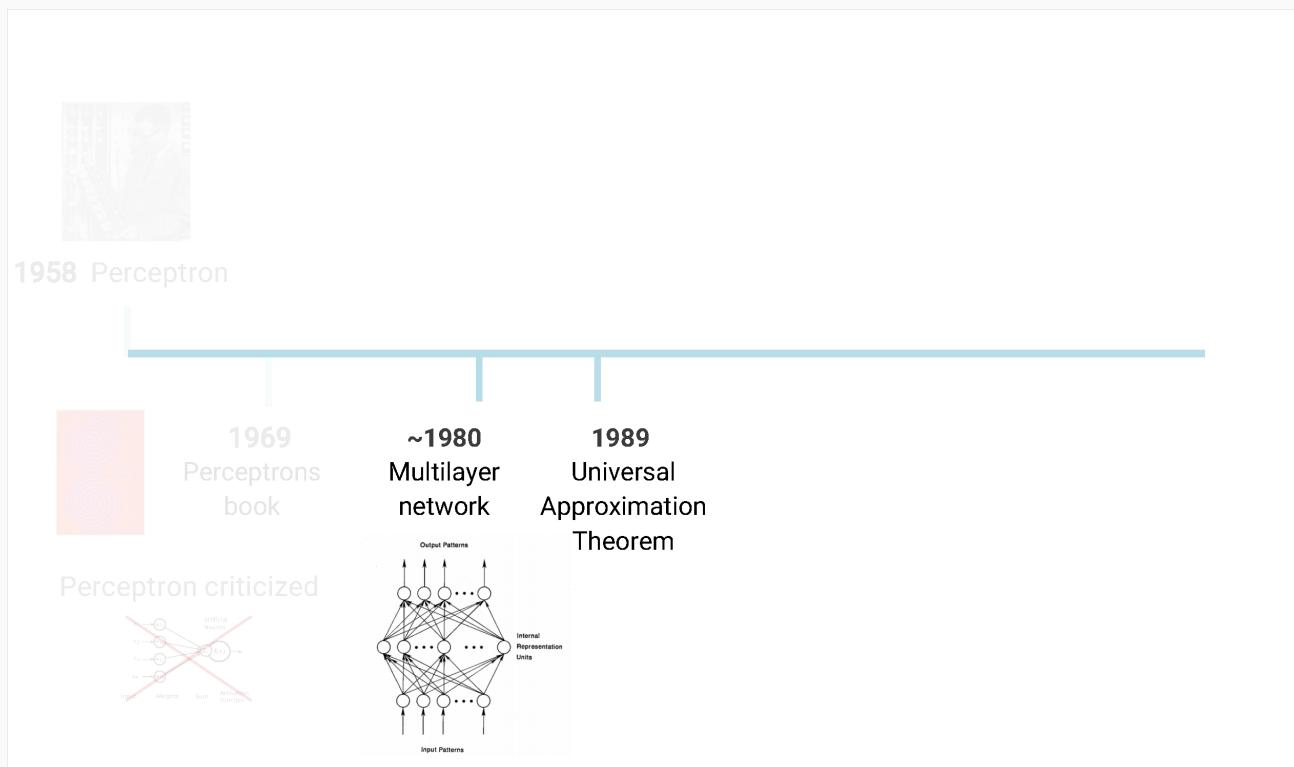
(Source: Vincent Lepetit)

Artificial neural network



Machine learning – Artificial neural network

Artificial neural network



(Source: Lucas Masuch & Vincent Lepetit)

Machine learning – Artificial neural network

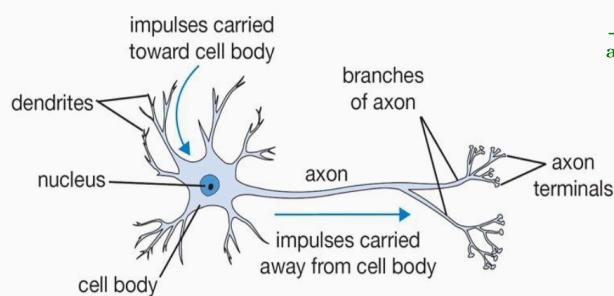
Artificial neural network



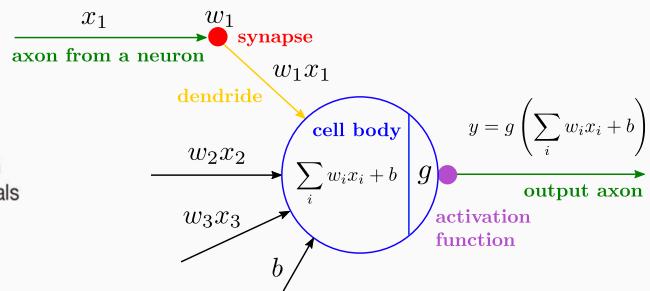
- Supervised learning method initially inspired by the behavior of the human brain.
- Consists of the inter-connection of several small units (just like in the human brain).
- Essentially numerical but can handle classification and discrete inputs with appropriate coding.
- Introduced in the late 50s,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

Machine learning – Artificial neural network

Artificial neuron (McCulloch & Pitts, 1943)



Biological neuron

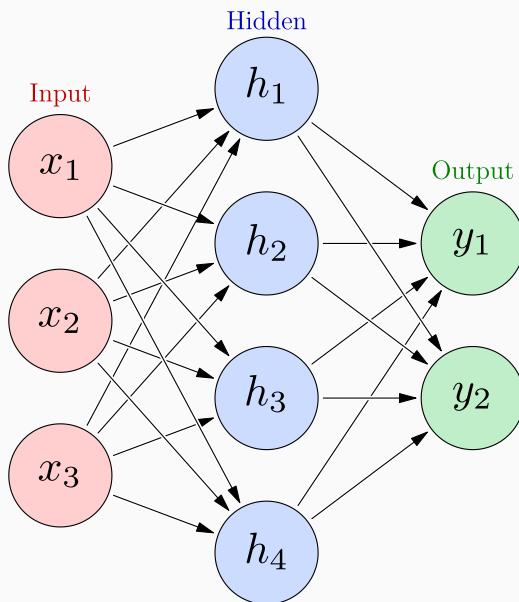


Artificial neuron

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

Machine learning – ANN

Artificial neural network / Multilayer perceptron / NeuralNet

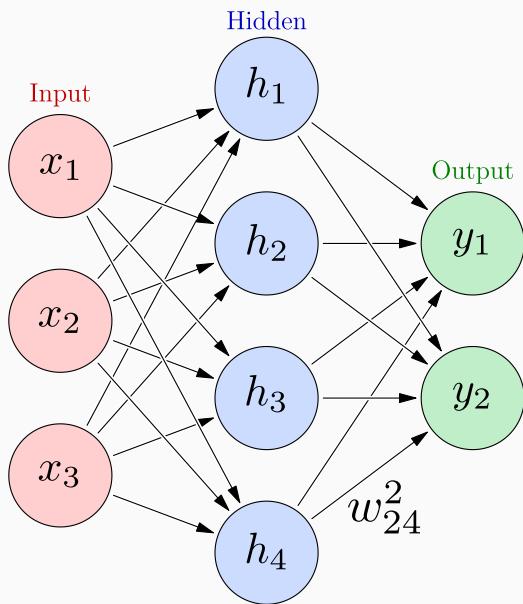


Quiz: how many layers does this network have?

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

Machine learning – ANN

Artificial neural network / Multilayer perceptron / NeuralNet



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

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

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

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

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

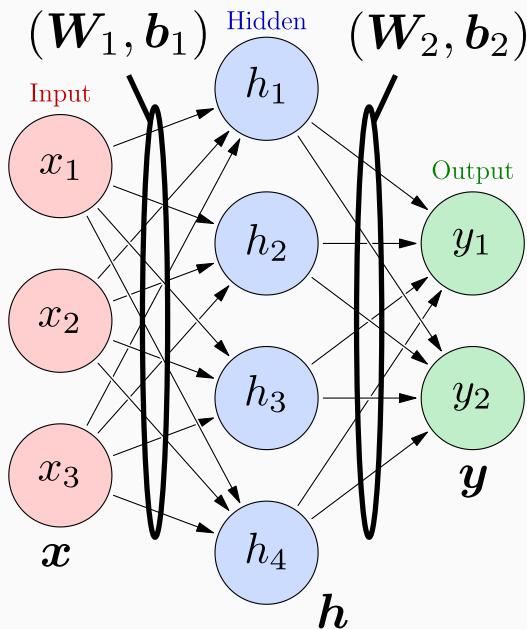
$$y_2 = g_2 (w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2)$$

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

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

Machine learning – ANN

Artificial neural network / Multilayer perceptron / NeuralNet



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

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

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

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

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

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

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

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

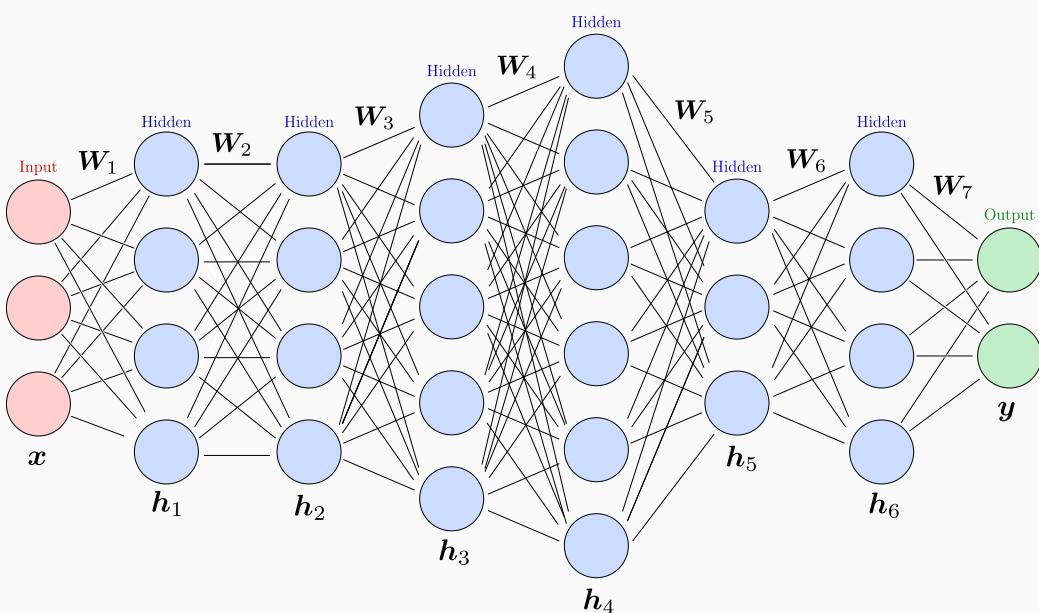
w_{ij}^k synaptic weight between previous node j and next node i at layer k .

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

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

Machine learning – ANN

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.

Machine learning – ANN

Artificial neural network / Multilayer perceptron

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

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

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

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

- It can be defined recursively as

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

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

Machine learning – ANN – Activation functions

Activation functions

Linear units: $g(a) = a$

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

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

Sometimes used for linear dimensionality reduction (similarly to PCA).

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.

Machine learning – ANN – Activation functions

Activation functions

Threshold units: for instance the sign function

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

or Heaviside (aka, step) activation functions

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

Unlike the perceptron, discontinuities in the hidden layers
make the learning really difficult.

Machine learning – ANN – Activation functions

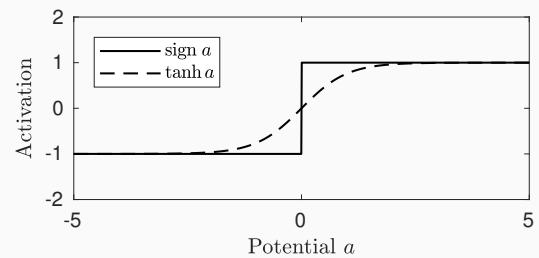
Activation functions

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 :

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

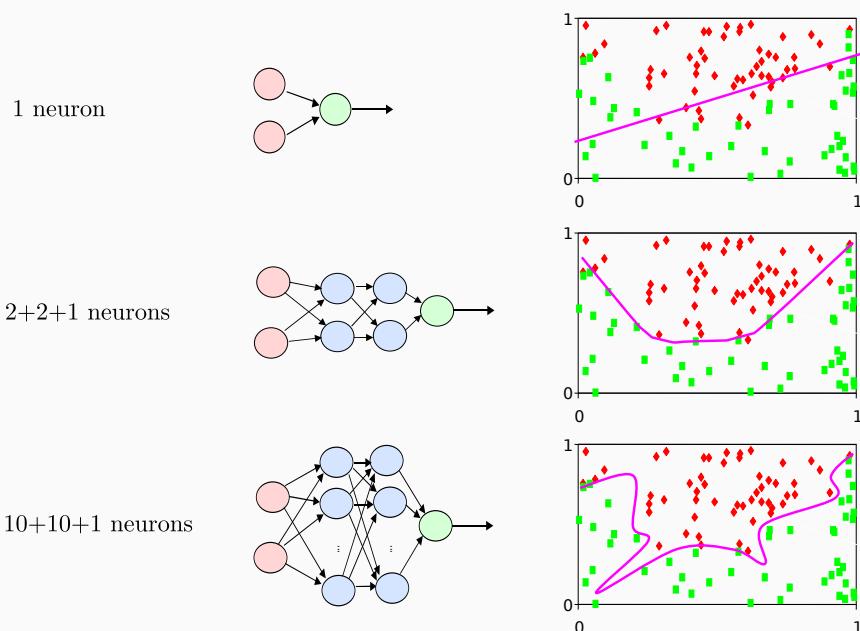
- Differentiable approximations of the step and sign functions, respectively.
- Act as threshold units for large values of $|a|$ and as linear for small values.

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:

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

The architecture of the network defines the shape of the separator



Separation

$$\{\mathbf{x} \setminus P(C_1|\mathbf{x}) = P(C_2|\mathbf{x})\}$$

Complexity/capacity of
the network

⇒

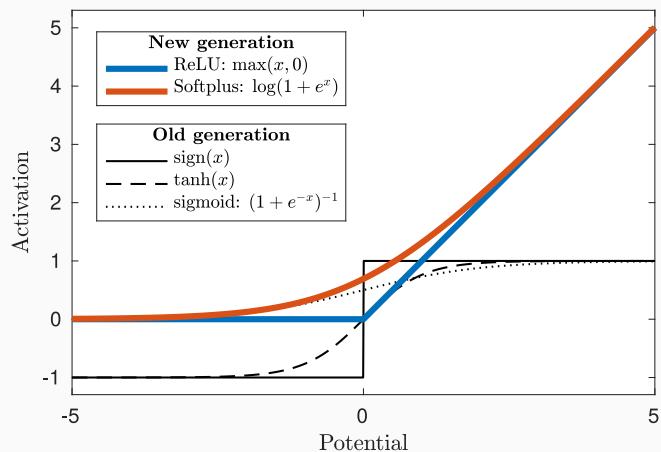
**Trade-off between
generalization and
overfitting.**

Machine learning – ANN – Activation functions

Activation functions

“Modern” units:

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

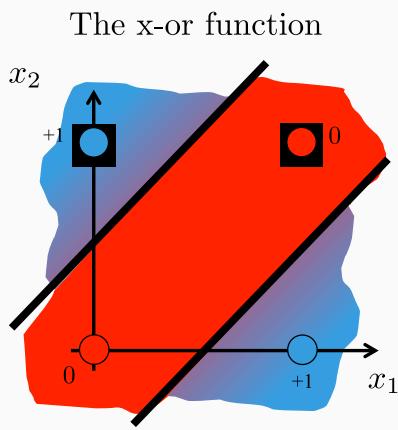


Most neural networks use **ReLU**
(Rectifier linear unit) – $\max(a, 0)$ –
nowadays for hidden layers, since it
trains much faster, is more expressive
than logistic function and prevents the
gradient vanishing problem (see later).

(Source: Lucas Masuch)

Machine learning – ANN

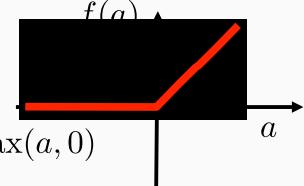
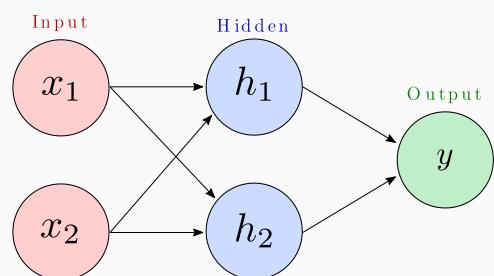
Neural networks solve non-linear separable problems



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

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

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



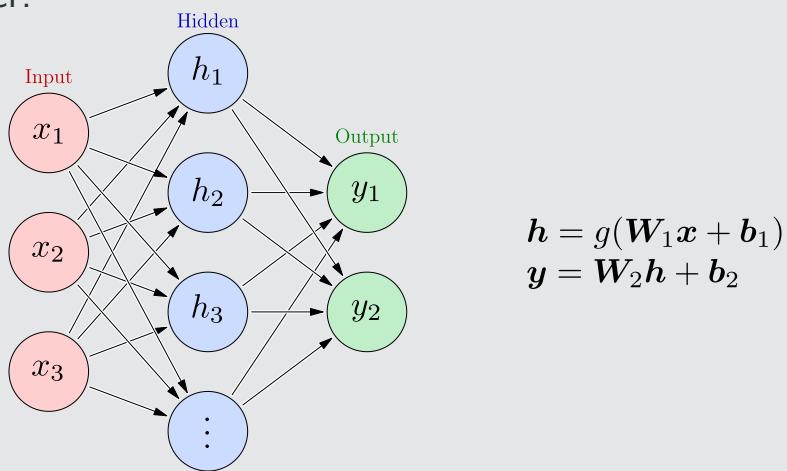
(Source: Vincent Lepetit)

Machine learning – UAT

Universal Approximation Theorem

(Hornik et al, 1989; Cybenko, 1989)

Any continuous function can be approximated by a feedforward **shallow network** (i.e., with 1-hidden layer only) with a sufficient number of neurons in the hidden layer.



- The theorem does not say how large the network needs to be.
- No guarantee that the training algorithm will be able to train the network.

Tasks, architectures and loss functions



Tasks, architectures and loss functions

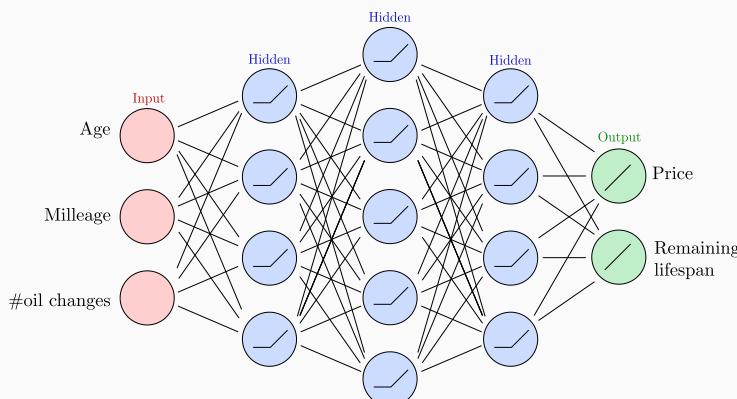
Approximation – Least square regression

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

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

↓
 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:**

$$g(a) = a$$

Tasks, architectures and loss functions

Approximation – Least square regression

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

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

and look for \mathbf{W}^* such that $\nabla E(\mathbf{W}^*) = 0$. *Recall: SSE \equiv SSD \equiv MSE*

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

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

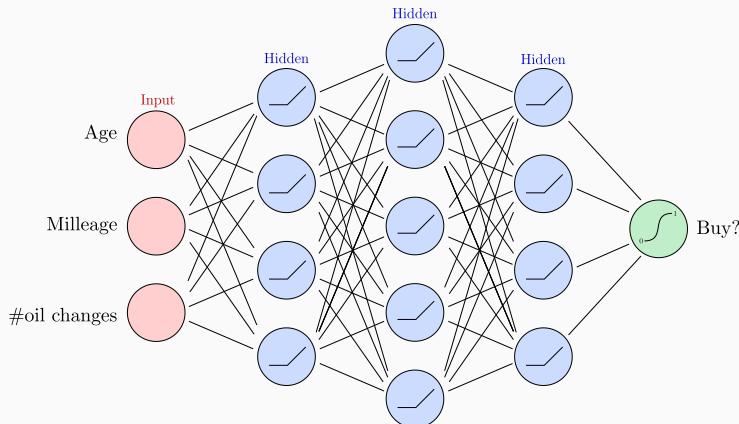
Tasks, architectures and loss functions

Binary classification – Logistic regression

- **Goal:** Classify object x into class C_1 or C_2 .
- **How:** estimate the coefficients \mathbf{W} of a real function $y = f(\mathbf{x}; \mathbf{W}) \in [0, 1]$ from training examples with labels 1 (for class C_1) and 0 (otherwise):

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

- **Typical architecture:**



- **Hidden layer:**

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

- **Output layer:**

$$\text{logistic}(x) = \frac{1}{1 + e^{-x}}$$

Tasks, architectures and loss functions

Binary classification – Logistic regression

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

$$E(\mathbf{W}) = - \sum_{i=1}^N d^i \log y^i + (1 - d^i) \log(1 - y^i) \quad \text{with} \quad y^i = f(\mathbf{x}^i; \mathbf{W})$$

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

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

$$y^* = f(\mathbf{x}; \mathbf{W}^*) = \underbrace{\mathbb{P}(C_1 | \mathbf{x})}_{\text{posterior probability}}$$

Tasks, architectures and loss functions

Multiclass classification – Multivariate logistic regression (aka, multinomial classification)

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

$$\mathbf{y} = f(\mathbf{x}; \mathbf{W}) \in [0, 1]^K$$

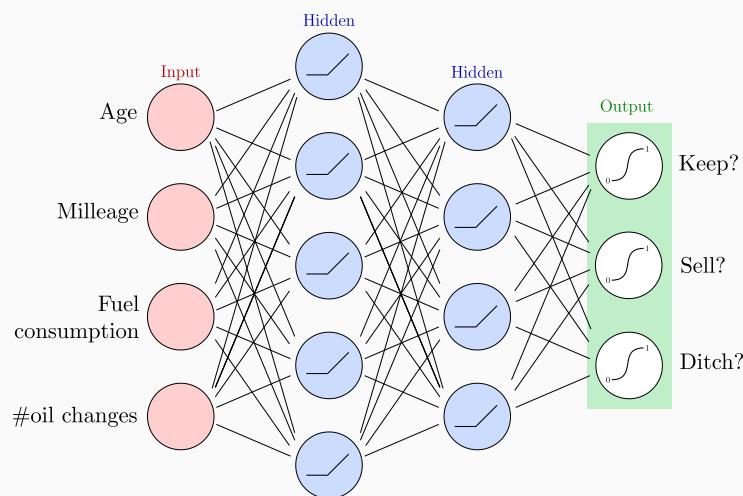
from training examples $\mathcal{T} = \{(\mathbf{x}^i, \mathbf{d}^i)\}$ where \mathbf{d}^i is a 1-of-K (one-hot) code

- Class 1: $\mathbf{d}^i = (1, 0, \dots, 0)^T$ if $\mathbf{x}^i \in C_1$
 - Class 2: $\mathbf{d}^i = (0, 1, \dots, 0)^T$ if $\mathbf{x}^i \in C_2$
 - ...
 - Class K: $\mathbf{d}^i = (0, 0, \dots, 1)^T$ if $\mathbf{x}^i \in C_K$
-
- **Remark:** do not use the class index k directly as a scalar label.

Tasks, architectures and loss functions

Multiclass classification – Multivariate logistic regression

- Typical architecture:



- Hidden layer:

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

- Output layer:

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

Softmax guarantees the outputs y_k to be positive and sum to 1.

Generalization of the logistic sigmoid activation function.

Smooth version of winner-takes-all activation model (maxout).
(largest gets +1 others get 0).

Tasks, architectures and loss functions

Multiclass classification – Multivariate logistic regression

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

$$E(\mathbf{W}) = - \sum_{i=1}^N \sum_{k=1}^K d_k^i \log y_k^i \quad \text{with} \quad \mathbf{y}^i = f(\mathbf{x}^i; \mathbf{W})$$

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

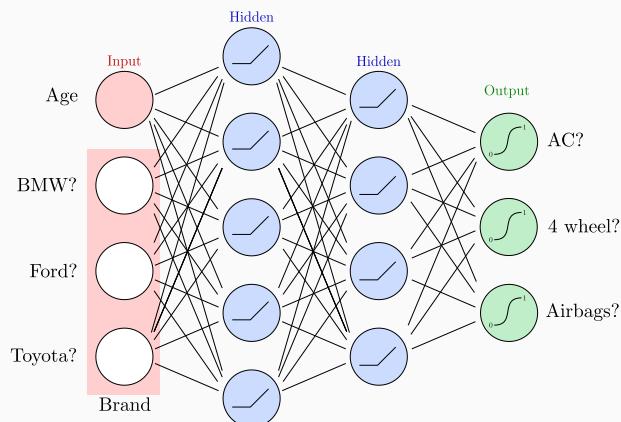
- **Solution:** Provided the network has enough flexibility and the size of the training set grows to infinity

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

Machine learning – ANN

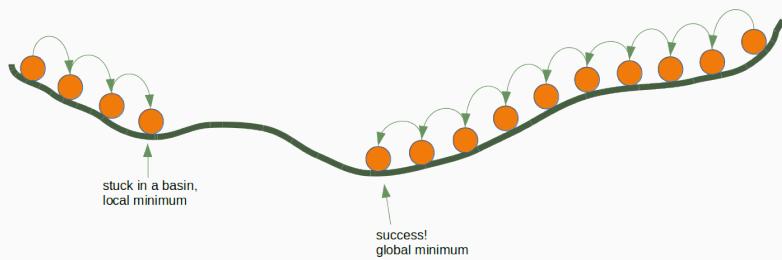
Multi-label classification: (aka, multi-output classification)

- **Goal:** Classify object x into zero or several classes C_1, \dots, C_K .
The classes need to be non-mutually exclusive.
- **How:** Combination of binary-classification networks.
- **Typical architecture:**



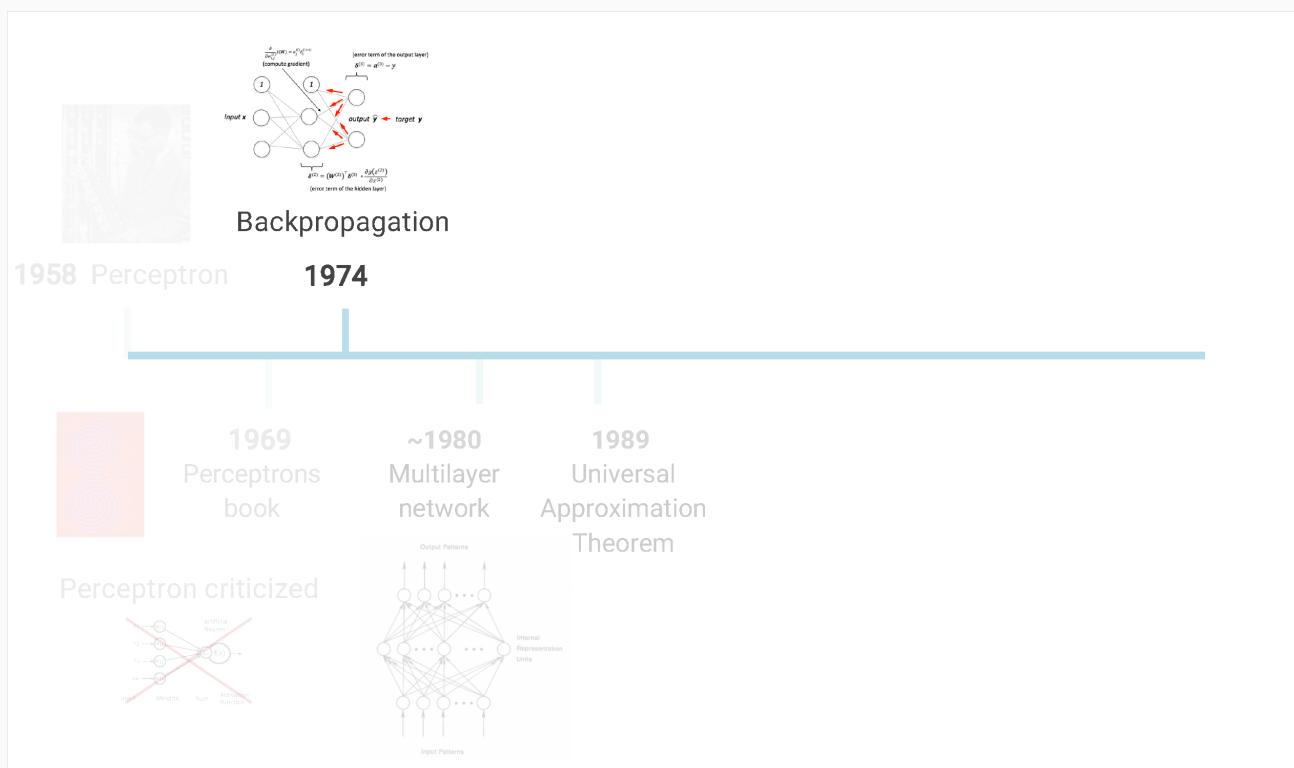
- **Remark:** For categorical inputs, use also 1-of-K codes.

Backpropagation



Machine learning – ANN - Backpropagation

Learning with backpropagation



(Source: Lucas Masuch & Vincent Lepetit)

Machine learning – ANN – Learning

Training process

- To train a neural network over a large set of labeled data, you must continuously compute the difference between the network's predicted output and the actual output.
- This difference is measured by the loss E , and the process for training a net is known as **backpropagation**, or **backprop**.
- During backprop, **weights and biases are tweaked** slightly until the lowest possible loss is achieved.

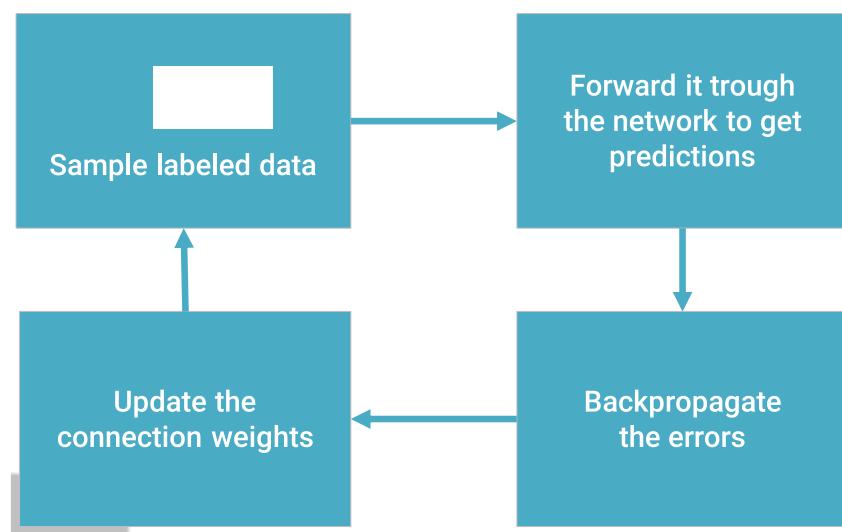
→ **Optimization: look for $\nabla E = 0$**

- The gradient is an important aspect of this process, as a measure of how much the loss changes with respect to a change in a weight or bias value.

(Source: Caner Hazırbaş)

Machine learning – ANN – Learning

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)

41

Machine learning – ANN – Optimization

Objective: $\min_{\mathbf{W}} E(\mathbf{W}) \Rightarrow \nabla E(\mathbf{W}) = \left(\frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_1} \quad \dots \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_L} \right)^T = 0$

Loss functions: recall that classical loss functions are

- Square error (for regression: $d_k \in \mathbb{R}$, $y_k \in \mathbb{R}$)

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

- Cross-entropy (for multi-class classification: $d_k \in \{0, 1\}$, $y_k \in [0, 1]$)

$$E(\mathbf{W}) = - \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k d_k \log y_k$$

Solution: no close-form solutions \Rightarrow use **gradient descent**. **What is it?**

Machine learning – ANN – Optimization – Gradient descent

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

Gradient descent

- Let F be a real function, differentiable and lower bounded with a L Lipschitz gradient (see next slide). Then, whatever the initialization x^0 , if $0 < \gamma < 2/L$, the sequence

$$x^{t+1} = x^t - \gamma \nabla F(x^t) ,$$

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

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

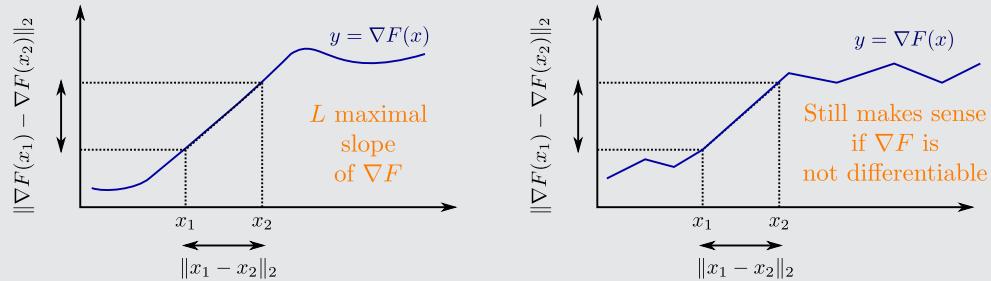
-
- The parameter γ is called the step size (or learning rate in ML field).
 - A too small step size γ leads to slow convergence.

Machine learning – ANN – Optimization – Gradient descent

Lipschitz gradient

- A differentiable function F has L Lipschitz gradient, if

$$\|\nabla F(x_1) - \nabla F(x_2)\|_2 \leq L\|x_1 - x_2\|_2, \quad \text{for all } x_1, x_2.$$



- The mapping $x \mapsto \nabla F(x)$ is necessarily continuous.
- If F is twice differentiable with bounded Hessian, then

$$L = \sup_x \underbrace{\|\nabla^2 F(x)\|_2}_{\text{Hessian matrix of } F}.$$

where for a matrix A , its ℓ_2 -norm $\|A\|_2$ is its maximal singular value.

Machine learning – ANN – Optimization – Gradient descent

Gradient descent and Lipschitz gradient

Example (1d quadratic loss)

- Consider: $F(x) = \frac{1}{2}(x - y)^2$
- We have: $F'(x) = x - y$
- And: $F''(x) = 1$
- Then: $L = \sup_x |F''(x)| = 1$
- Thus: The range for γ is $0 < \gamma < 2/L = 2$
- And: $x^{t+1} = x^t - \gamma(x^t - y)$ converges towards x^* , whatever x^0 , such that $F'(x^*) = 0 \Rightarrow x^* = y$

Question: what is the best value for γ ?

Machine learning – ANN – Optimization – Gradient descent

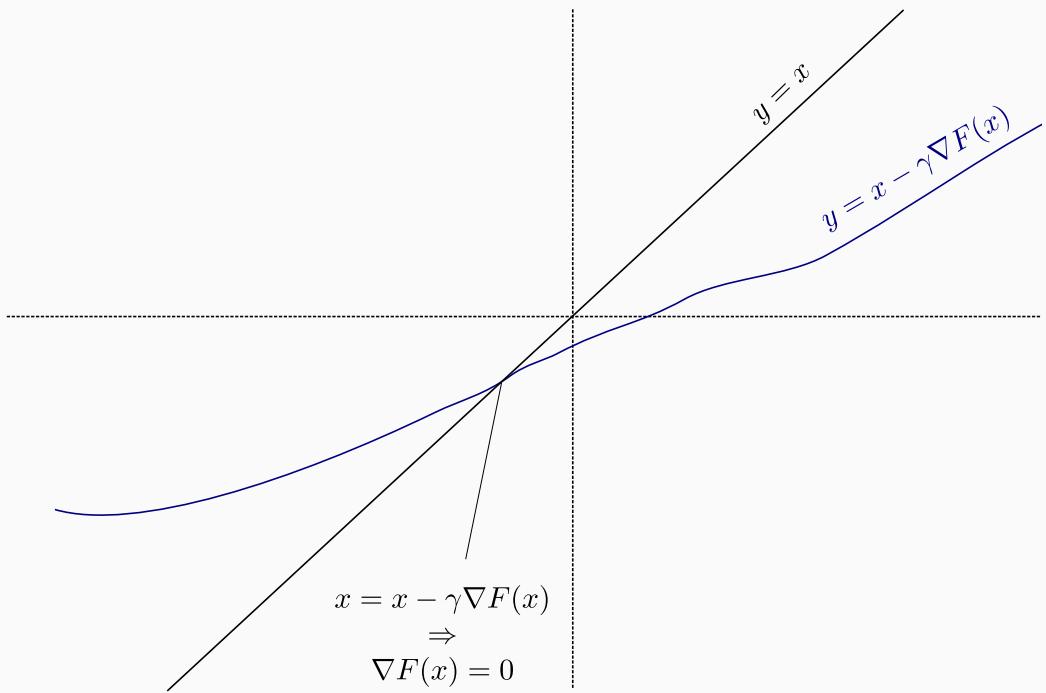
Gradient descent and Lipschitz gradient

Example (1d quartic loss)

- Consider: $F(x) = \frac{1}{4}(x - y)^4$
- We have: $F'(x) = (x - y)^3$
- And: $F''(x) = 3(x - y)^2$
- Then: $L = \sup_x |F''(x)| = \infty$
- Thus: There are no γ satisfying $0 < \gamma < 2/L$
- And: $x^{t+1} = x^t - \gamma(x^t - y)^3$, for $\gamma > 0$, may diverge,
oscilate forever or converge to the solution y .

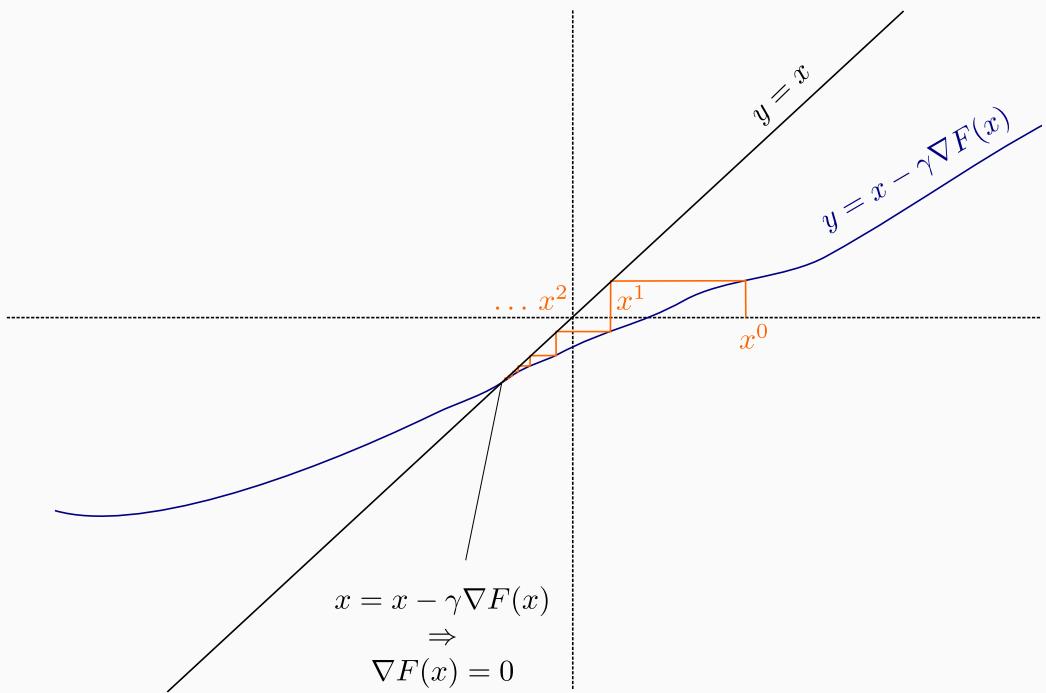
Convergence can be obtained for some specific choice
of γ and initialization x^0 .

Machine learning – ANN – Optimization – Gradient descent



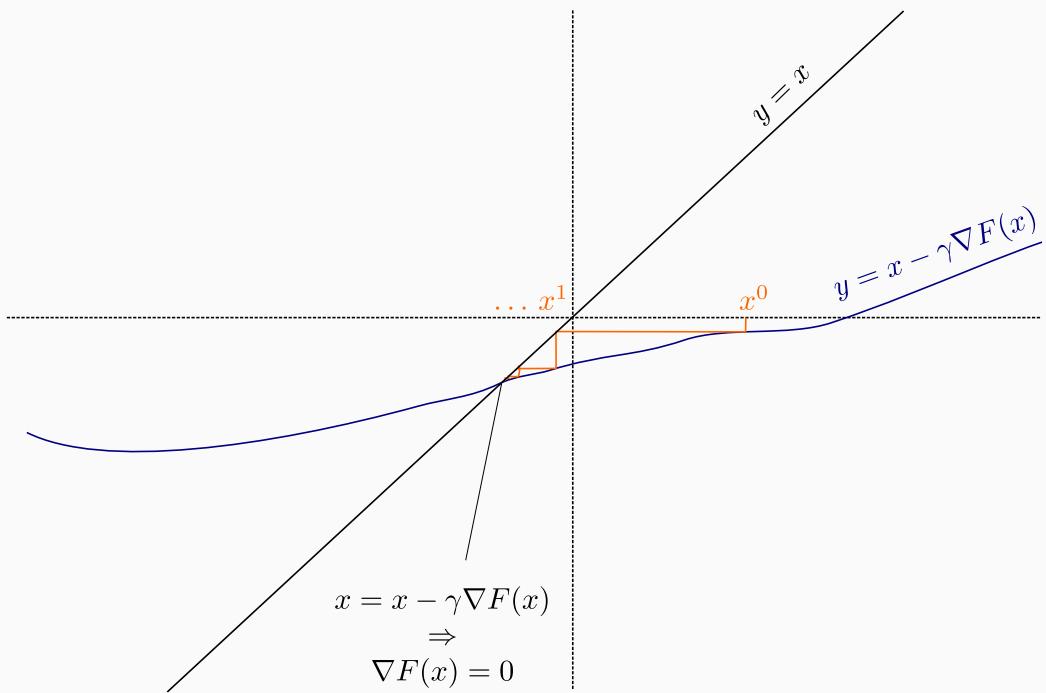
These two curves cross at x^* such that $\nabla F(x^*) = 0$

Machine learning – ANN – Optimization – Gradient descent



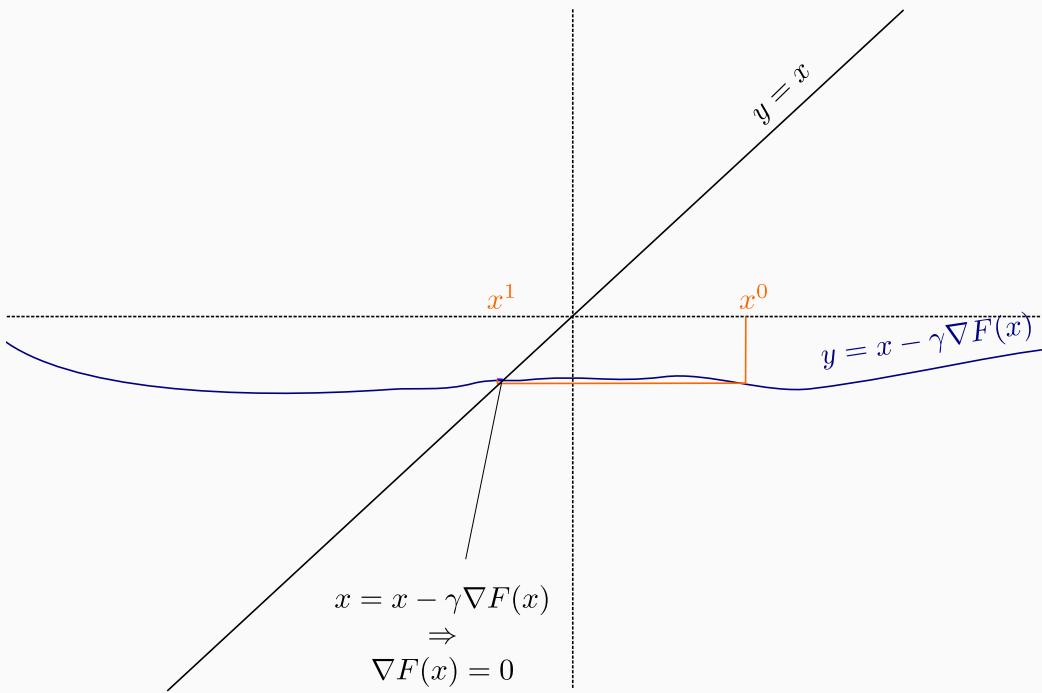
Here γ is small: slow convergence

Machine learning – ANN – Optimization – Gradient descent



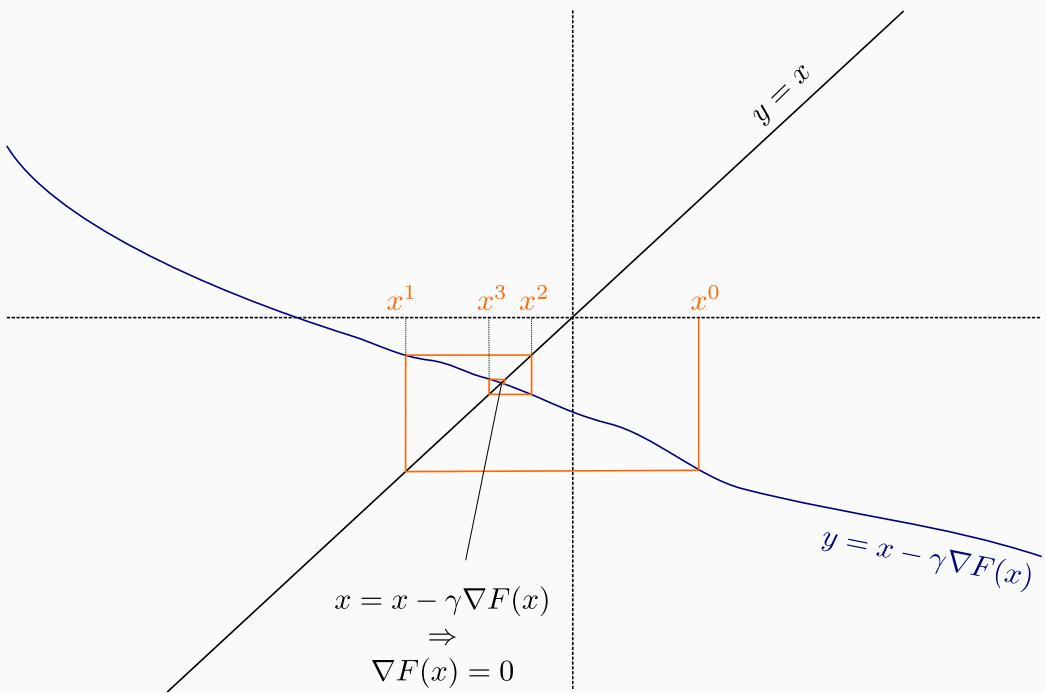
γ a bit larger: faster convergence

Machine learning – ANN – Optimization – Gradient descent



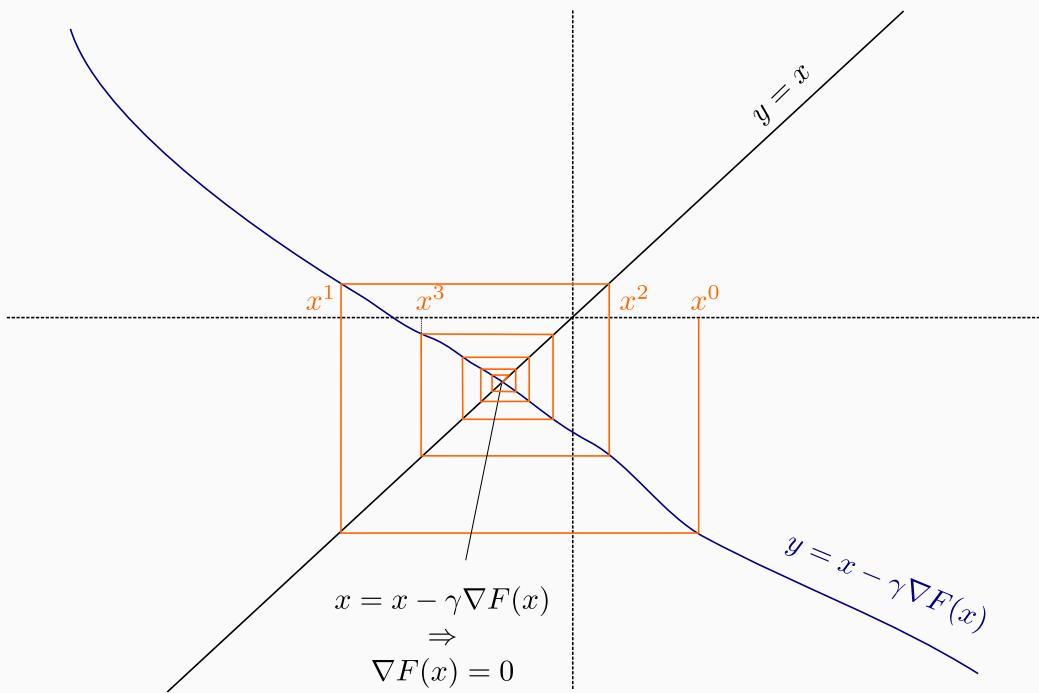
$\gamma \approx 1/L$ even larger: around fastest convergence

Machine learning – ANN – Optimization – Gradient descent



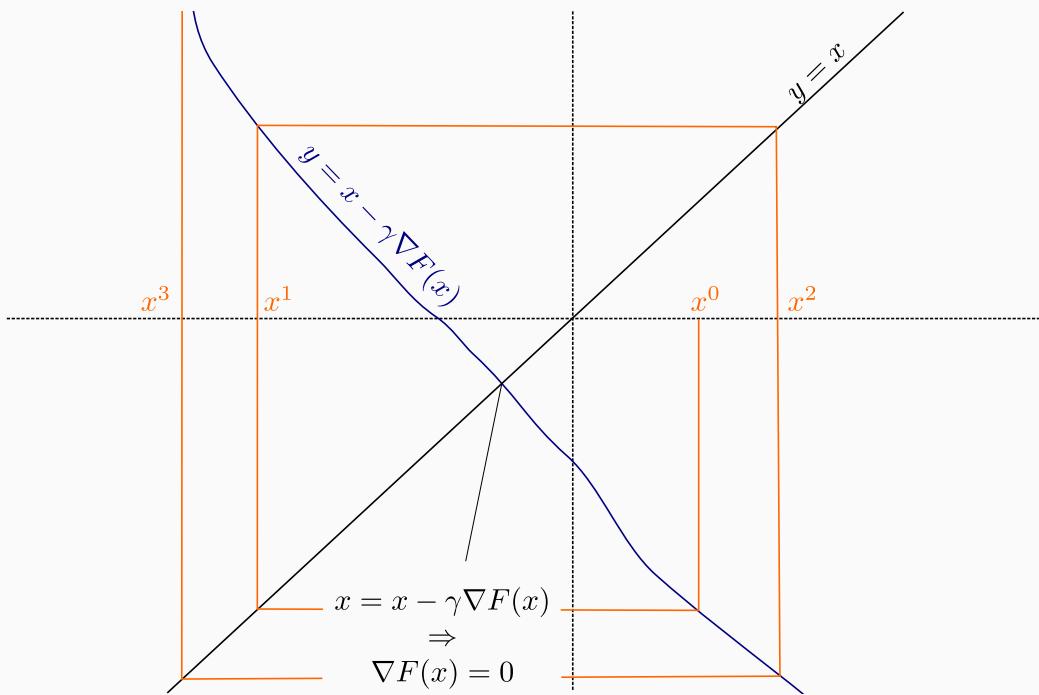
γ a bit too large: convergence slows down

Machine learning – ANN – Optimization – Gradient descent



γ too large: convergence too slow again

Machine learning – ANN – Optimization – Gradient descent



$\gamma > 2/L$: divergence

Machine learning – ANN – Optimization – Gradient descent

Gradient descent for convex function

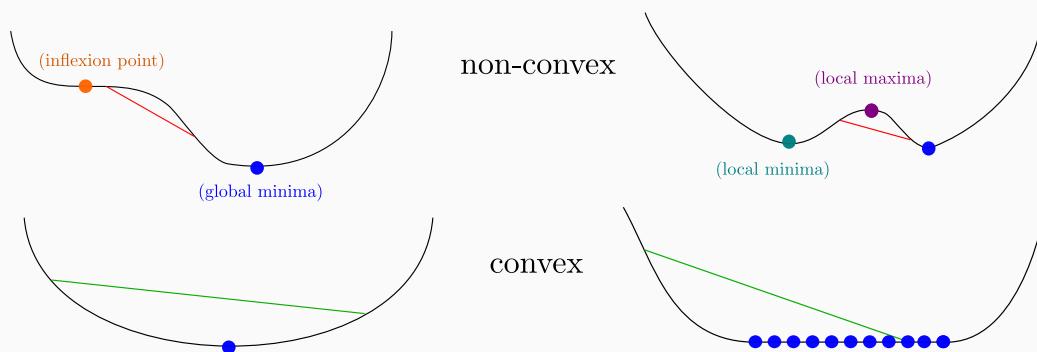
- If moreover F is **convex**

$$F(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda F(x_1) + (1 - \lambda)F(x_2), \quad \forall x_1, x_2, \lambda \in (0, 1) ,$$

then, the gradient descent converges towards a **global minimum**

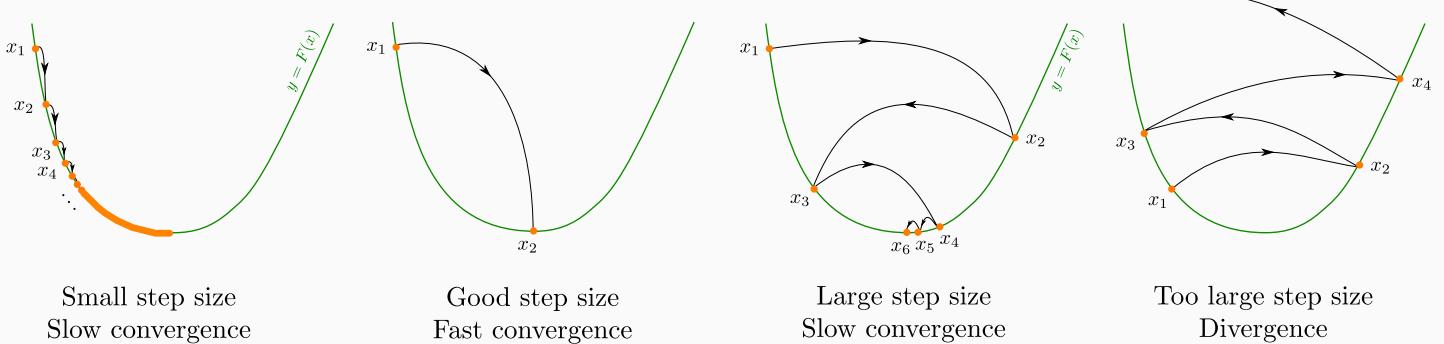
$$x^* \in \operatorname{argmin}_x F(x).$$

- For $0 < \gamma \leq 1/L$, the sequence $|F(x^k) - F(x^*)|$ decays in $O(1/k)$.
- NB: All stationary points are global minima (non necessarily unique).

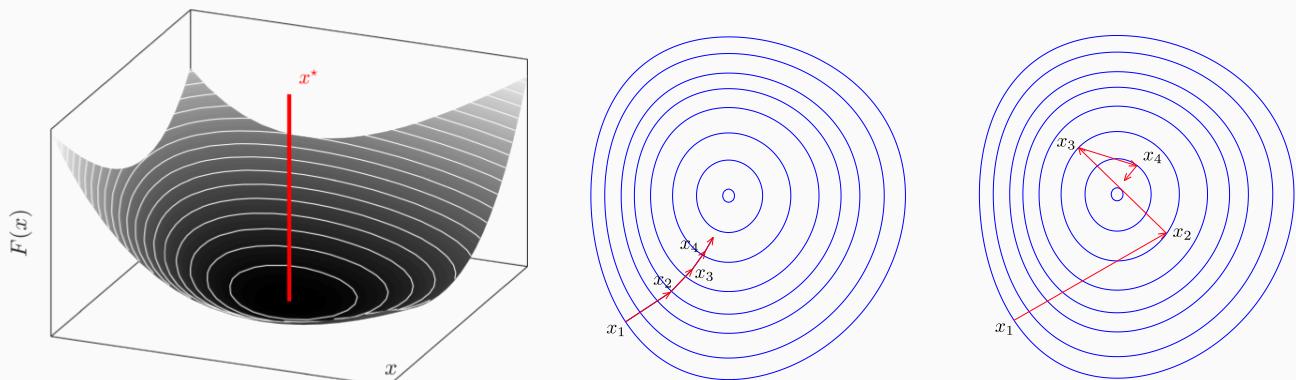


Machine learning – ANN – Optimization – Gradient descent

One-dimension

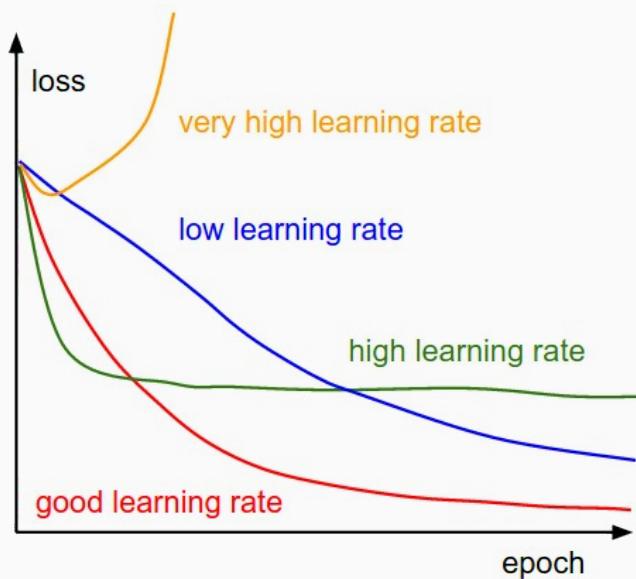


Two-dimensions



Machine learning – ANN – Optimization – Gradient descent

Influence of the step parameter (learning rate)



Machine learning – ANN – Optimization

Let's start with a single artifical neurons

Example ((Batch) Perceptron algorithm)

- Model: $y = \text{sign}\langle \mathbf{w}, \mathbf{x} \rangle$
- Loss:
$$E(\mathbf{w}) = - \sum_{\substack{(\mathbf{x}, d) \in \mathcal{T} \\ \text{st } y \neq d}} d \times \langle \mathbf{w}, \mathbf{x} \rangle$$
- Gradient:
$$\nabla E(\mathbf{w}) = - \sum_{\substack{(\mathbf{x}, d) \in \mathcal{T} \\ \text{st } y \neq d}} d \times \mathbf{x}$$
- Gradient descent:
$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \gamma \sum_{\substack{(\mathbf{x}, d) \in \mathcal{T} \\ \text{st } y^t \neq d}} d \times \mathbf{x}$$

Convex or non-convex?

Machine learning – ANN – Optimization

Let's start with a single artifical neurons

Example ((Batch) ADALINE)

- Model: $y = \langle \mathbf{w}, \mathbf{x} \rangle$

- Loss:
$$E(\mathbf{w}) = \frac{1}{2} \sum_{(\mathbf{x}, d) \in \mathcal{T}} \underbrace{(\langle \mathbf{w}, \mathbf{x} \rangle - d)^2}_{=d^2 + \mathbf{w}^T \mathbf{x} \mathbf{x}^T \mathbf{w} - 2d \mathbf{w}^T \mathbf{x}}$$

- Gradient:
$$\nabla E(\mathbf{w}) = \sum_{(\mathbf{x}, d) \in \mathcal{T}} \mathbf{x} (\underbrace{\mathbf{x}^T \mathbf{w} - d}_y)$$

- Gradient descent:
$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \gamma \sum_{(\mathbf{x}, d) \in \mathcal{T}} (d - y^t) \mathbf{x}$$

Convex or non-convex?

Machine learning – ANN – Optimization

Let's start with a single artifical neurons

Example ((Batch) ADALINE)

- Model: $y = \langle \mathbf{w}, \mathbf{x} \rangle$

- Loss:
$$E(\mathbf{w}) = \frac{1}{2} \sum_{(\mathbf{x}, d) \in \mathcal{T}} \underbrace{(\langle \mathbf{w}, \mathbf{x} \rangle - d)^2}_{=d^2 + \mathbf{w}^T \mathbf{x} \mathbf{x}^T \mathbf{w} - 2\mathbf{w}^T \mathbf{x}}$$

- Gradient:
$$\nabla E(\mathbf{w}) = \sum_{(\mathbf{x}, d) \in \mathcal{T}} \mathbf{x} (\underbrace{\mathbf{x}^T \mathbf{w} - d}_y)$$

- Gradient descent:
$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \gamma \sum_{(\mathbf{x}, d) \in \mathcal{T}} (d - y^t) \mathbf{x}$$

Convex or non-convex?

If enough training samples: $\lim_{t \rightarrow \infty} \mathbf{w}^t = \underbrace{\left(\sum_{(\mathbf{x}, d) \in \mathcal{T}} \mathbf{x} \mathbf{x}^T \right)}_{\text{Hessian}}^{-1} \left(\sum_{(\mathbf{x}, d) \in \mathcal{T}} d \mathbf{x} \right)$

Machine learning – ANN – Optimization

Back to our optimization problem

In our case $\mathbf{W} \mapsto E(\mathbf{W})$ is non-convex \Rightarrow No guarantee of convergence.

Convergence will depends on $\left\{ \begin{array}{l} \bullet \text{ the initialization,} \\ \bullet \text{ the step size } \gamma. \end{array} \right.$

Because of this:

- Normalizing each data point x in the range $[-1, +1]$ is important to control for the Hessian \rightarrow the stability and the speed of the algorithm,
- The activation functions and the loss should be chosen to have a second derivative smaller than 1 (when combined),
- The initialization should be random with well chosen variance
(we will go back to this later),
- If all of these are satisfied, we can generally choose $\gamma \in [.001, 1]$.

Machine learning – ANN – Optimization

Back to our optimization problem

In our case $\mathbf{W} \mapsto E(\mathbf{W})$ is non-convex \Rightarrow No guarantee of convergence.

Even if so, the limit solution depends on: $\left\{ \begin{array}{l} \bullet \text{ the initialization,} \\ \bullet \text{ the step size } \gamma. \end{array} \right.$

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

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

Gradient descent can be expressed coordinate by coordinate as:

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

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

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

Machine learning – ANN – Optimization

Backpropagation: computation of $\frac{\partial E(\mathbf{W})}{\partial w_{i,j}}$

Feedforward least square regression context

- **Model:** Feed-forward neural network.
(for simplicity without bias)

- **Loss function:** $E(\mathbf{W}) = \frac{1}{2} \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k (y_k - d_k)^2$

We have:

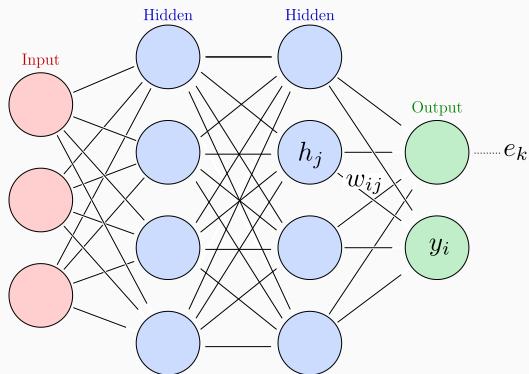
$$E(\mathbf{W}) = \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k \underbrace{\frac{1}{2} (y_k - d_k)^2}_{e_k}$$

Apply linearity:

$$\frac{\partial E(\mathbf{W})}{\partial w_{i,j}} = \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k \frac{\partial e_k}{\partial w_{i,j}}$$

Machine learning – ANN – Optimization

1. Case where w_{ij} is a synaptic weight for the output layer



- j : neuron in the previous hidden layer
- h_j : response of hidden neuron j
- $w_{i,j}$: synaptic weight between j and i
- y_i : response of output neuron i

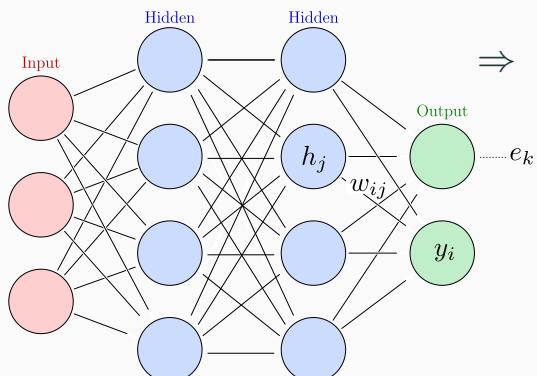
$$y_i = g(a_i) \quad \text{with} \quad a_i = \sum_j w_{i,j} h_j$$

Apply chain rule: $\frac{\partial E(\mathbf{W})}{\partial w_{i,j}} = \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k \frac{\partial e_k}{\partial w_{i,j}} = \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k \frac{\partial e_k}{\partial y_i} \frac{\partial y_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}}$

Machine learning – ANN – Optimization

1. Case where w_{ij} is a synaptic weight for the output layer

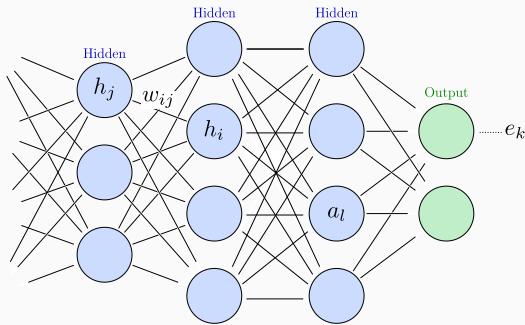
$$\begin{aligned}
 e_k &= \frac{1}{2}(y_k - d_k)^2 \quad \Rightarrow \quad \frac{\partial e_k}{\partial y_i} = \begin{cases} y_i - d_i & \text{if } k = i \\ 0 & \text{otherwise} \end{cases} \\
 y_i &= g(a_i) \quad \Rightarrow \quad \frac{\partial y_i}{\partial a_i} = g'(a_i), \\
 a_i &= \sum_{j'} w_{i,j'} h_{j'} \quad \Rightarrow \quad \frac{\partial a_i}{\partial w_{i,j}} = h_j
 \end{aligned}$$



$$\begin{aligned}
 \frac{\partial E(\mathbf{W})}{\partial w_{i,j}} &= \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k \frac{\partial e_k}{\partial y_i} \frac{\partial y_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}} \\
 &= \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \underbrace{(y_i - d_i) g'(a_i)}_{\delta_i} h_j \\
 &= \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \delta_i h_j \quad \text{where} \quad \delta_i = \sum_k \frac{\partial e_k}{\partial a_i}
 \end{aligned}$$

Machine learning – ANN – Optimization

2. Case where w_{ij} is a synaptic weight for a hidden layer



- j : neuron in the previous hidden layer
- h_j : response of hidden neuron j
- $w_{i,j}$: synaptic weight between j and i
- h_i : response of hidden neuron i

$$h_i = g(a_i) \quad \text{with} \quad a_i = \sum_i w_{i,j} h_j$$

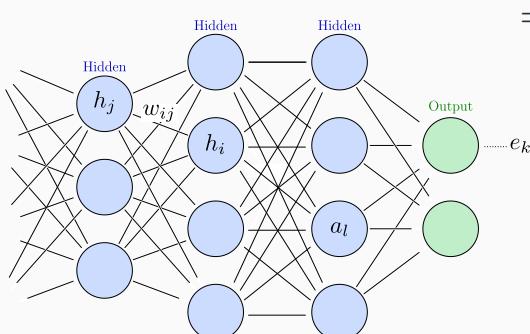
Apply chain rule:

$$\begin{aligned} \frac{\partial E(\mathbf{W})}{\partial w_{i,j}} &= \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k \frac{\partial e_k}{\partial w_{i,j}} = \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k \frac{\partial e_k}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}} \\ &= \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_k \left(\sum_l \frac{\partial e_k}{\partial a_l} \frac{\partial a_l}{\partial h_i} \right) \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}} \\ &= \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_l \underbrace{\left(\sum_k \frac{\partial e_k}{\partial a_l} \right)}_{\delta_l} \frac{\partial a_l}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}} \end{aligned}$$

Machine learning – ANN – Optimization

2. Case where w_{ij} is a synaptic weight for a hidden layer

$$\begin{aligned}
 a_l &= \sum_{i'} w_{l,i'} h_{i'} \quad \Rightarrow \quad \frac{\partial a_l}{\partial h_i} = w_{l,i} \\
 h_i &= g(a_i) \quad \Rightarrow \quad \frac{\partial h_i}{\partial a_i} = g'(a_i), \\
 a_i &= \sum_{j'} w_{i,j'} h_{j'} \quad \Rightarrow \quad \frac{\partial a_i}{\partial w_{i,j}} = h_j
 \end{aligned}$$



$$\begin{aligned}
 \Rightarrow \quad \frac{\partial E(\mathbf{W})}{\partial w_{i,j}} &= \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \sum_l \delta_l \frac{\partial a_l}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}} \\
 &= \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \underbrace{\left(\sum_l w_{l,i} \delta_l \right)}_{\delta_i} g'(a_i) h_j \\
 &= \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \delta_i h_j
 \end{aligned}$$

Machine learning – ANN – Optimization

Backpropagation algorithm

(Werbos, 1974 & Rumelhart, Hinton and Williams, 1986)

$$\frac{\partial E(\mathbf{W})}{\partial w_{i,j}} = \sum_{(\mathbf{x}, \mathbf{d}) \in \mathcal{T}} \delta_i h_j \quad \text{where } h_j = x_j \text{ if } j \text{ is an input node}$$

where $\delta_i = g'(a_i) \times \begin{cases} y_i - d_i & \text{if } i \text{ is the output node} \\ \sum_l w_{l,i} \delta_l & \text{otherwise} \end{cases}$

For all input \mathbf{x} and desired output \mathbf{d}

- Forward step:
 - compute the response (h_j , a_i and y_i) of all neurons,
 - start from the first hidden layer and pursue towards the output one.
- Backward step:
 - Retropropagate the error (δ_i) from the output layer to the first layer.

Update $w_{i,j} \leftarrow w_{i,j} - \gamma \sum \delta_j h_i$, and repeat everything until convergence.

Machine learning – ANN – Optimization

Backpropagation algorithm with matrix-vector form

Easier to use **matrix-vector notations** for each layer:
(k denotes the layer)

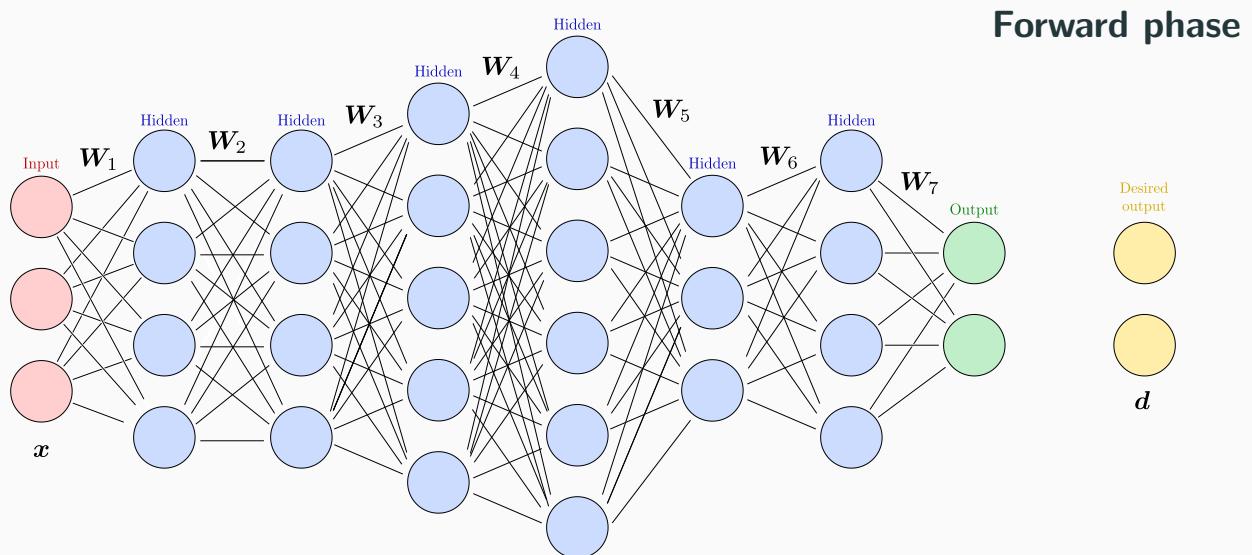
$$\nabla_{\mathbf{W}_k} E(\mathbf{W}) = \boldsymbol{\delta}_k \mathbf{h}_{k-1}^T \quad \text{where } \mathbf{h}_0 = \mathbf{x}$$

$$\text{where } \boldsymbol{\delta}_k = \underbrace{g'(\mathbf{a}_k)}_{\text{element-wise}} \times \begin{cases} \mathbf{y} - \mathbf{d} & \text{if } k \text{ is an output layer} \\ \mathbf{W}_{k+1}^T \boldsymbol{\delta}_{k+1} & \text{otherwise} \end{cases}$$

- \mathbf{x} : matrix with all training input vectors in column,
- \mathbf{d} : matrix with corresponding desired target vectors in column,
- \mathbf{y} : matrix with all predictions in column,
- $\mathbf{a}_k = \mathbf{W}_k \mathbf{h}_{k-1}$: matrix with all weighted sums in column,
- $\mathbf{h}_k = g(\mathbf{a}_k)$: matrix with all hidden outputs in column,
- \mathbf{W}_k : matrix of weights at layer k ,

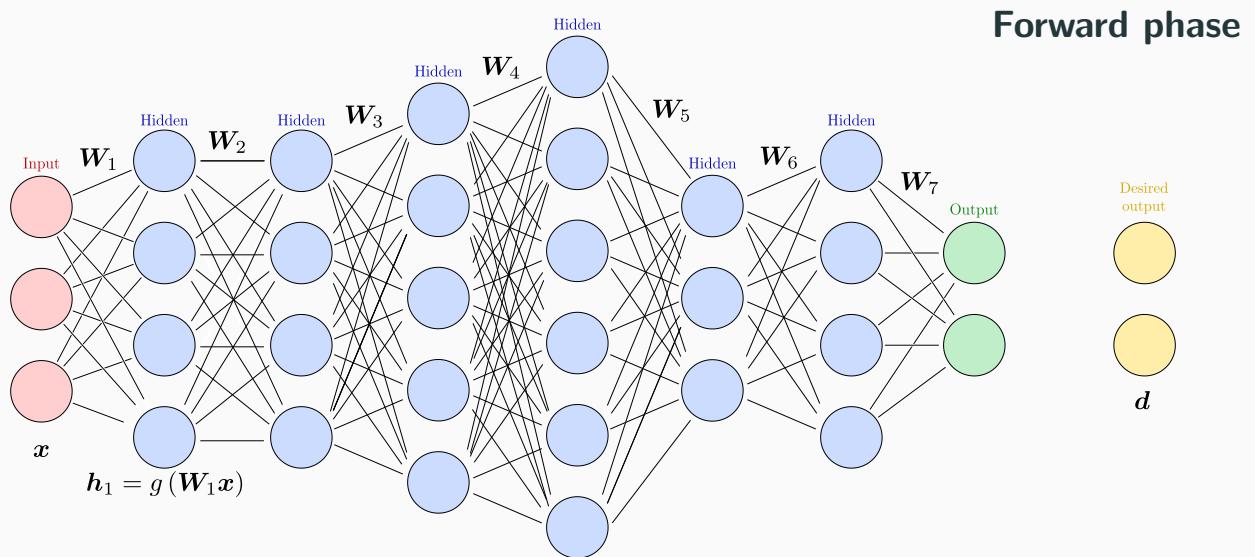
Machine learning – ANN – Optimization

Backpropagation algorithm



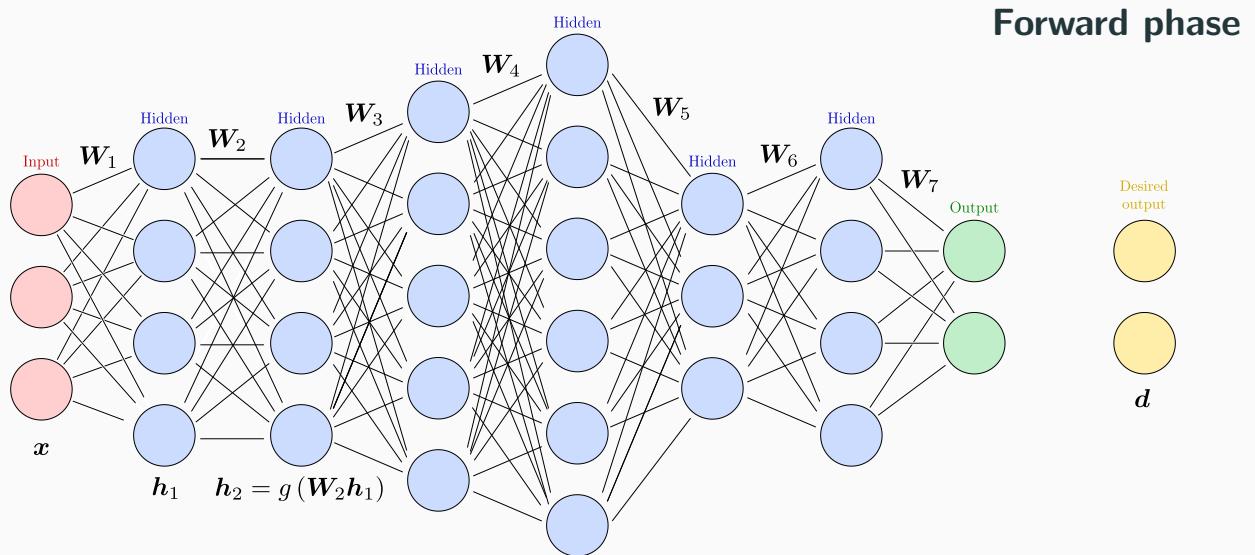
Machine learning – ANN – Optimization

Backpropagation algorithm



Machine learning – ANN – Optimization

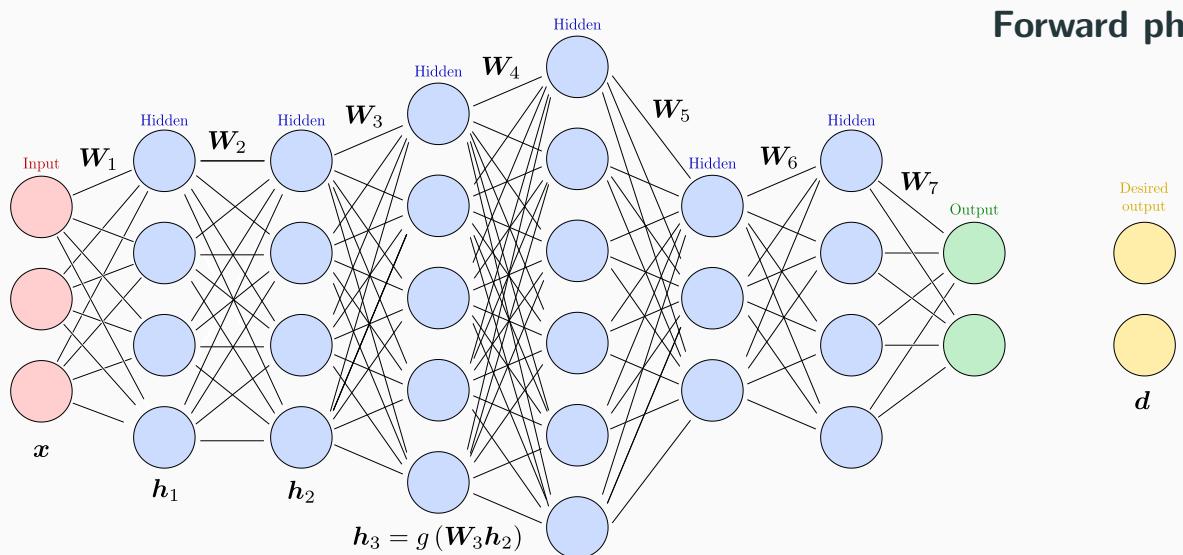
Backpropagation algorithm



Machine learning – ANN – Optimization

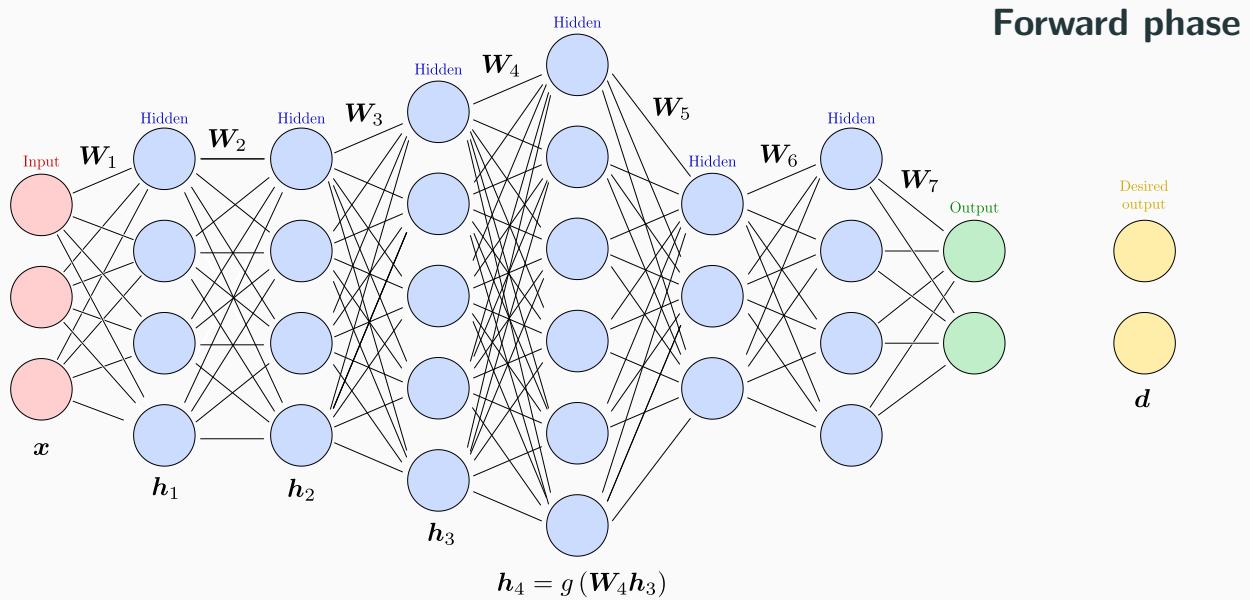
Backpropagation algorithm

Forward phase



Machine learning – ANN – Optimization

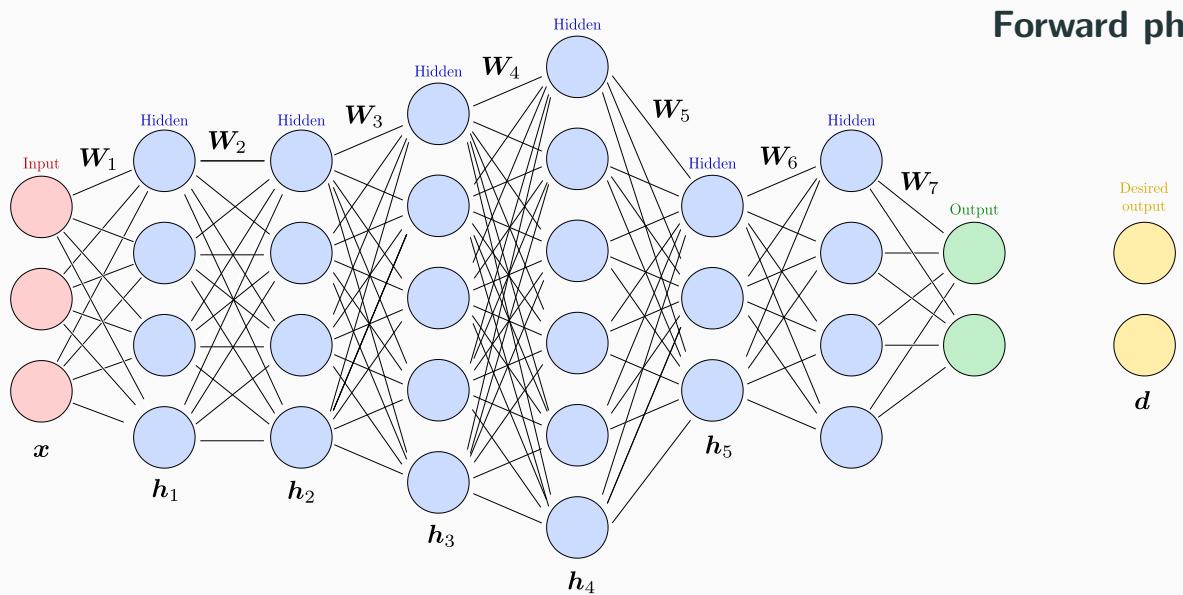
Backpropagation algorithm



Machine learning – ANN – Optimization

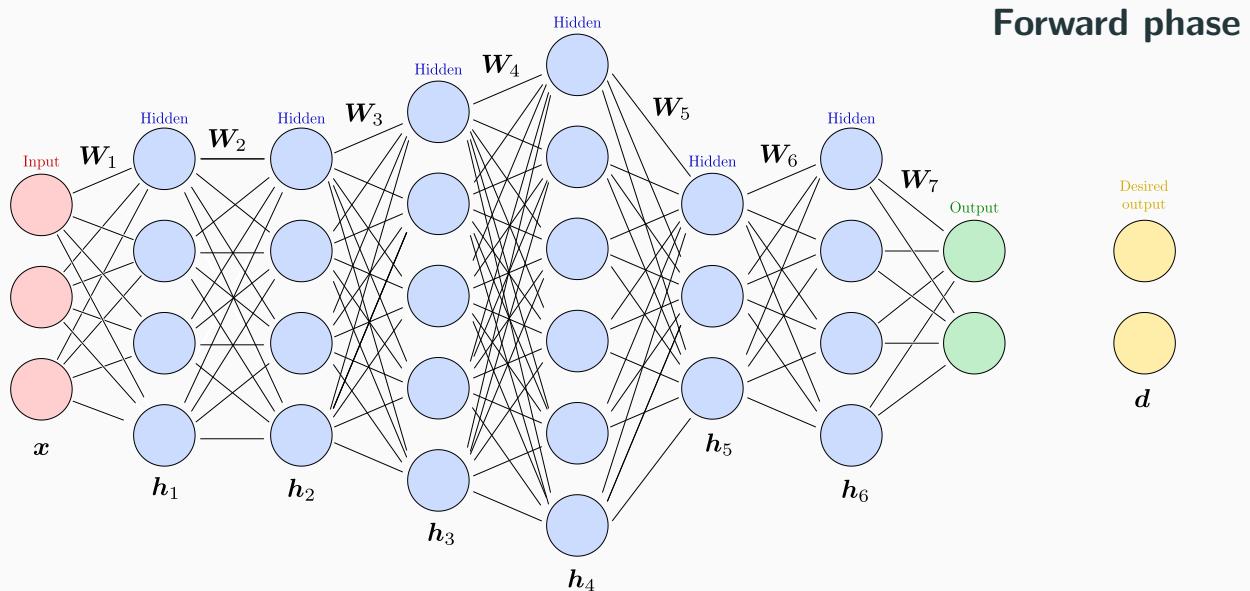
Backpropagation algorithm

Forward phase



Machine learning – ANN – Optimization

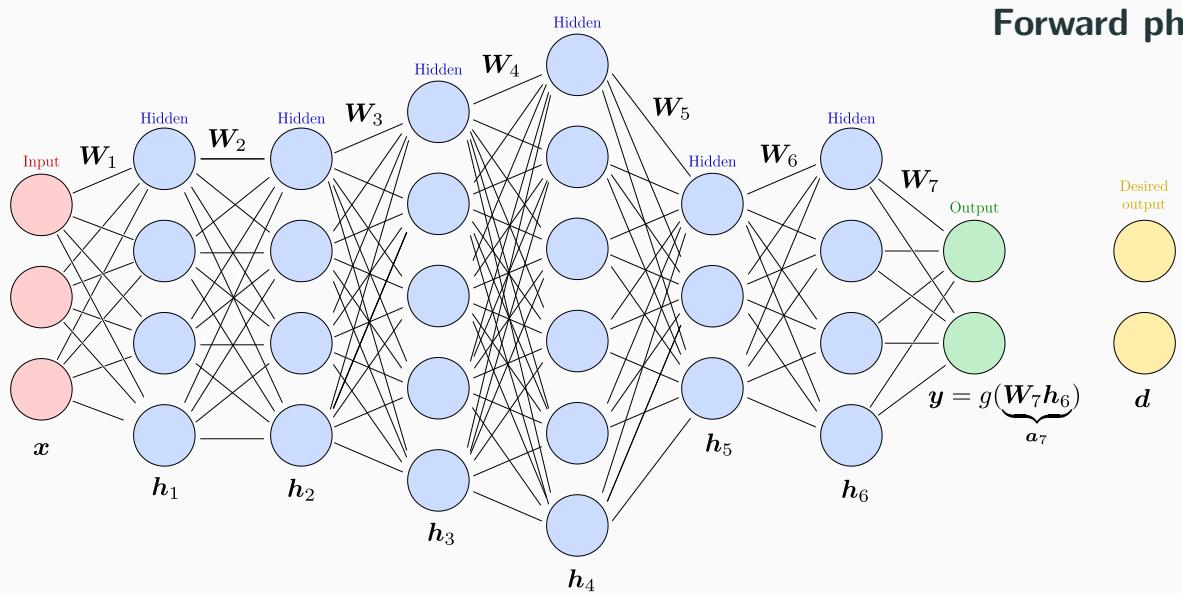
Backpropagation algorithm



Machine learning – ANN – Optimization

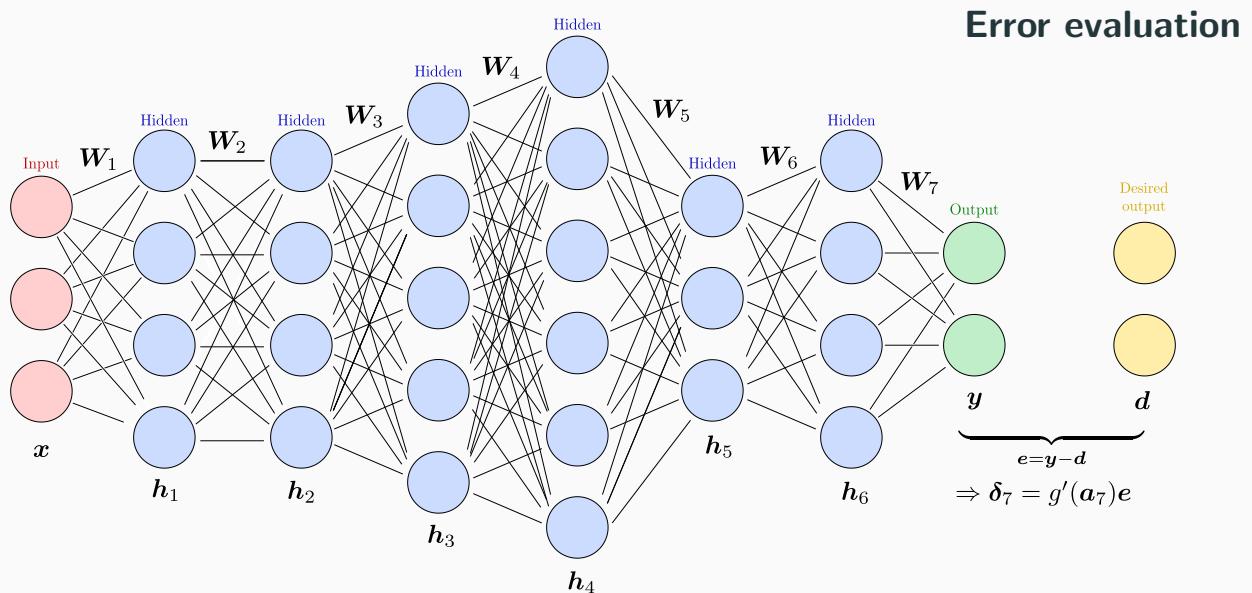
Backpropagation algorithm

Forward phase



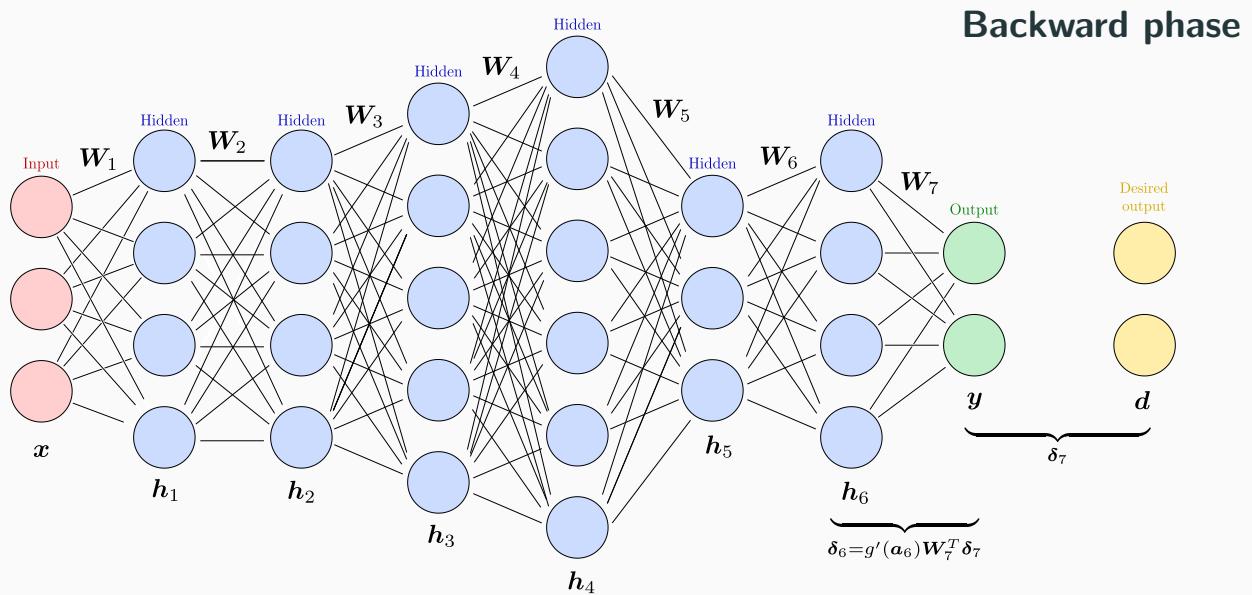
Machine learning – ANN – Optimization

Backpropagation algorithm



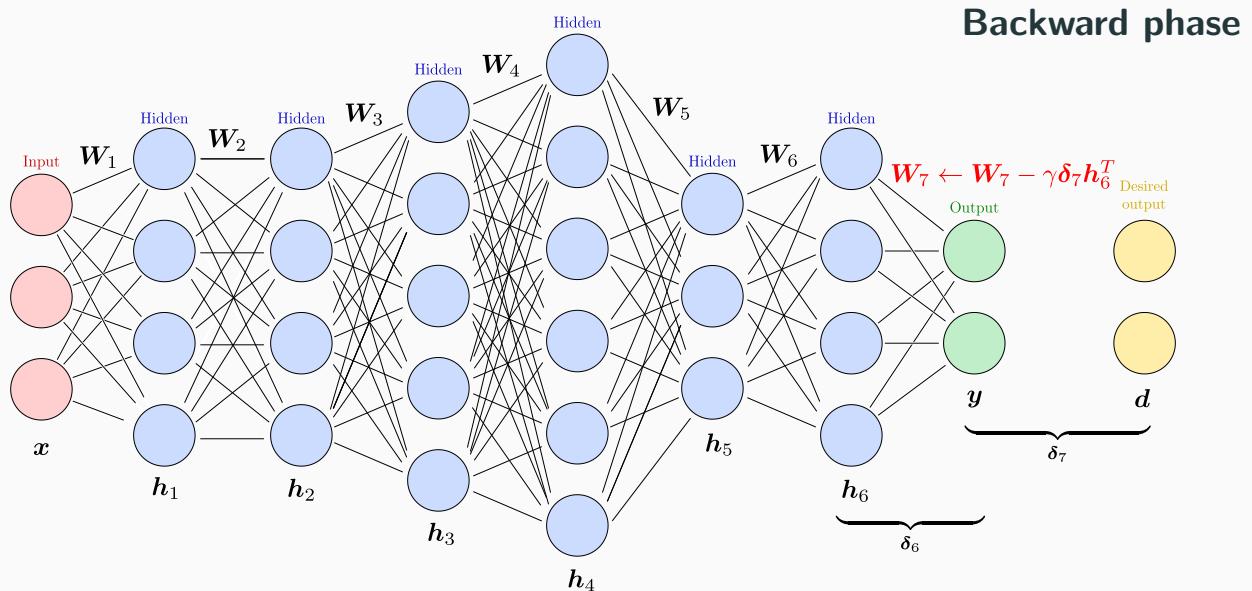
Machine learning – ANN – Optimization

Backpropagation algorithm



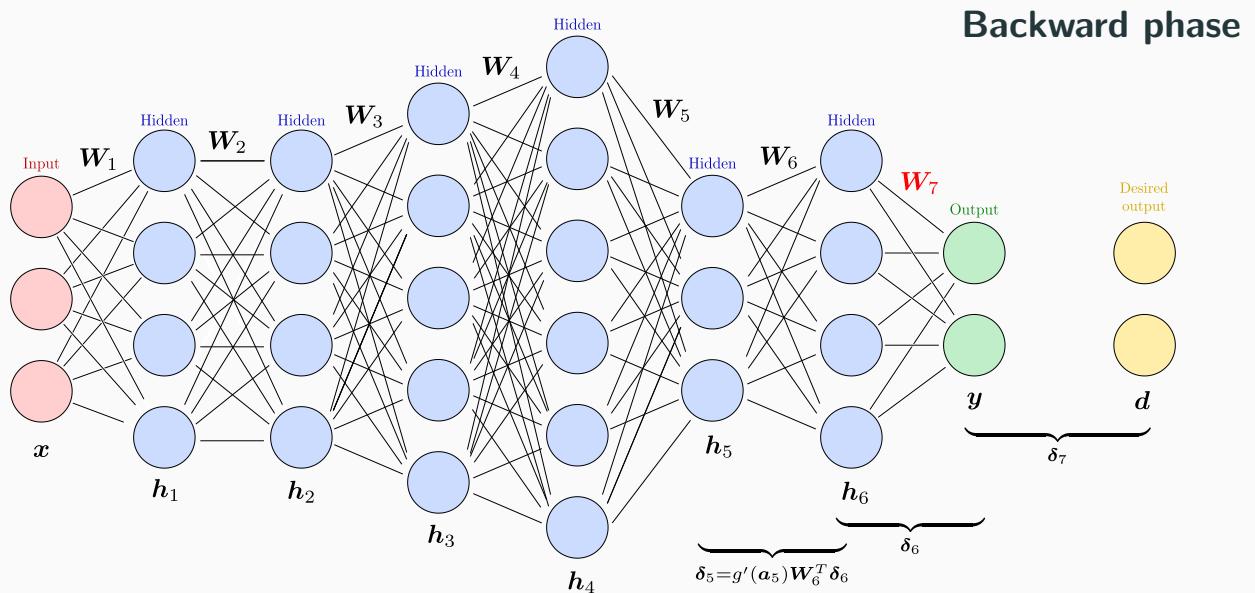
Machine learning – ANN – Optimization

Backpropagation algorithm



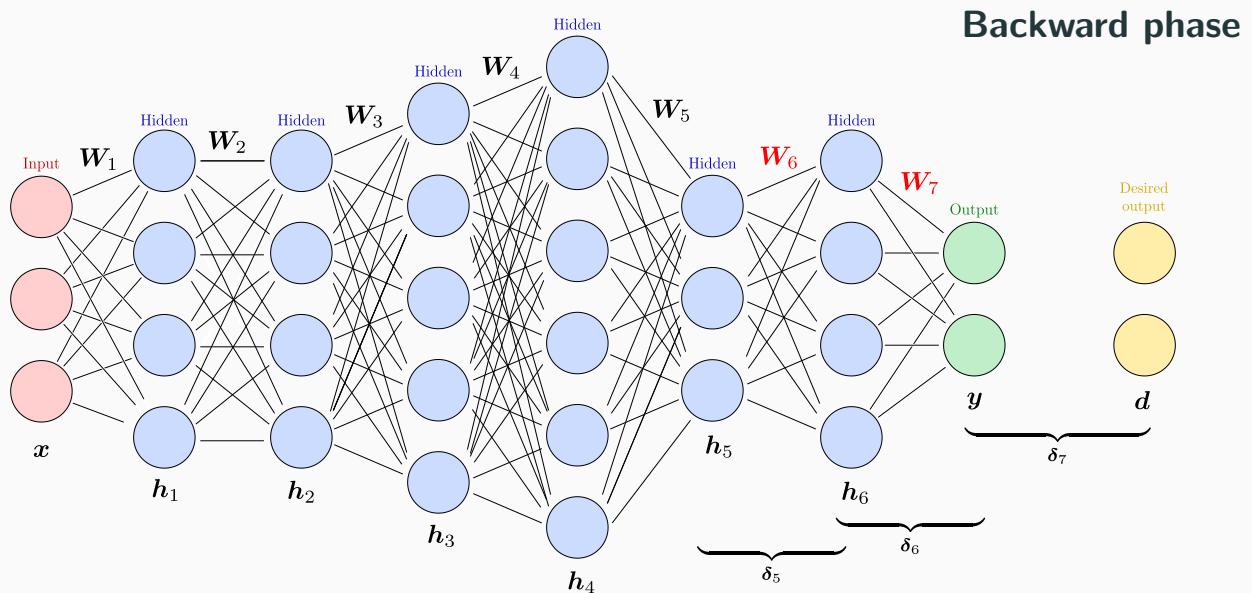
Machine learning – ANN – Optimization

Backpropagation algorithm



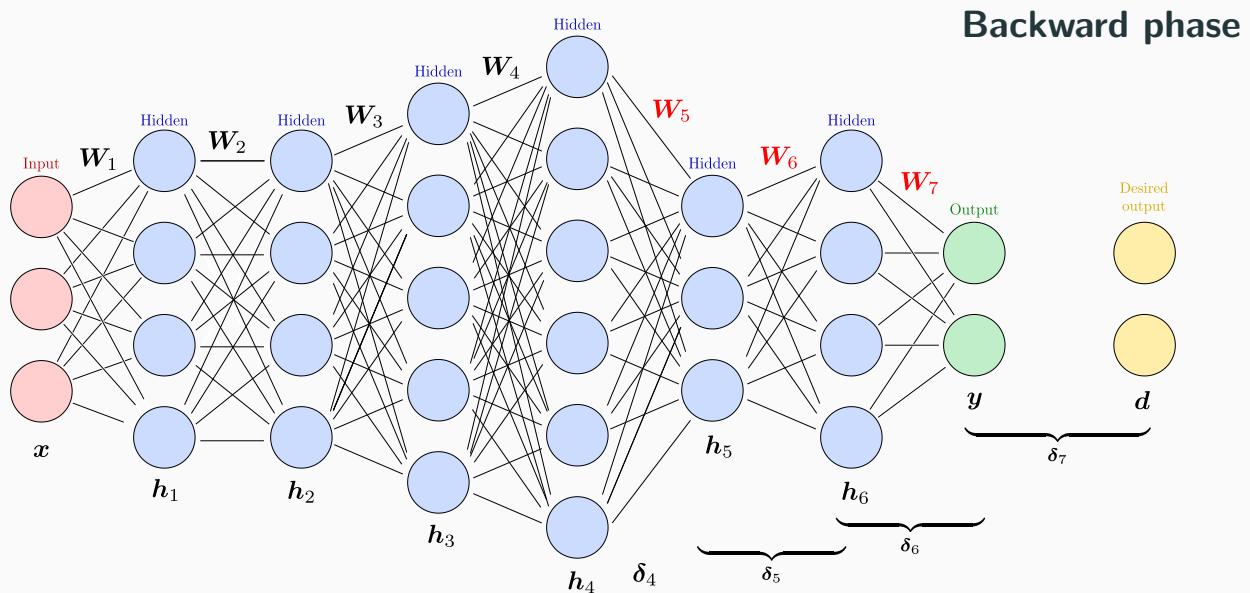
Machine learning – ANN – Optimization

Backpropagation algorithm



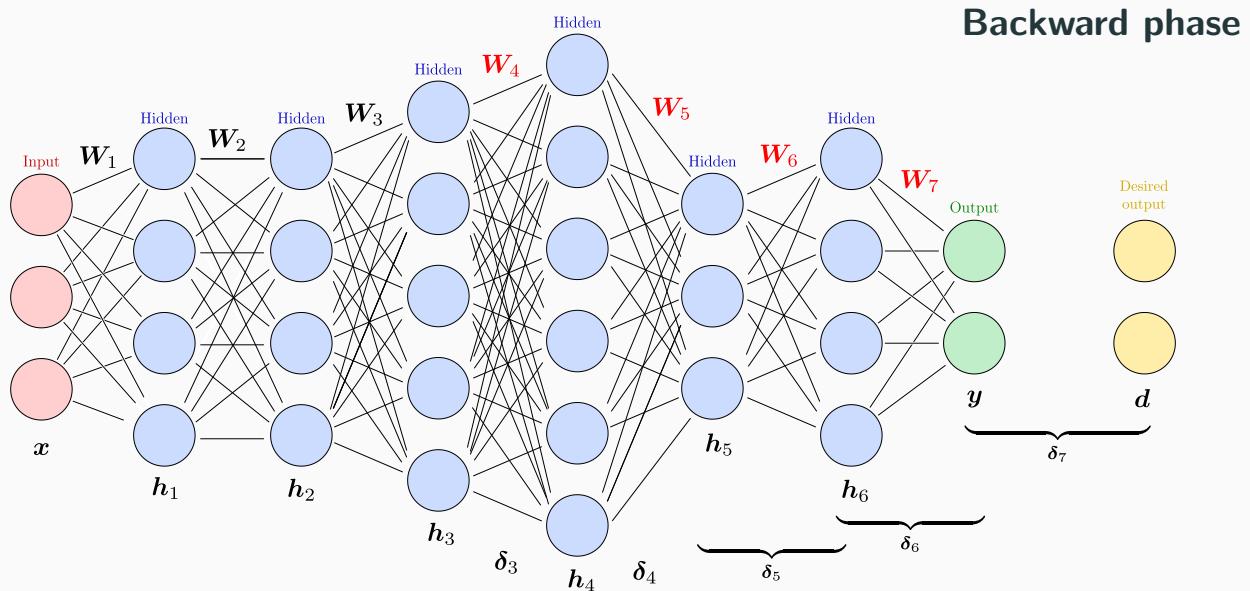
Machine learning – ANN – Optimization

Backpropagation algorithm



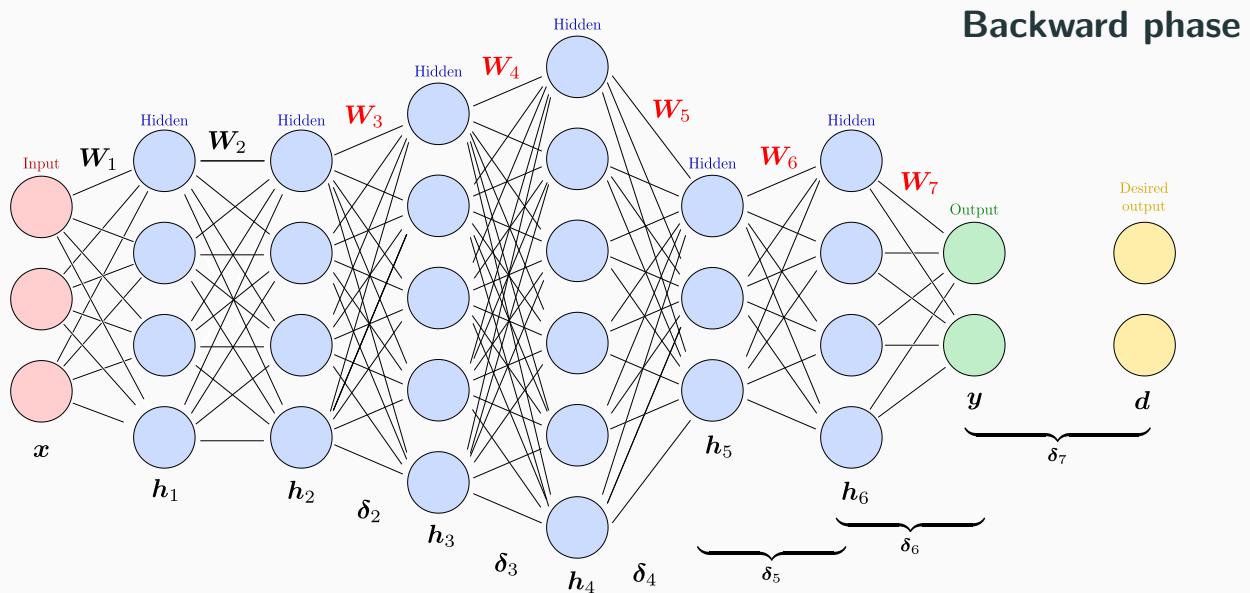
Machine learning – ANN – Optimization

Backpropagation algorithm



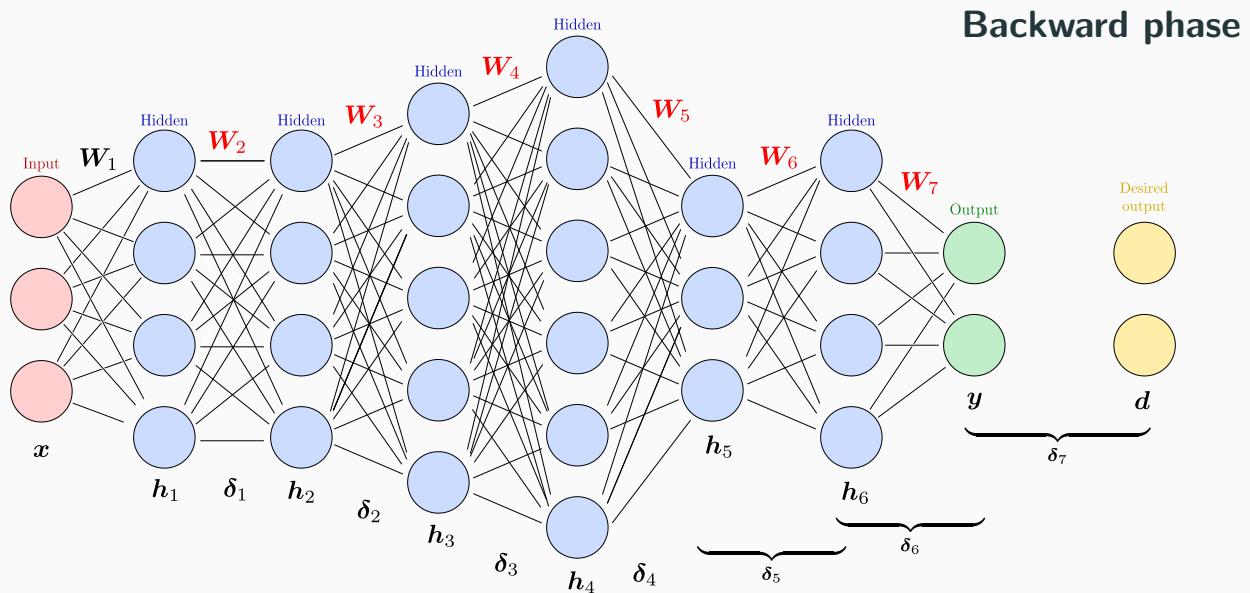
Machine learning – ANN – Optimization

Backpropagation algorithm



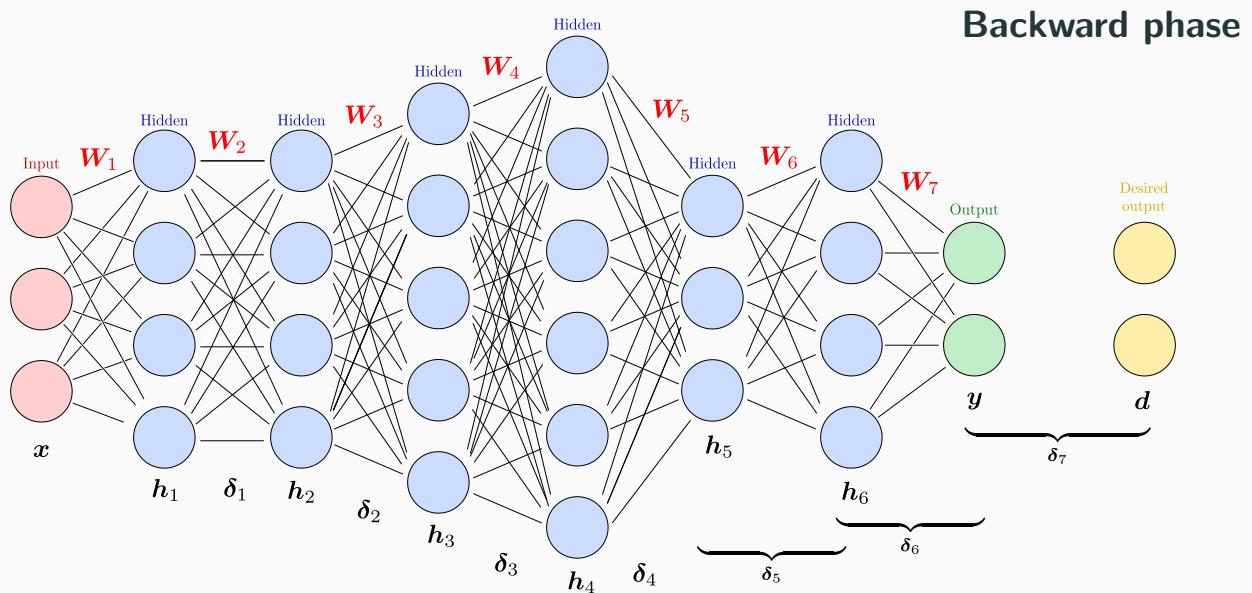
Machine learning – ANN – Optimization

Backpropagation algorithm



Machine learning – ANN – Optimization

Backpropagation algorithm



Machine learning – ANN – Optimization

Case where g is the logistic sigmoid function

- Recall that the logistic sigmoid function is given by:

$$g(a) = \frac{1}{1 + e^{-a}} = \frac{1}{u(a)} \quad \text{where} \quad u(a) = 1 + e^{-a}$$

- We have:

$$u'(a) = -e^{-a}$$

- Then, we get

$$\begin{aligned} g'(a) &= \frac{-u'(a)}{u(a)^2} = \frac{e^{-a}}{(1 + e^{-a})^2} = \frac{1 + e^{-a}}{(1 + e^{-a})^2} - \frac{1}{(1 + e^{-a})^2} \\ &= \frac{1}{1 + e^{-a}} - \frac{1}{(1 + e^{-a})^2} = \frac{1}{1 + e^{-a}} \left(1 - \frac{1}{1 + e^{-a}} \right) \\ &= g(a)(1 - g(a)) \end{aligned}$$

Machine learning – ANN – Optimization

What if g is non-differentiable?

For instance: $g(a) = \text{ReLU}(a) = \max(a, 0)$

- ReLU is **continuous and almost everywhere differentiable**:

$$\frac{\partial \max(a, 0)}{\partial a} = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{if } a < 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- Gradient descent can handle this case with a simple modification:

$$\frac{\partial \max(a, 0)}{\partial a} = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{if } a \leq 0 \end{cases}$$

- When g is convex, this is called a **sub-gradient**, and gradient descent is called **sub-gradient descent**.

Machine learning – ANN – Optimization

Let's try to learn the x-or function

```
import numpy as np

# Training set
x = np.array([[0,0], [0,1], [1,0], [1,1]]).T
d = np.array([0, +1, +1, 0]).T

# Initialization for a 2 layer feedforward network
b1 = np.random.rand(2, 1)
W1 = np.random.rand(2, 2)
b2 = np.random.rand(1, 1)
W2 = np.random.rand(1, 2)

# Activation functions and their derivatives
def g1(a): return a * (a > 0) # ReLU
def g1p(a): return 1 * (a > 0)
def g2(a): return a # Linear
def g2p(a): return 1
```

Machine learning – ANN – Optimization

Let's try to learn the x-or function

```
import numpy as np

# Training set
x = np.array([[0,0], [0,1], [1,0], [1,1]]).T
d = np.array([0, +1, +1, 0]).T

# Initialization for a 2 layer feedforward network
b1 = np.random.rand(2, 1)
W1 = np.random.rand(2, 2)
b2 = np.random.rand(1, 1)
W2 = np.random.rand(1, 2)

# Activation functions and their derivatives
def g1(a): return a * (a > 0) # ReLU
def g1p(a): return 1 * (a > 0)
def g2(a): return 1 / (1 + np.exp(-a)) # Logistic
def g2p(a): return g2(a) * (1 - g2(a))
```

Machine learning – ANN – Optimization

Let's try to learn the x-or function

```
gamma = .01 # step parameter (learning rate)
for t in range(0, 10000):
    # Forward phase
    a1      = W1.dot(x)
    h1      = g1(a1)
    a2      = W2.dot(h1)
    y       = g2(a2)
    # Error evaluation
    e       = y - d
    # Backward phase
    delta2 = g2p(a2) * e
    delta1 = g1p(a1) * W2.T.dot(delta2)
    # gradient update
    W2      = W2 - gamma * delta2.dot(h1.T)
    W1      = W1 - gamma * delta1.dot(x.T)
    -
    -
```

Machine learning – ANN – Optimization

Let's try to learn the x-or function

```
gamma = .01 # step parameter
for t in range(0, 10000):
    # Forward phase
    a1      = W1.dot(x) + b1
    h1      = g1(a1)
    a2      = W2.dot(h1) + b2
    y       = g2(a2)
    # Error evaluation
    e       = y - d
    # Backward phase
    delta2 = g2p(a2) * e
    delta1 = g1p(a1) * W2.T.dot(delta2)
    # gradient update
    W2      = W2 - gamma * delta2.dot(h1.T)
    W1      = W1 - gamma * delta1.dot(x.T)
    b2      = b2 - gamma * delta2.sum(axis=1, keepdims=True)
    b1      = b1 - gamma * delta1.sum(axis=1, keepdims=True)
```

Remark 1: dealing with the bias is similar

Machine learning – ANN – Optimization

Let's try to learn the x-or function

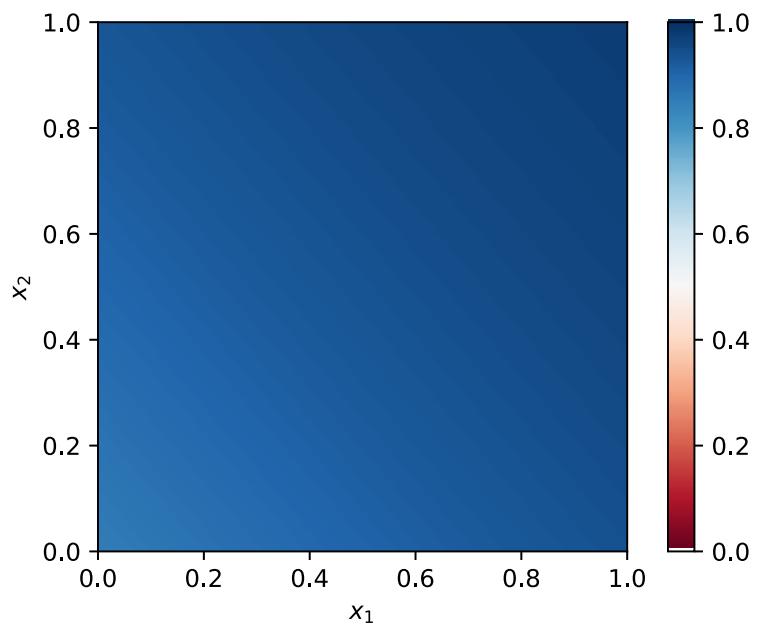
```
gamma = .01 # step parameter
for t in range(0, 10000):
    # Forward phase
    a1      = W1.dot(x) + b1
    h1      = g1(a1)
    a2      = W2.dot(h1) + b2
    y       = g2(a2)
    # Error evaluation
    e       = -d / y + (1 - d) / (1 - y)
    # Backward phase
    delta2 = g2p(a2) * e
    delta1 = g1p(a1) * W2.T.dot(delta2)
    # gradient update
    W2      = W2 - gamma * delta2.dot(h1.T)
    W1      = W1 - gamma * delta1.dot(x.T)
    b2      = b2 - gamma * delta2.sum(axis=1, keepdims=True)
    b1      = b1 - gamma * delta1.sum(axis=1, keepdims=True)
```

Remark 1: dealing with the bias is similar

Remark 2: using cross-entropy is simple too

Machine learning – ANN – Optimization

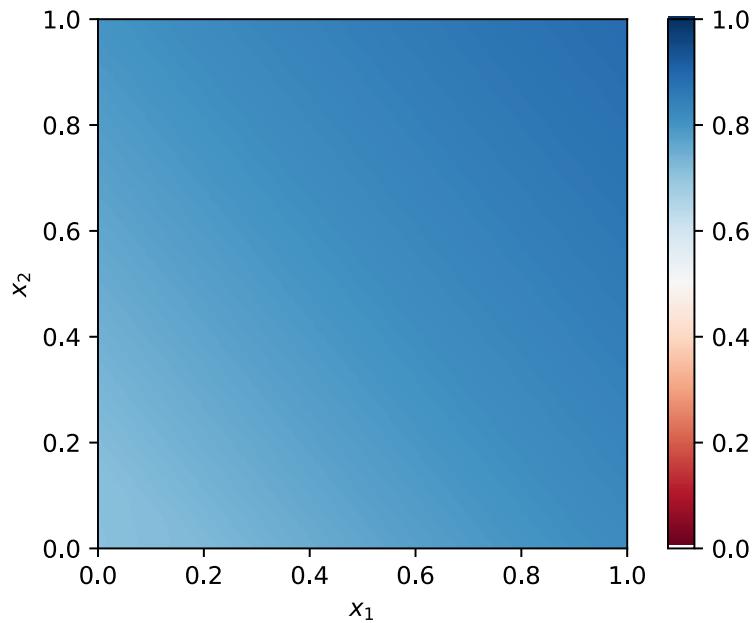
Let's see how it works



Iteration 1

Machine learning – ANN – Optimization

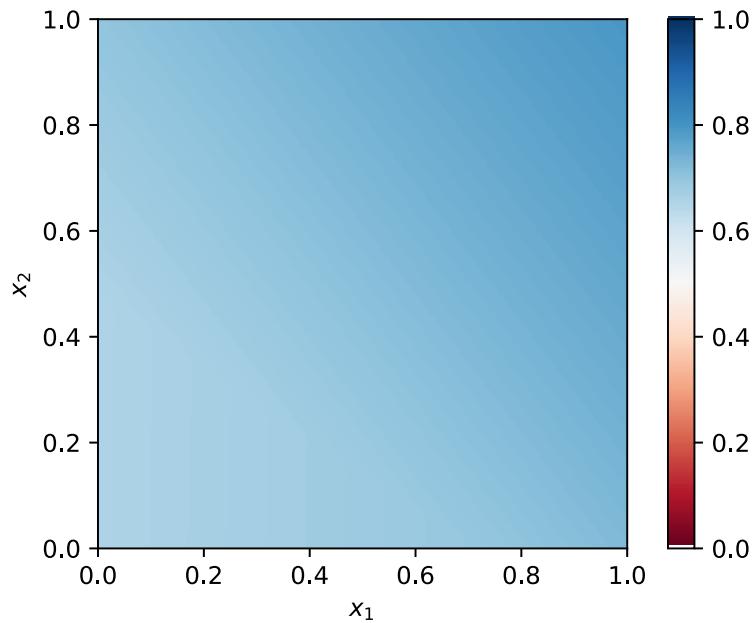
Let's see how it works



Iteration 11

Machine learning – ANN – Optimization

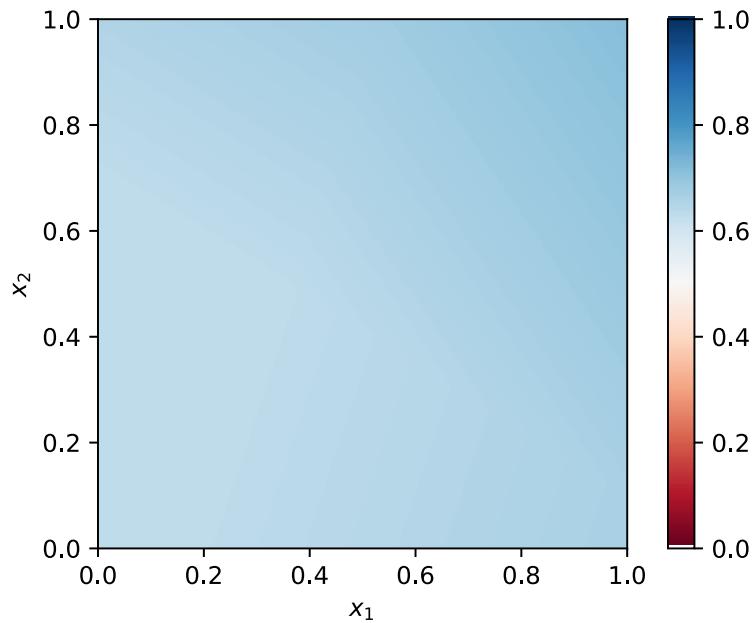
Let's see how it works



Iteration 21

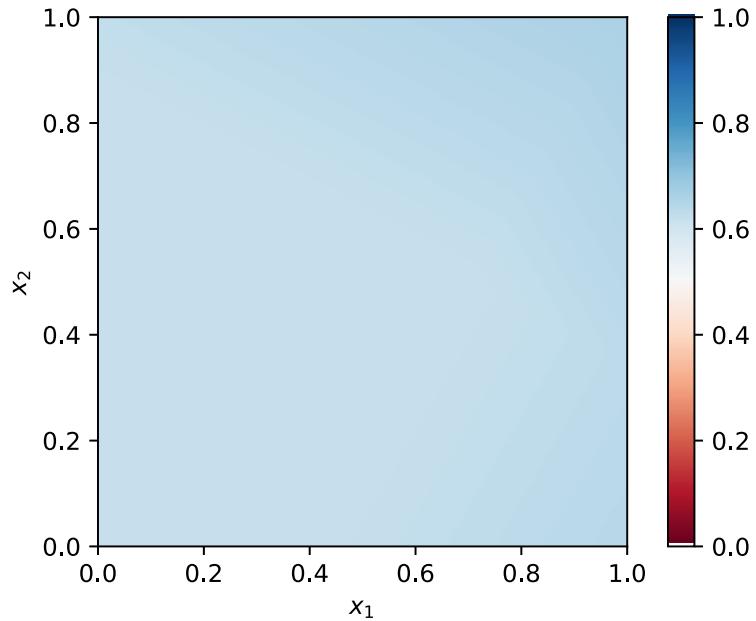
Machine learning – ANN – Optimization

Let's see how it works



Machine learning – ANN – Optimization

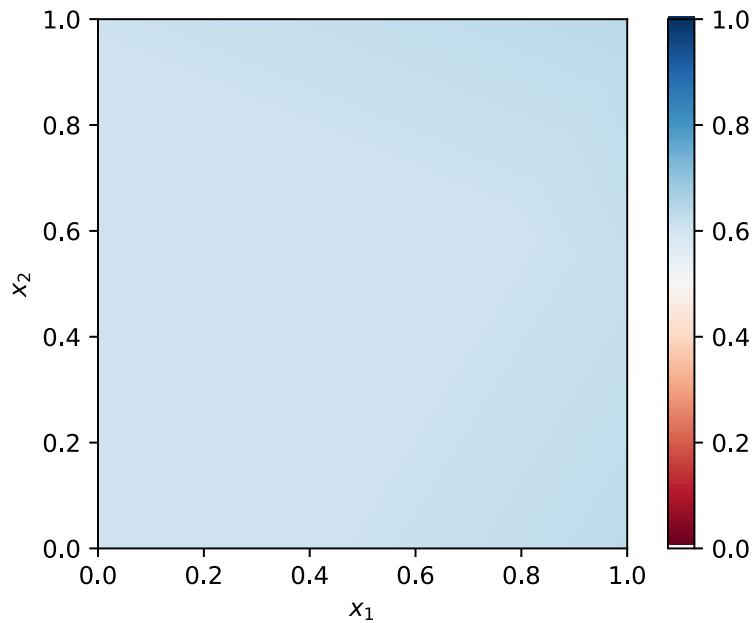
Let's see how it works



Iteration 41

Machine learning – ANN – Optimization

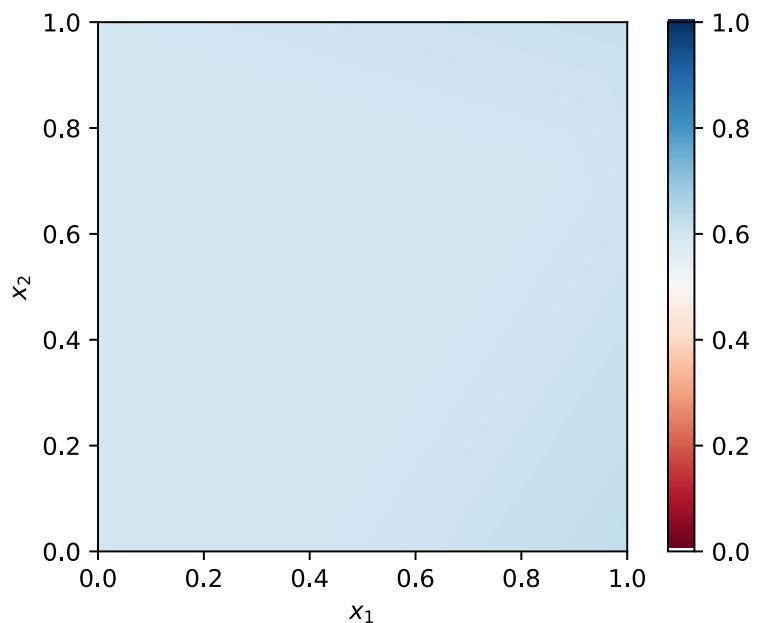
Let's see how it works



Iteration 51

Machine learning – ANN – Optimization

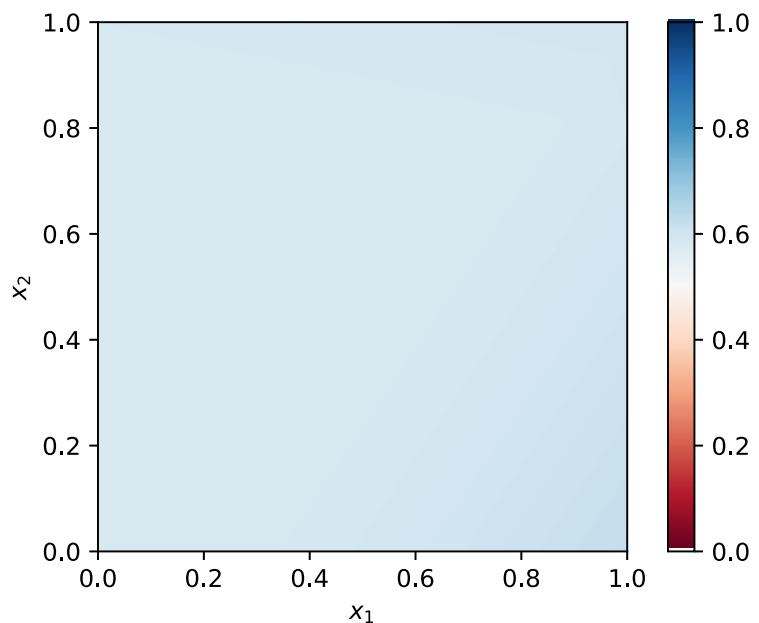
Let's see how it works



Iteration 61

Machine learning – ANN – Optimization

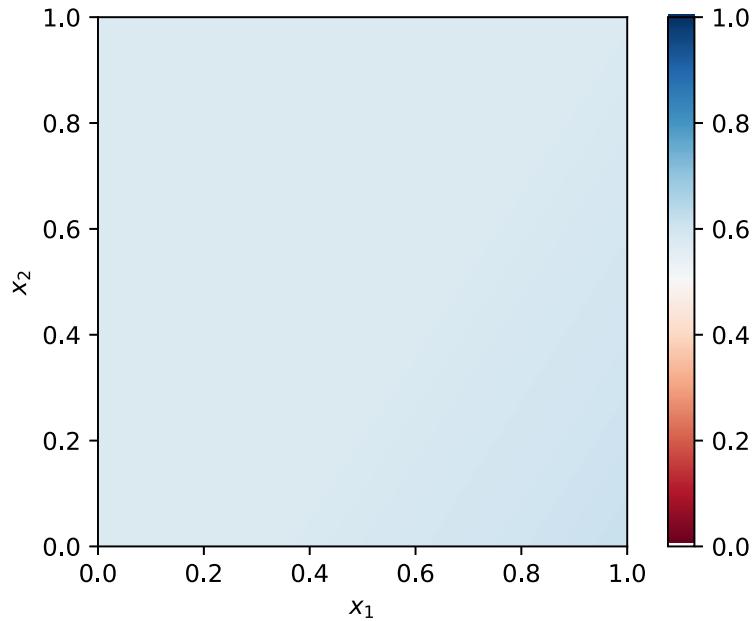
Let's see how it works



Iteration 71

Machine learning – ANN – Optimization

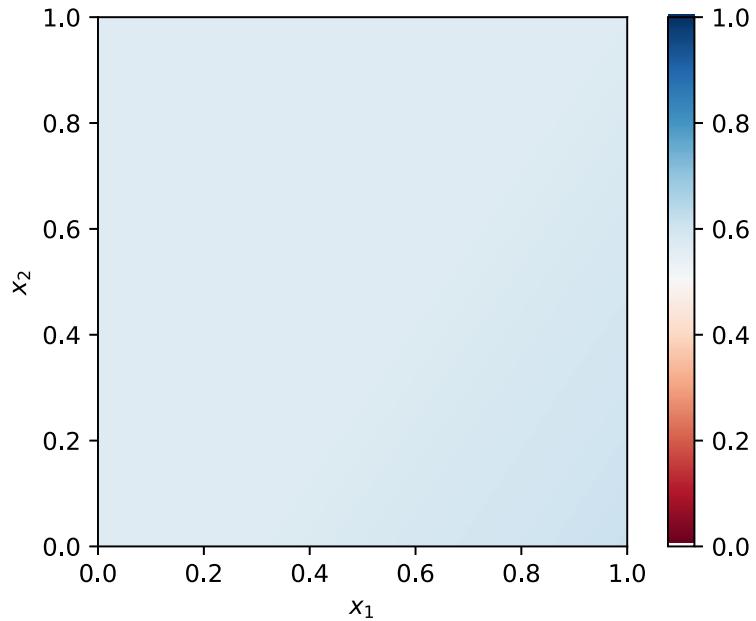
Let's see how it works



Iteration 81

Machine learning – ANN – Optimization

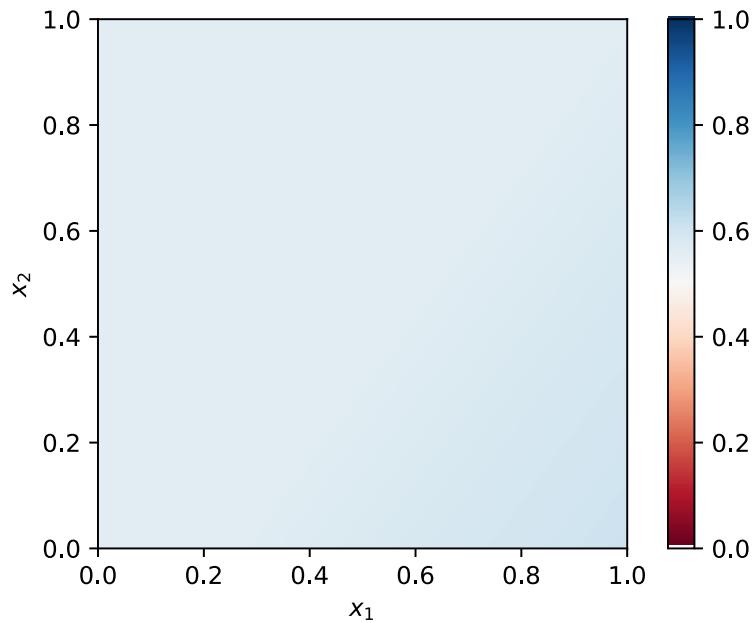
Let's see how it works



Iteration 91

Machine learning – ANN – Optimization

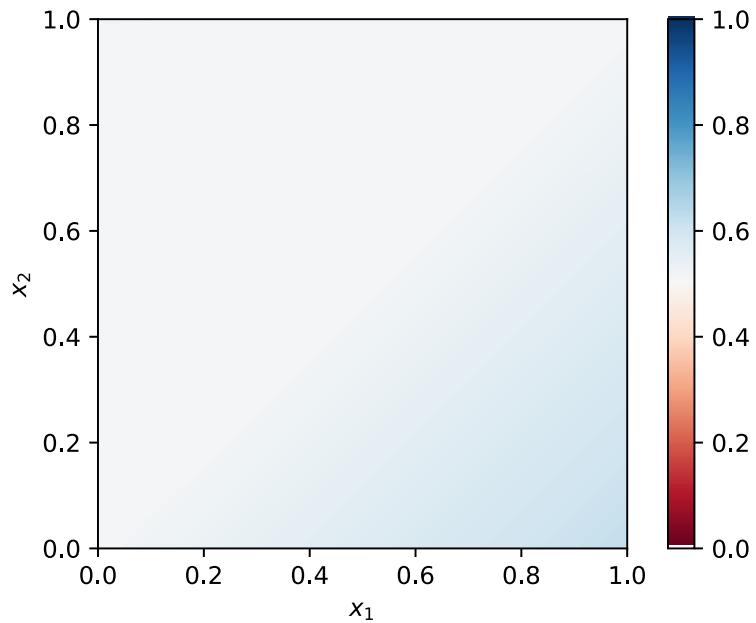
Let's see how it works



Iteration 101

Machine learning – ANN – Optimization

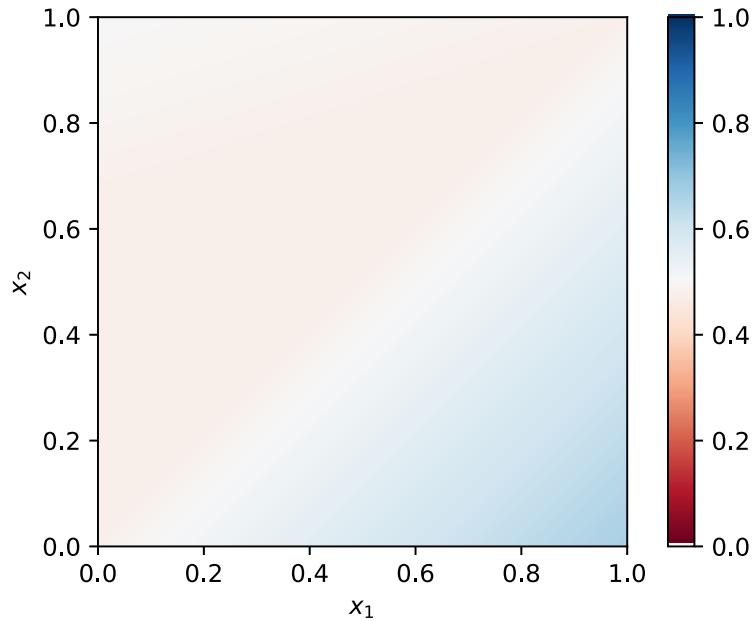
Let's see how it works



Iteration 201

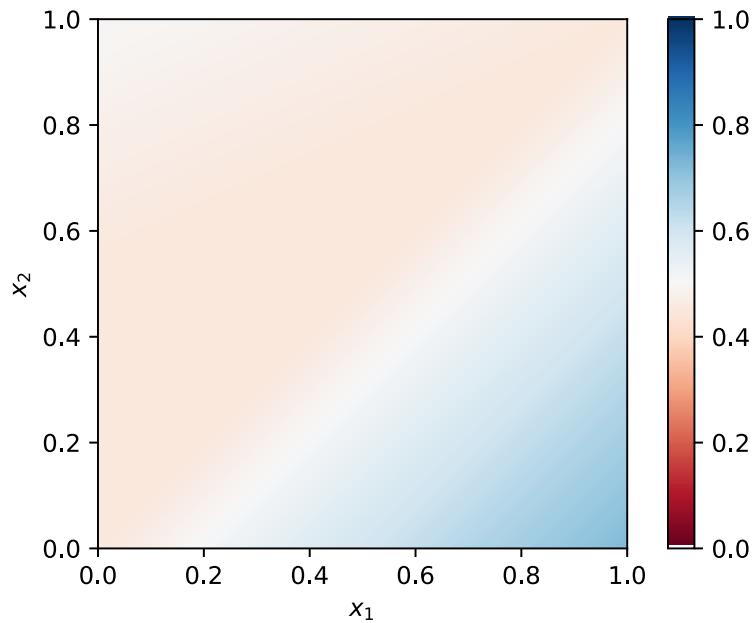
Machine learning – ANN – Optimization

Let's see how it works



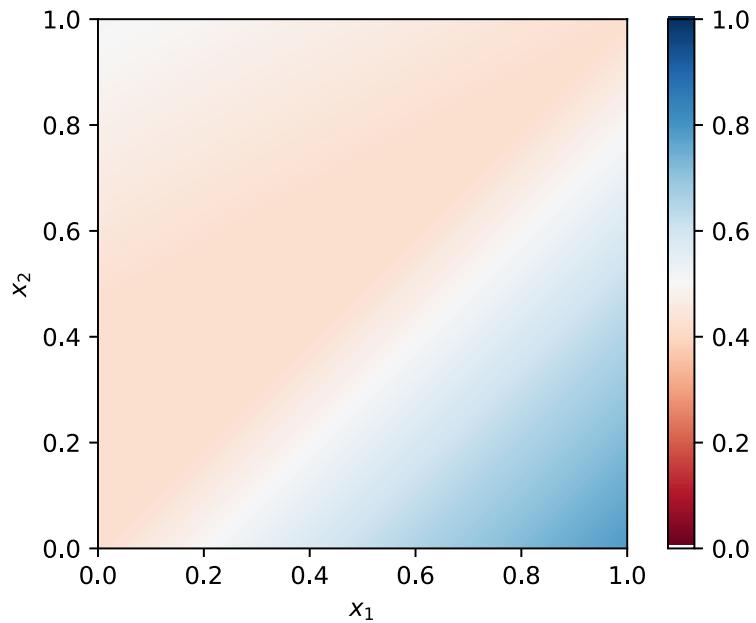
Machine learning – ANN – Optimization

Let's see how it works



Machine learning – ANN – Optimization

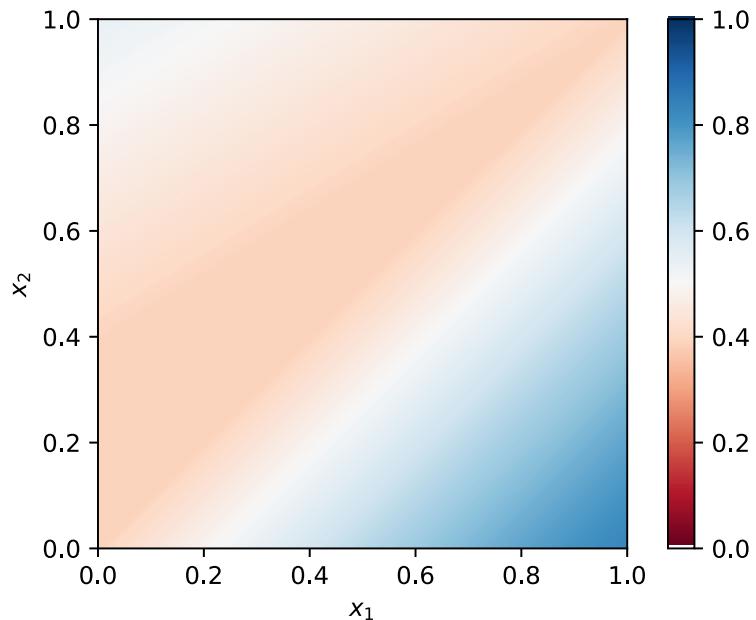
Let's see how it works



Iteration 501

Machine learning – ANN – Optimization

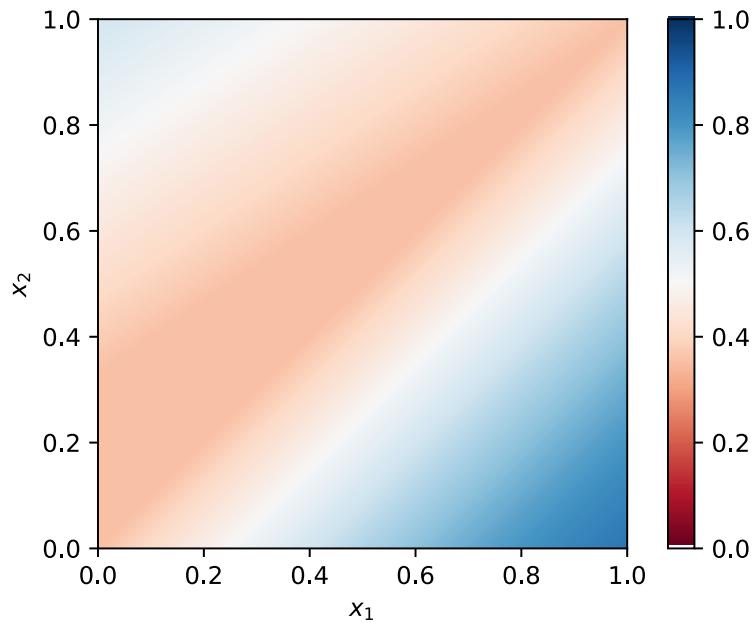
Let's see how it works



Iteration 601

Machine learning – ANN – Optimization

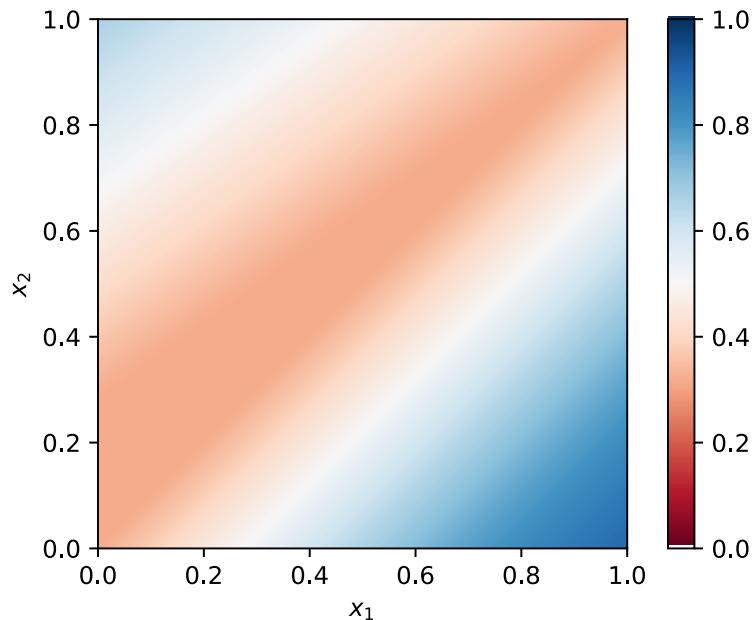
Let's see how it works



Iteration 701

Machine learning – ANN – Optimization

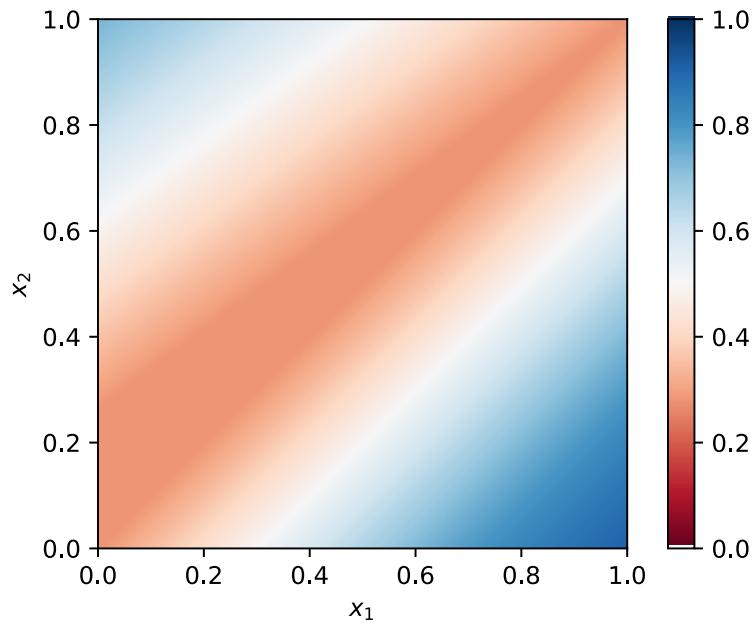
Let's see how it works



Iteration 801

Machine learning – ANN – Optimization

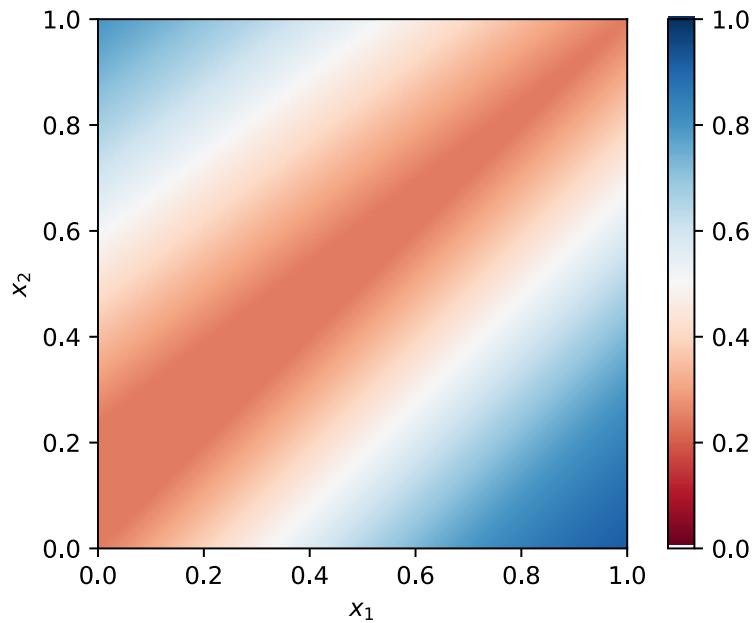
Let's see how it works



Iteration 901

Machine learning – ANN – Optimization

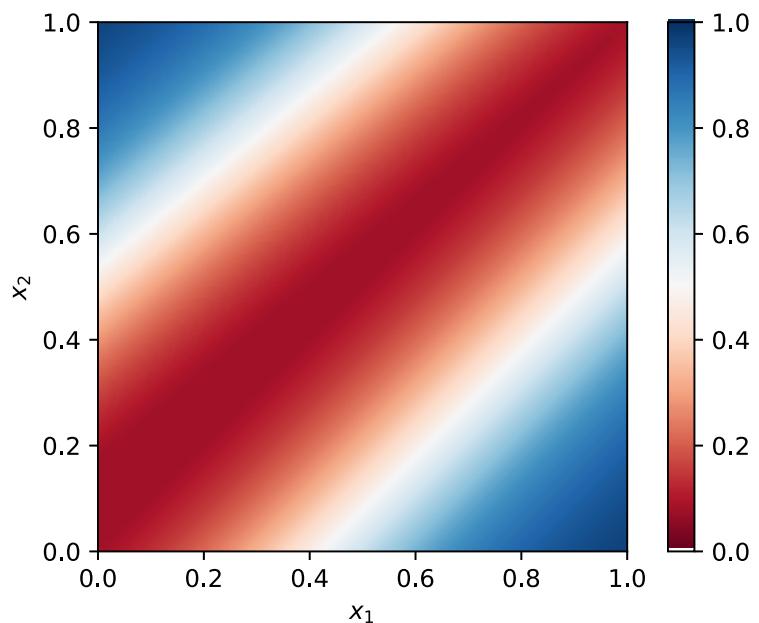
Let's see how it works



Iteration 1001

Machine learning – ANN – Optimization

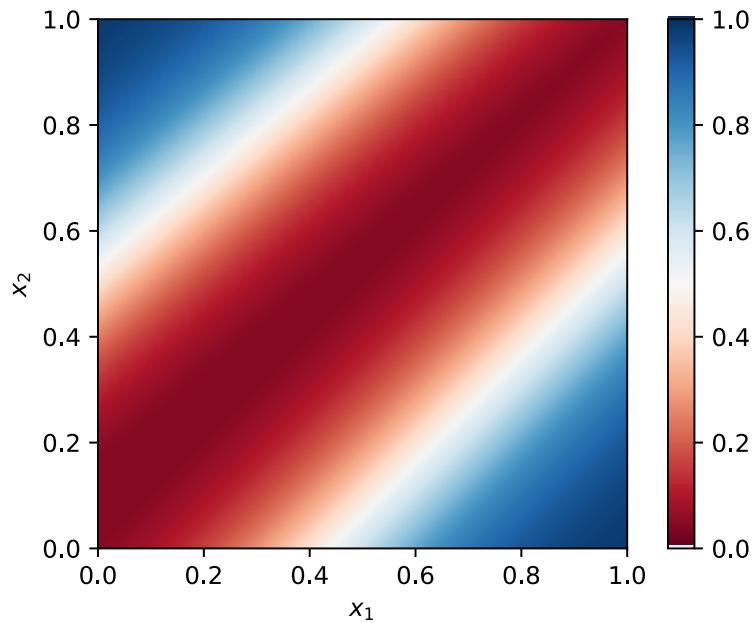
Let's see how it works



Iteration 2001

Machine learning – ANN – Optimization

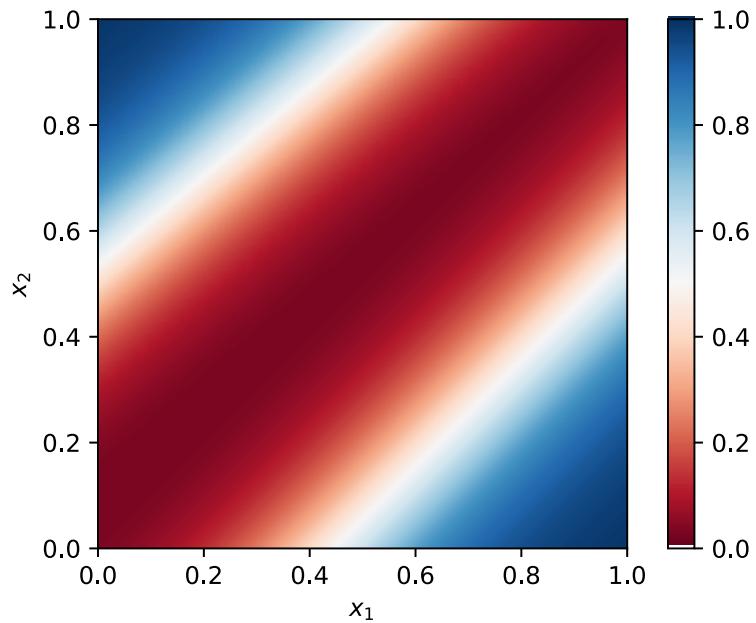
Let's see how it works



Iteration 3001

Machine learning – ANN – Optimization

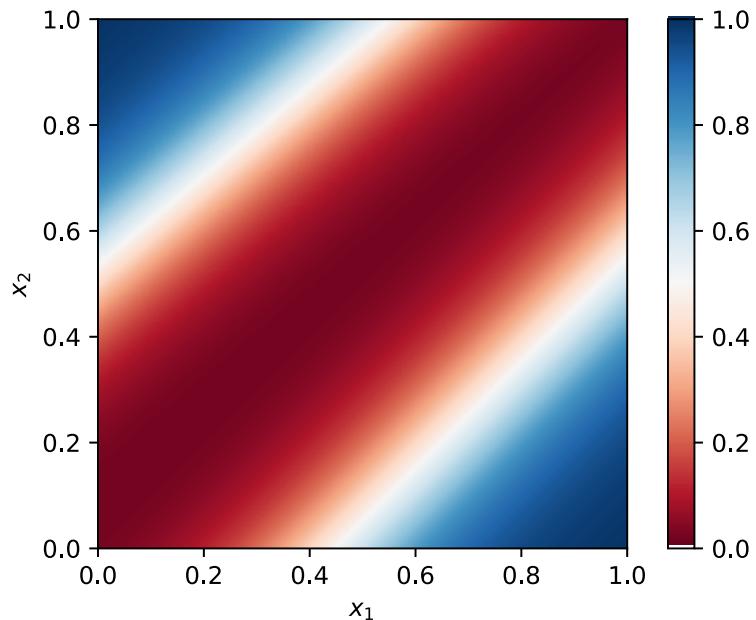
Let's see how it works



Iteration 4001

Machine learning – ANN – Optimization

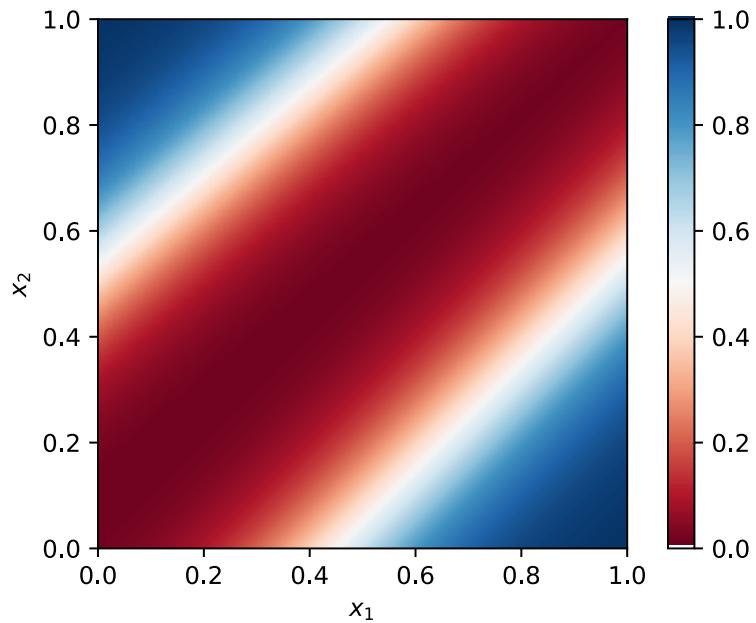
Let's see how it works



Iteration 5001

Machine learning – ANN – Optimization

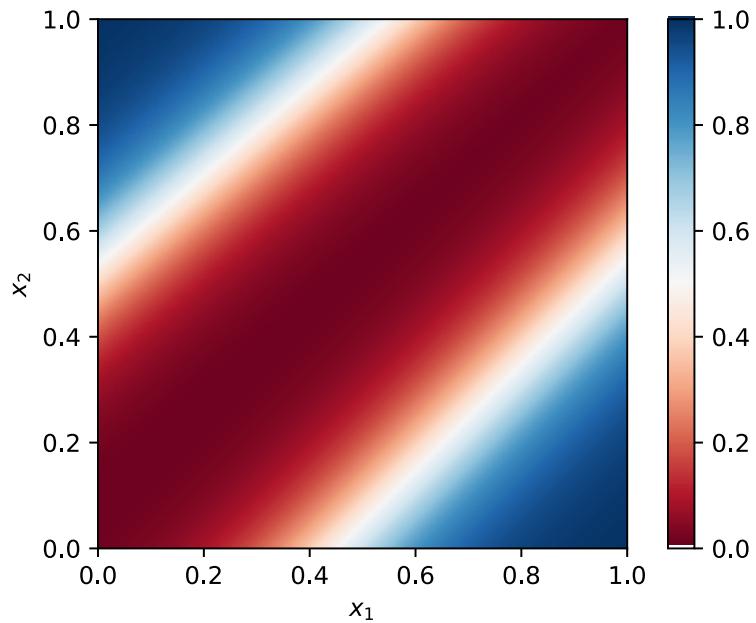
Let's see how it works



Iteration 6001

Machine learning – ANN – Optimization

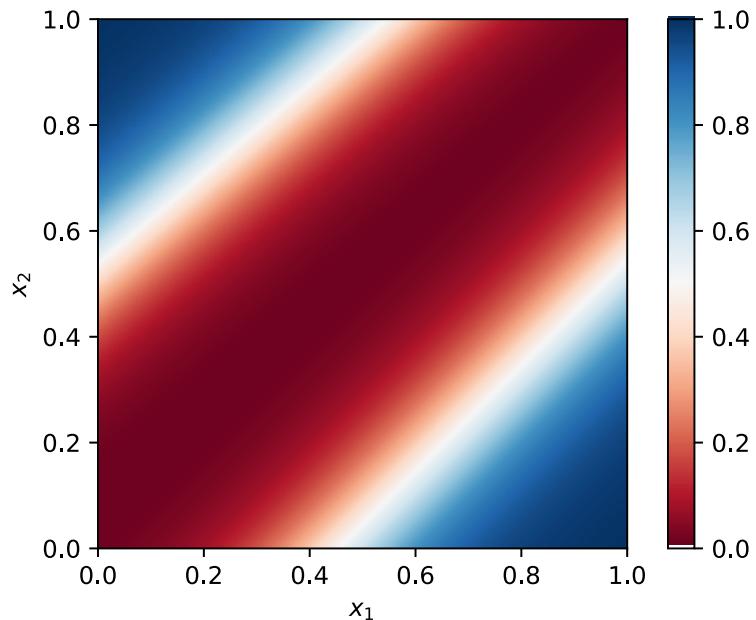
Let's see how it works



Iteration 7001

Machine learning – ANN – Optimization

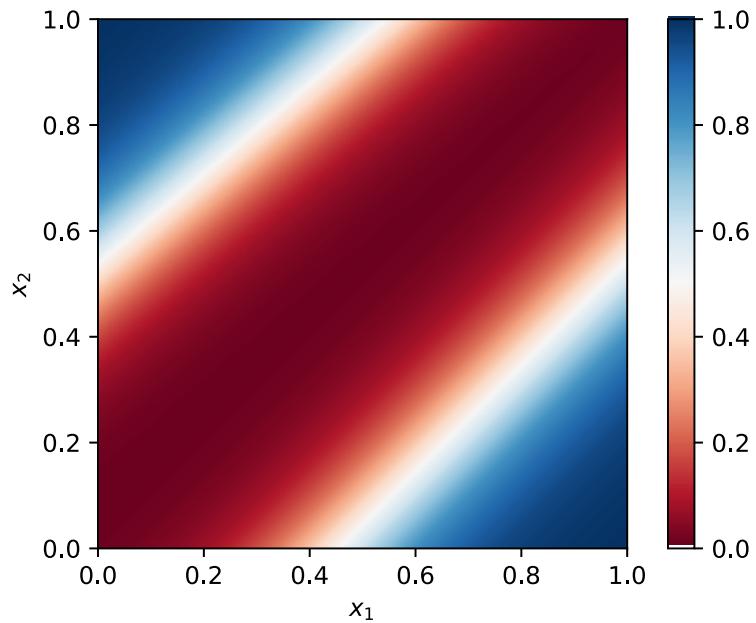
Let's see how it works



Iteration 8001

Machine learning – ANN – Optimization

Let's see how it works

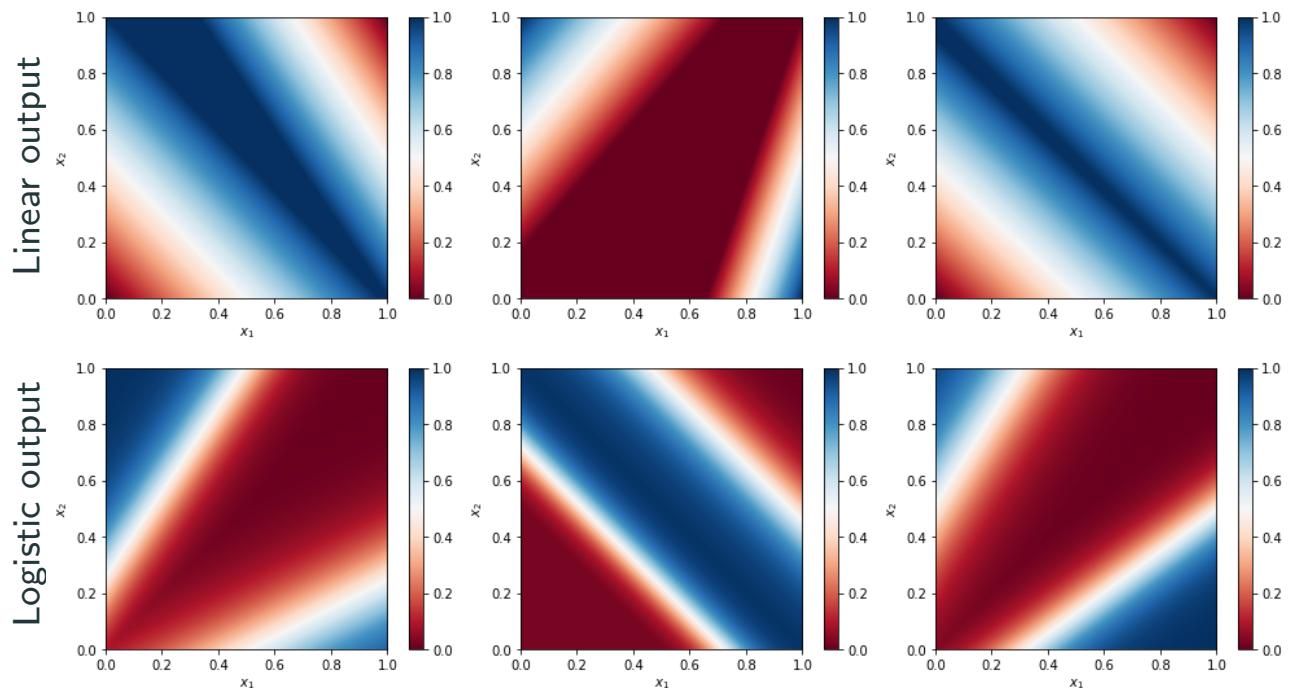


Iteration 9001

Machine learning – ANN – Optimization

Comparisons – different initializations & activations

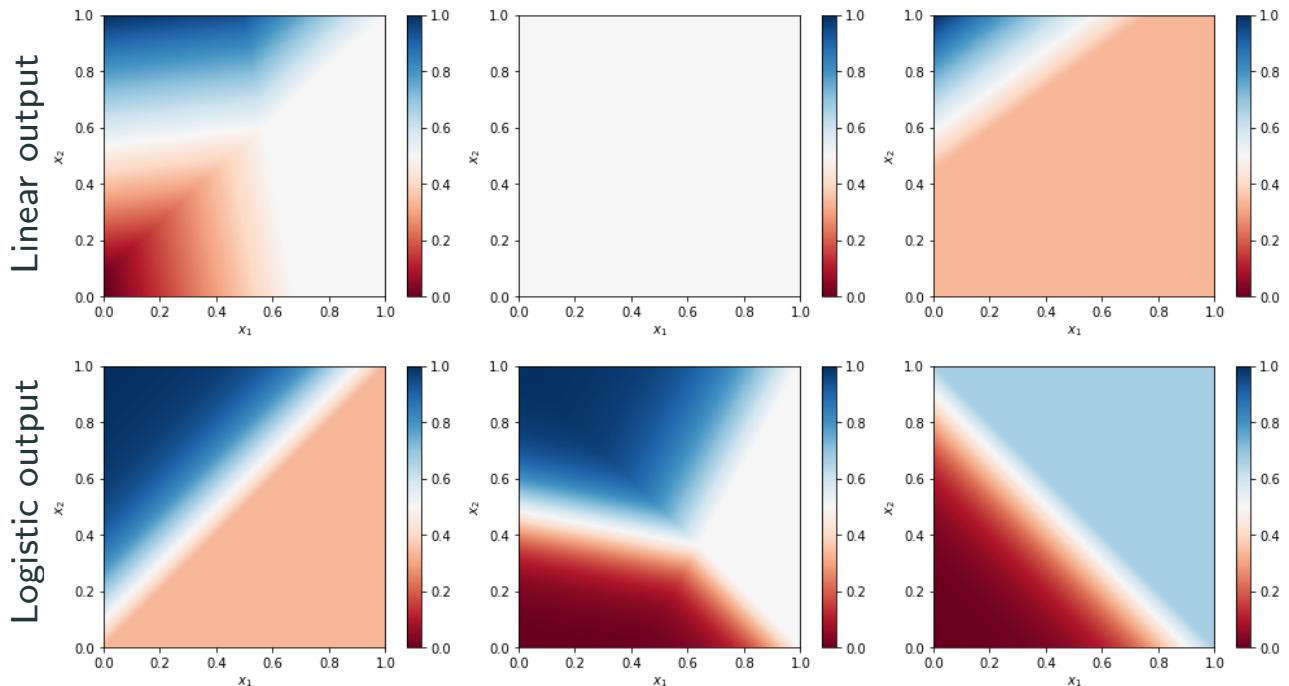
Success (Global minima)



Machine learning – ANN – Optimization

Comparisons – different initializations & activations

Failures (Local minima or saddle points)



Machine learning – ANN – Optimization

The 1990s view of ANN and back-propagation

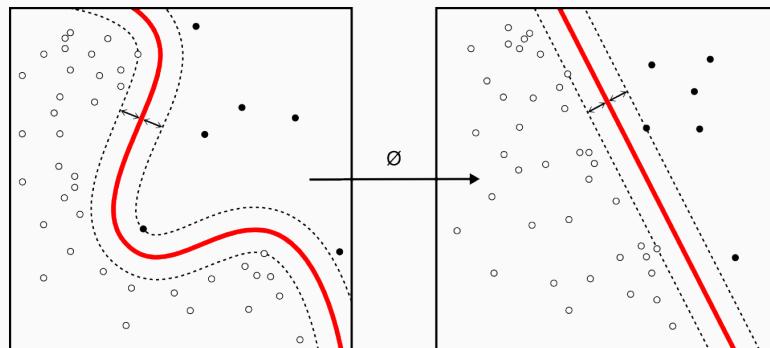
Advantages

- Universal approximators,
- May be very accurate.

Drawbacks

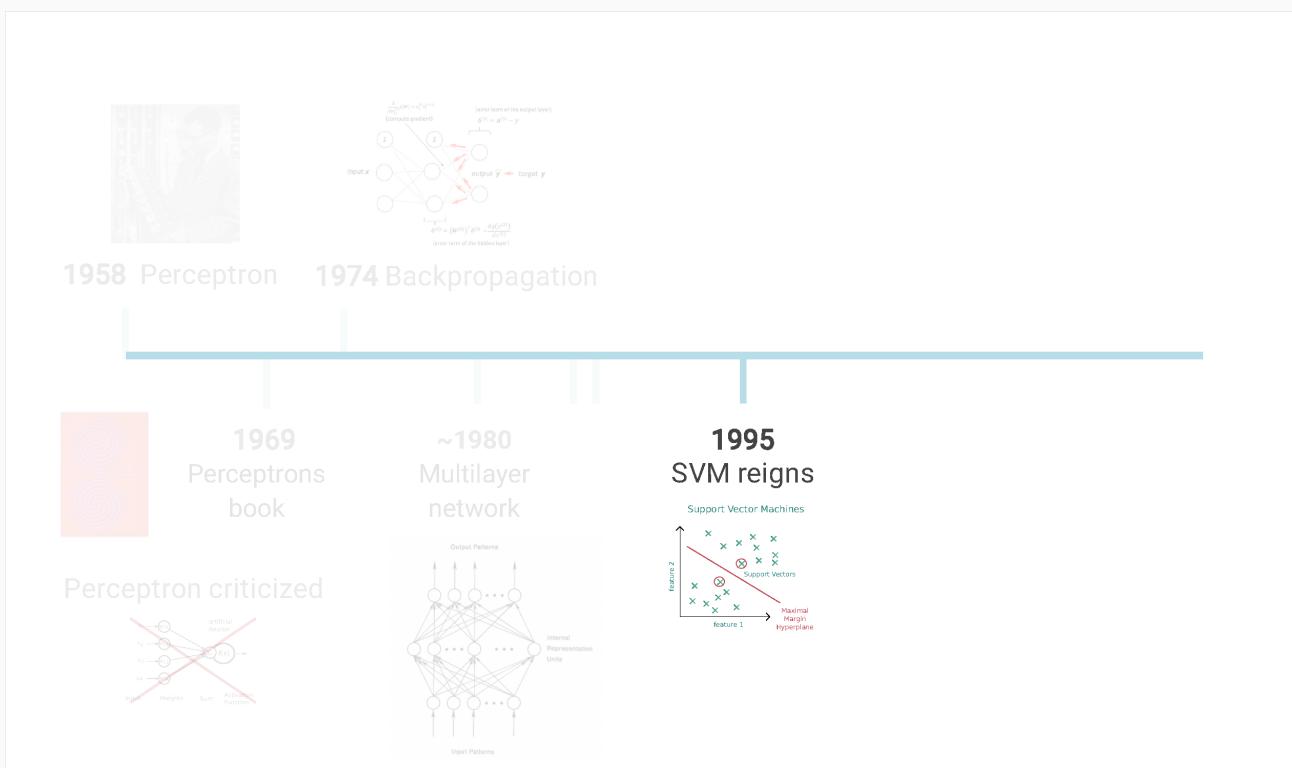
- Black box models (very difficult to interpret).
- The learning time does not scale well
 - it was very slow in networks with multiple hidden layers.
- It got stuck at local optima or saddle points
 - these can be nevertheless often surprisingly good.
- All global minima do not have the same quality (generalization error)
 - strong variations in prediction errors on different testing datasets.

Support Vector Machine



Machine learning – Support Vector Machine

Support Vector Machine



(Source: Lucas Masuch & Vincent Lepetit)

Machine learning – Support Vector Machine

Support Vector Machine

(Vapnik, 1995)

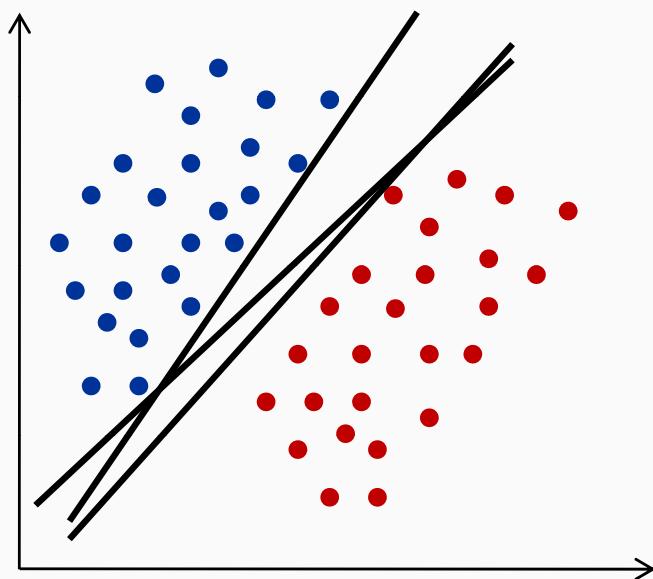
- Very successful methods in the mid-90's.
- Developed for binary classification.
- Clever type of perceptron.
- Based on two major ideas
 - ① large margin,
 - ② kernel trick.

Many NNs researchers switched to SVMs in the 1990s
because they (used to) work better.

Machine learning – Support Vector Machine

Shattering points with oriented hyperplanes

- **Goal:** Build hyperplanes that separate points in two classes,
- **Question:** Which is the best separating line?

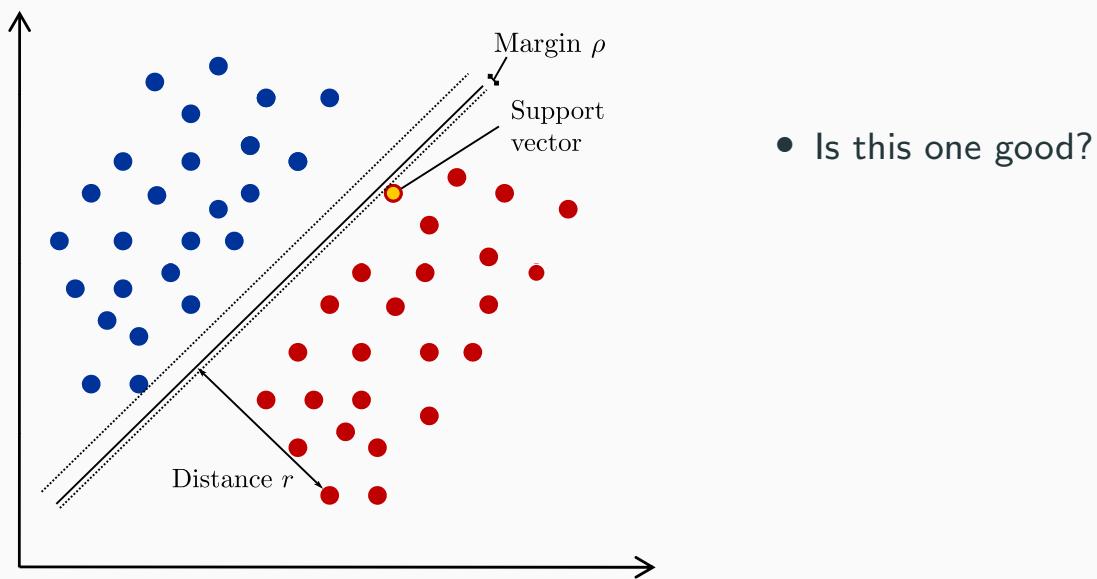


Remember, hyperplane: $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$

Machine learning – Support Vector Machine

Classification margin

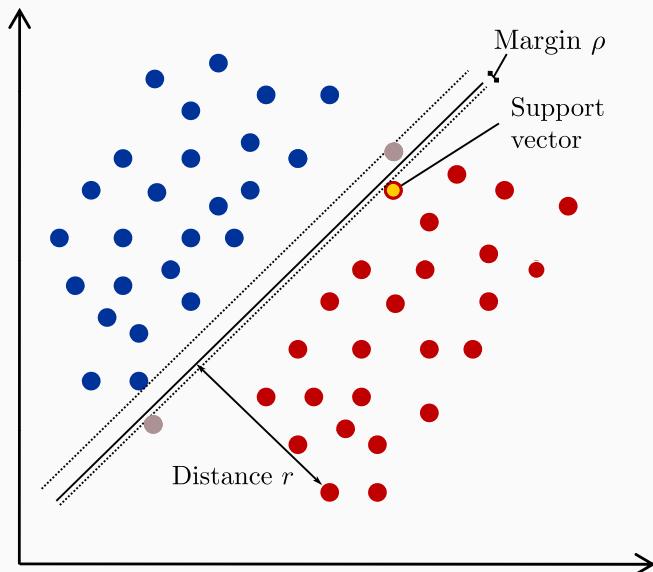
- Signed distance from an example to the separator: $r = \frac{\langle \mathbf{x}, \mathbf{w} \rangle + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are **support vectors**.
- **Margin** ρ is the distance between the separator and the support vector(s).



Machine learning – Support Vector Machine

Classification margin

- Signed distance from an example to the separator: $r = \frac{\langle \mathbf{x}, \mathbf{w} \rangle + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are **support vectors**.
- **Margin** ρ is the distance between the separator and the support vector(s).

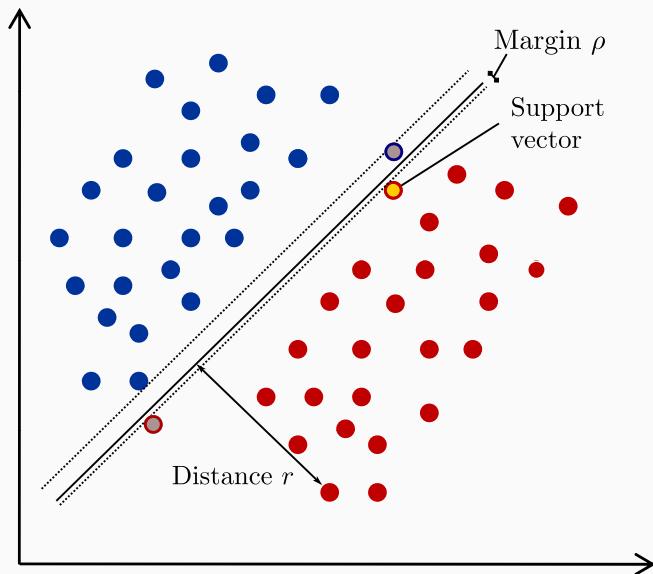


- Is this one good?
- Consider two testing samples, where are they assigned?

Machine learning – Support Vector Machine

Classification margin

- Signed distance from an example to the separator: $r = \frac{\langle \mathbf{x}, \mathbf{w} \rangle + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are **support vectors**.
- **Margin** ρ is the distance between the separator and the support vector(s).

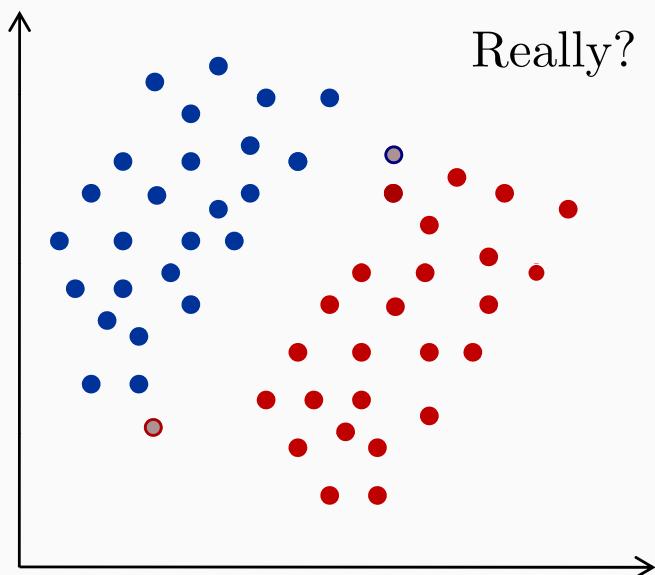


- Is this one good?
- Consider two testing samples, where are they assigned?

Machine learning – Support Vector Machine

Classification margin

- Signed distance from an example to the separator: $r = \frac{\langle \mathbf{x}, \mathbf{w} \rangle + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are **support vectors**.
- **Margin** ρ is the distance between the separator and the support vector(s).

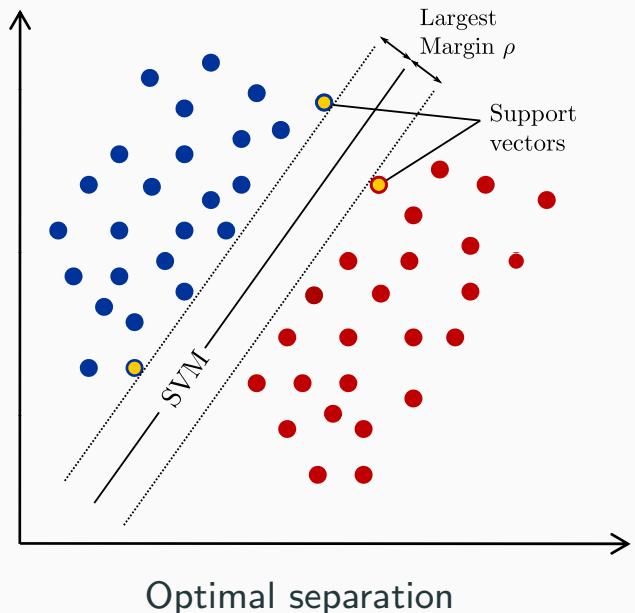
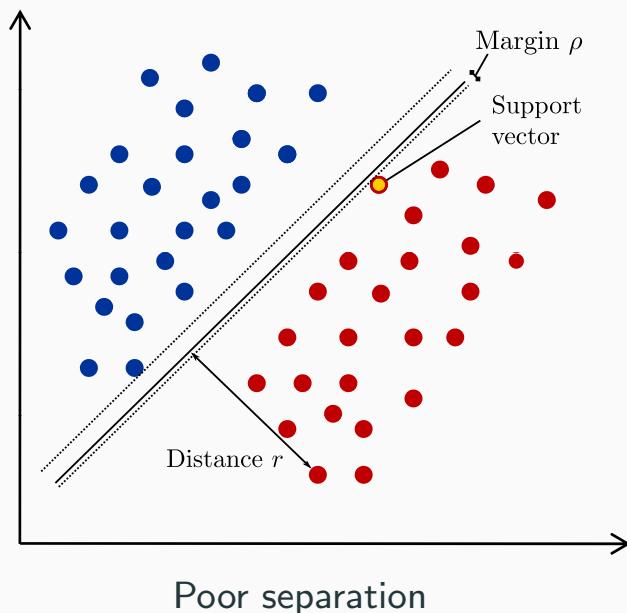


- Is this one good?
- Consider two testing samples, where are they assigned?
- This separator may have large generalization error.

Machine learning – Support Vector Machine

Largest margin = Support Vector Machine

- Maximizing the margin reduces generalization error (see, PAC learning).
- Then, there are necessary support vectors on both sides of the hyperplane.
- Only support vectors are important \Rightarrow other examples can be discarded.



Machine learning – Support Vector Machine

Training

- As for neural networks, the parameters w and b are obtained by optimizing a loss function (the margin).
- What is the formula for the margin?

- Training dataset: feature vectors x^i , targeted class $d^i \in \{-1, +1\}$
- Assume the training samples to be **linearly separable**,
i.e., there exist w and b such that

$$\begin{cases} \langle x^i, w \rangle + b \geq +1 & \text{if } d^i = +1 \\ \langle x^i, w \rangle + b \leq -1 & \text{if } d^i = -1 \end{cases} \quad \underline{d^i(\langle x^i, w \rangle + b) \geq 1}$$

Then: $|\langle x^i, w \rangle + b| \geq 1$

Machine learning – Support Vector Machine

Training

- For support vectors, the inequality can be forced to be an equality

$$|\langle \mathbf{x}^i, \mathbf{w} \rangle + b| = 1$$

- Then, the distance between any support vector and the hyperplane is

$$|r| = \frac{|\langle \mathbf{x}^i, \mathbf{w} \rangle + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

- So, the margin is (by definition)

$$\rho = |r| = \frac{1}{\|\mathbf{w}\|}$$

- Maximizing the margin is then equivalent to minimizing $\|\mathbf{w}\|^2$, providing the hyperplane (\mathbf{w}, b) separates the data.

Machine learning – Support Vector Machine

Training

- Therefore, the problem can be recast as

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{subject to} \quad d^i(\langle \mathbf{x}^i, \mathbf{w} \rangle + b) \geq 1, \quad \text{for all } i.$$

⇒ Quadratic (convex) optimization problem subject to linear constraints,

⇒ No local minima! Only a single global one.

Equivalent formulation:

$$\min_{\mathbf{w}, b} E(\mathbf{w}, b)$$

$$\text{where } E(\mathbf{w}, b) = \begin{cases} \frac{1}{2} \|\mathbf{w}\|^2 & \text{if } d^i(\langle \mathbf{x}^i, \mathbf{w} \rangle + b) \geq 1, \quad \text{for all } i \\ +\infty & \text{otherwise} \end{cases}$$

Machine learning – Support Vector Machine

Training

$$\min_{\mathbf{w}, b} E(\mathbf{w}, b)$$

where $E(\mathbf{w}, b) = \begin{cases} \frac{1}{2} \|\mathbf{w}\|^2 & \text{if } d^i(\langle \mathbf{x}^i, \mathbf{w} \rangle + b) \geq 1, \quad \text{for all } i \\ +\infty & \text{otherwise} \end{cases}$

- We can use the techniques of **Lagrange multipliers**:

- ① Introduce a Lagrange multiplier $\alpha_i \geq 0$ for each constraint.
- ② Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots)^T$ be the vector of Lagrange multipliers.
- ③ Define the (primal) function as

$$\mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_i \underbrace{\alpha^i (d^i(\langle \mathbf{x}^i, \mathbf{w} \rangle + b) - 1)}_{\text{inequality constraint}}$$

- ④ Solve the saddle point problem

$$E(\mathbf{w}, b) = \max_{\boldsymbol{\alpha}} \mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha}) \Rightarrow \min_{\mathbf{w}, b} E(\mathbf{w}, b) = \min_{\mathbf{w}, b} \max_{\boldsymbol{\alpha}} \mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha})$$

Machine learning – Support Vector Machine

Training

- When the original objective-function is convex (and only then), we can interchange the minimization and maximization

$$\min_{\mathbf{w}, b} \max_{\boldsymbol{\alpha}} \mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}} \min_{\mathbf{w}, b} \mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha})$$

- The following is called **dual problem**

$$\mathcal{L}_D(\boldsymbol{\alpha}) = \min_{\mathbf{w}, b} \left\{ \mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \underbrace{\|\mathbf{w}\|^2}_{\langle \mathbf{w}, \mathbf{w} \rangle} - \sum_i \alpha^i (d^i (\langle \mathbf{x}^i, \mathbf{w} \rangle + b) - 1) \right\}$$

- For a fixed $\boldsymbol{\alpha}$, the minimum is achieved by simultaneously canceling the gradients with respect to \mathbf{w} and b

$$\nabla_{\mathbf{w}} \mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \Rightarrow \mathbf{w} - \sum_i \alpha^i d^i \mathbf{x}^i = 0 \Rightarrow \mathbf{w} = \sum_i \alpha^i d^i \mathbf{x}^i$$

$$\text{and } \nabla_b \mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \Rightarrow \sum_i \alpha^i d^i = 0$$

Machine learning – Support Vector Machine

Training

The dual is obtained by plugging these equations in $\mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha})$:

$$\begin{aligned}\mathcal{L}_D(\boldsymbol{\alpha}) &= \frac{1}{2} \underbrace{\|\mathbf{w}\|^2}_{\langle \mathbf{w}, \mathbf{w} \rangle} - \sum_i \alpha^i (d^i (\langle \mathbf{x}^i, \mathbf{w} \rangle + b) - 1) \\ &= \frac{1}{2} \left\langle \sum_i \alpha^i d^i \mathbf{x}^i, \sum_j \alpha^j d^j \mathbf{x}^j \right\rangle - \sum_i \alpha^i d^i (\langle \mathbf{x}^i, \sum_j \alpha^j d^j \mathbf{x}^j \rangle + b) + \sum_i \alpha^i \\ &= \frac{1}{2} \sum_i \sum_j \alpha^i \alpha^j d^i d^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle - \sum_i \sum_j \alpha^i \alpha^j d^i d^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle + b \underbrace{\sum_i \alpha^i d^i}_{=0} + \sum_i \alpha^i \\ &= \sum_i \alpha^i - \frac{1}{2} \sum_i \sum_j \alpha^i \alpha^j d^i d^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle\end{aligned}$$

Machine learning – Support Vector Machine

SVM training algorithm

- ① Maximize the dual (e.g., using coordinate projected gradient descent):

$$\max_{\alpha} \left\{ \mathcal{L}_D(\alpha) = \sum_i \alpha^i - \frac{1}{2} \sum_i \sum_j \alpha^i \alpha^j d^i d^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle \right\}$$

subject to $\alpha^i \geq 0$ and $\sum_i \alpha^i d^i = 0$, for all i .

- ② Each non-zero α^i indicates that corresponding \mathbf{x}^i is a support vector.
③ Deduce the parameters from these support vectors (s.v.)

$$\mathbf{w} = \sum_{i \text{ (s.v.)}} \alpha^i d^i \mathbf{x}^i \quad \text{and} \quad b = d^i - \sum_{j \text{ (s.v.)}} \alpha^j d^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle \quad \text{for any s.v. } i$$

- The dual is in practice more efficient to solve than the original problem, and it **allows identifying the support vectors**.
- It only depends on the dot products $\langle \mathbf{x}^i, \mathbf{x}^j \rangle$ between all training points.

Machine learning – Support Vector Machine

SVM classification algorithm

- Given the parameters of the hyperplane \mathbf{w} and b ,
the SVM classify a new sample \mathbf{x} as

$$y = \langle \mathbf{w}, \mathbf{x} \rangle + b \leq 0$$

(same as for the perceptron).

- But (very important), this can be reformulated as

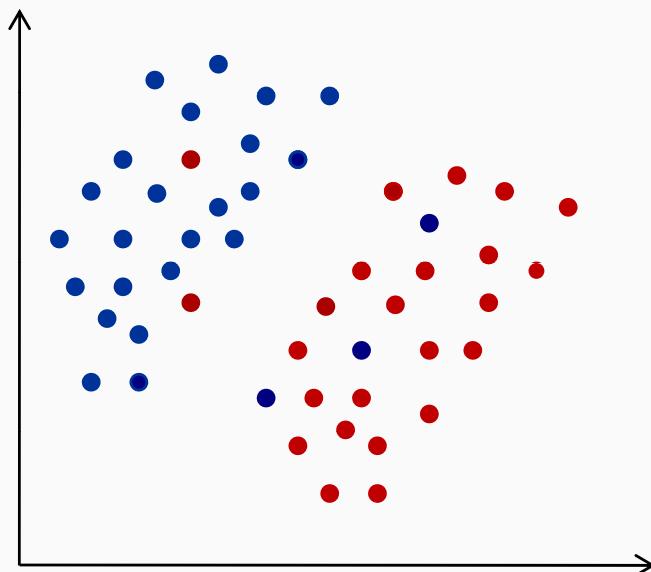
$$y = \sum_{i \text{ (s.v.)}} \alpha^i d^i \langle \mathbf{x}, \mathbf{x}^i \rangle + b$$

which only depends on the dot products $\langle \mathbf{x}, \mathbf{x}^i \rangle$ between \mathbf{x} and the support vectors – we will return to this later.

Machine learning – Support Vector Machine

Non-separable case

What if the training set is not linearly separable?

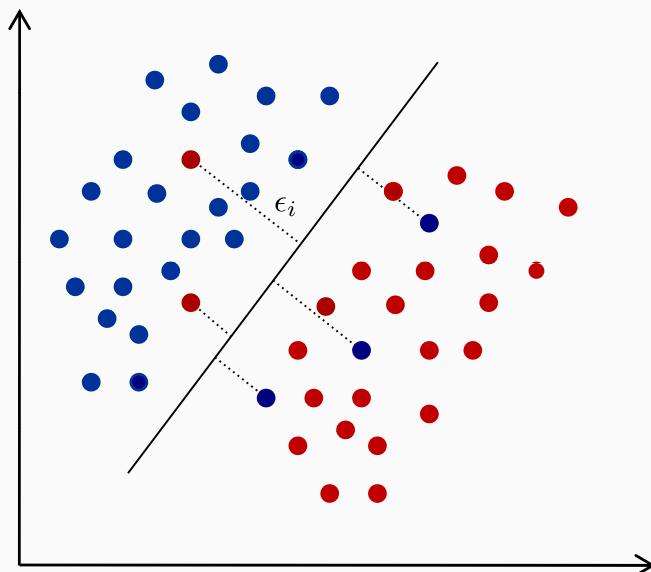


The optimization problem does not admit solutions.

Machine learning – Support Vector Machine

Non-separable case

What if the training set is not linearly separable?



The optimization problem does not admit solutions.

Solution: allows the system to make errors ε_i .

Machine learning – Support Vector Machine

Non-separable case – Soft-margin SVM

- Just relax the constraints by permitting errors to some extent

$$\min_{\mathbf{w}, b, \varepsilon} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \varepsilon_i$$

subject to $d^i(\langle \mathbf{x}^i, \mathbf{w} \rangle + b) \geq 1 - \varepsilon_i$ and $\varepsilon_i \geq 0$, for all i .

- Quantities ε_i are called slack variables.
- The parameter $C > 0$ is chosen by the user and controls overfitting.
- The dual problem becomes

$$\max_{\boldsymbol{\alpha}} \left\{ \mathcal{L}_D(\boldsymbol{\alpha}) = \sum_i \alpha^i - \frac{1}{2} \sum_i \sum_j \alpha^i \alpha^j d^i d^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle \right\}$$

subject to $0 \leq \alpha_i \leq C$ and $\sum_i \alpha^i d^i = 0$, for all i .

and does not depend on the slack variables ε_i .

Machine learning – Support Vector Machine

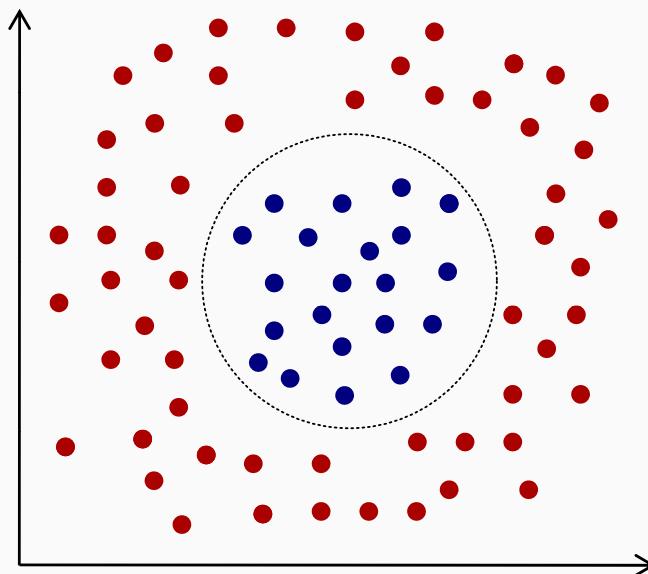
Linear SVM – Overview

- SVM finds the **separating hyperplane** maximizing the margin.
- Soft-margin SVM: trade-off between large margin and errors.
- These optimal hyperplanes are only defined in terms of **support vectors**.
- Lagrangian formulation allows identifying the support vectors with **non-zero Lagrange multipliers** α^i .
- The training and classification algorithms both depend only on **dot products between data points**.

So far, the SVM is a linear separator (the best in some sense).
But, it cannot solve non-linear problems (such as xor),
as done by multi-layer neural networks.

Machine learning – Support Vector Machine

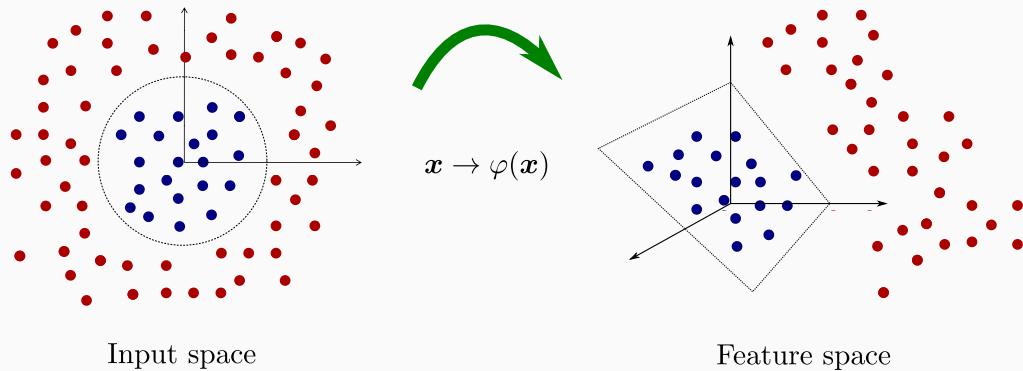
Non-linear SVM – Overview



What happens if the separator is non-linear?

Machine learning – Support Vector Machine

Non-linear SVM – Map to higher dimensions



$$\text{Here: } (x_1, x_2) \mapsto (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

The **input space** (original feature space) can always be mapped to some **higher-dimensional feature space** where the training data set is **linearly separable**, via some (non-linear) transformation $x \rightarrow \varphi(x)$.

Machine learning – Support Vector Machine

Non-linear SVM – Map to higher dimensions

Replace all occurrences of \mathbf{x} by $\varphi(\mathbf{x})$, for the training step

$$\mathcal{L}_D(\boldsymbol{\alpha}) = \sum_i \alpha^i - \frac{1}{2} \sum_i \sum_j \alpha^i \alpha^j d^i d^j \langle \varphi(\mathbf{x}^i), \varphi(\mathbf{x}^j) \rangle$$

and the classification step

$$y = \sum_{i \text{ (s.v.)}} \alpha^i d^i \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}^i) \rangle + b$$

- May require a feature space of really high (even infinite) dimension.
- In this case, manipulating $\varphi(\mathbf{x})$ might be tough, impossible, or lead to intense computation: complexity depends on the feature space dimension.

Machine learning – Support Vector Machine

Non-linear SVM – Kernel trick

- SVMs do not care about feature vectors $\varphi(\mathbf{x})$.
- They rely only on dot products between them. Define

$$K(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle$$

- K is called **kernel function**: dot product in a higher dimensional space.
- Even if $\varphi(\mathbf{x})$ is tough to manipulate, $K(\mathbf{x}, \mathbf{x}')$ can be rather simple.

- **Mercer's theorem:** K continuous and symmetric positive semi-definite (i.e., the matrix $\mathbf{K} = (K(\mathbf{x}^i, \mathbf{x}^j))_{i,j}$ is symmetric positive semi-definite for all finite sequences $\mathbf{x}^1, \dots, \mathbf{x}^n$) then there exists a mapping φ such that

$$K(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle$$

- We do not even need to know φ , pick a continuous symmetric positive semi-definite kernel K and use it instead of dot products.

Machine learning – Support Vector Machine

Non-linear SVM – Kernel trick

- Replace all occurrences of $\langle \mathbf{x}, \mathbf{x}' \rangle$ by $K(\mathbf{x}, \mathbf{x}')$, for the training step

$$\mathcal{L}_D(\boldsymbol{\alpha}) = \sum_i \alpha^i - \frac{1}{2} \sum_i \sum_j \alpha^i \alpha^j d^i d^j K(\mathbf{x}^i, \mathbf{x}^j)$$

and the classification step

$$y = \sum_{i \text{ (s.v.)}} \alpha^i d^i K(\mathbf{x}, \mathbf{x}^i) + b$$

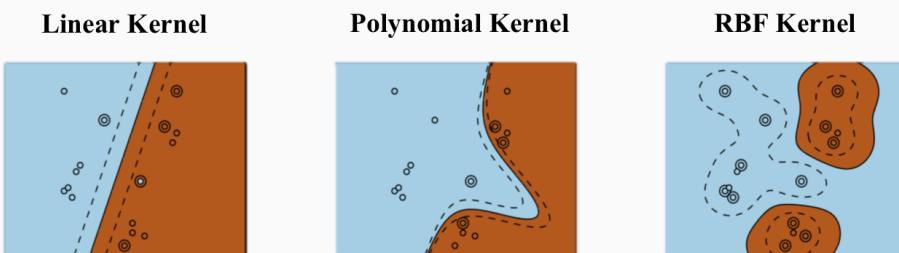
- Python implementation available in `scikit-learn`.

Complexity depends on the input space dimension.

Machine learning – Support Vector Machine

Non-linear SVM – Standard kernels

- Linear: $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$
- Polynomial: $K(\mathbf{x}, \mathbf{x}') = (\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + \beta)^p$
- Gaussian: $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$
→ Radial basis function (RBF) network.
- Sigmoid: $K(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + \beta)$



In practice: K is usually chosen by trial and error.

A linear SVM is an optimal perceptron but what is a non-linear SVM?

Machine learning – Support Vector Machine

Non-linear SVMs are shallow ANNs

Consider: $K(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + \beta)$

$$\text{We get: } y = \sum_{i \text{ (s.v.)}} \alpha^i d^i K(\mathbf{x}, \mathbf{x}^i) + b \quad \leftarrow \text{SVM}$$

$$y = \sum_{i \text{ (s.v.)}} \alpha^i d^i \tanh(\gamma \langle \mathbf{x}, \mathbf{x}^i \rangle + \beta) + b$$

$$y = \langle \mathbf{w}_2, g(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \rangle + b_2 \quad \leftarrow \text{Shallow ANN}$$

$$\text{where } \mathbf{W}_1 = \gamma \underbrace{\begin{pmatrix} \mathbf{x}^1 & \mathbf{x}^2 & \dots \end{pmatrix}}_{\text{support vectors}}^T, \quad \mathbf{b}_1 = \beta \mathbf{1},$$

$$\mathbf{w}_2 = \underbrace{\begin{pmatrix} \alpha^1 d^1 & \alpha^2 d^2 & \dots \end{pmatrix}}_{\text{Lagrange multipliers and desired class of support vectors}}, \quad b_2 = b, \quad \text{and} \quad g = \tanh.$$

The number of support vectors is the number of hidden units.

Machine learning – Support Vector Machine

Difference between SVMs and ANNs

- SVM: 
 - based on stronger mathematical theory,
 - avoid over-fitting,
 - not trapped in local-minima,
 - works well with fewer training samples.
- But limited to binary classification
 - extensions to regression in (Vapnik et al., 1997) and PCA in (Schölkopf et al, 1999).
- Tuning SVMs is tough:
 - selecting a specific kernel (feature space) and regularization parameter C done by trial and errors.

Machine learning – Support Vector Machine

Similarity between SVMs and ANNs (in binary classification)

- Similar formalisms:

$$y = \langle \mathbf{w}, \varphi(\mathbf{x}) \rangle + b \leq 0 ?$$

SVM

- $\varphi(\mathbf{x})$: feature vector,
- φ : non-linear function implicitly defined by the kernel K ,
- UAT: φ could be approximated by a shallow NN.

ANN

- $\varphi(\mathbf{x})$: output of the last hidden layer,
- φ : non-linear recursive function learned by backpropagation,
- the output of hidden layers can be interpreted as feature vectors.

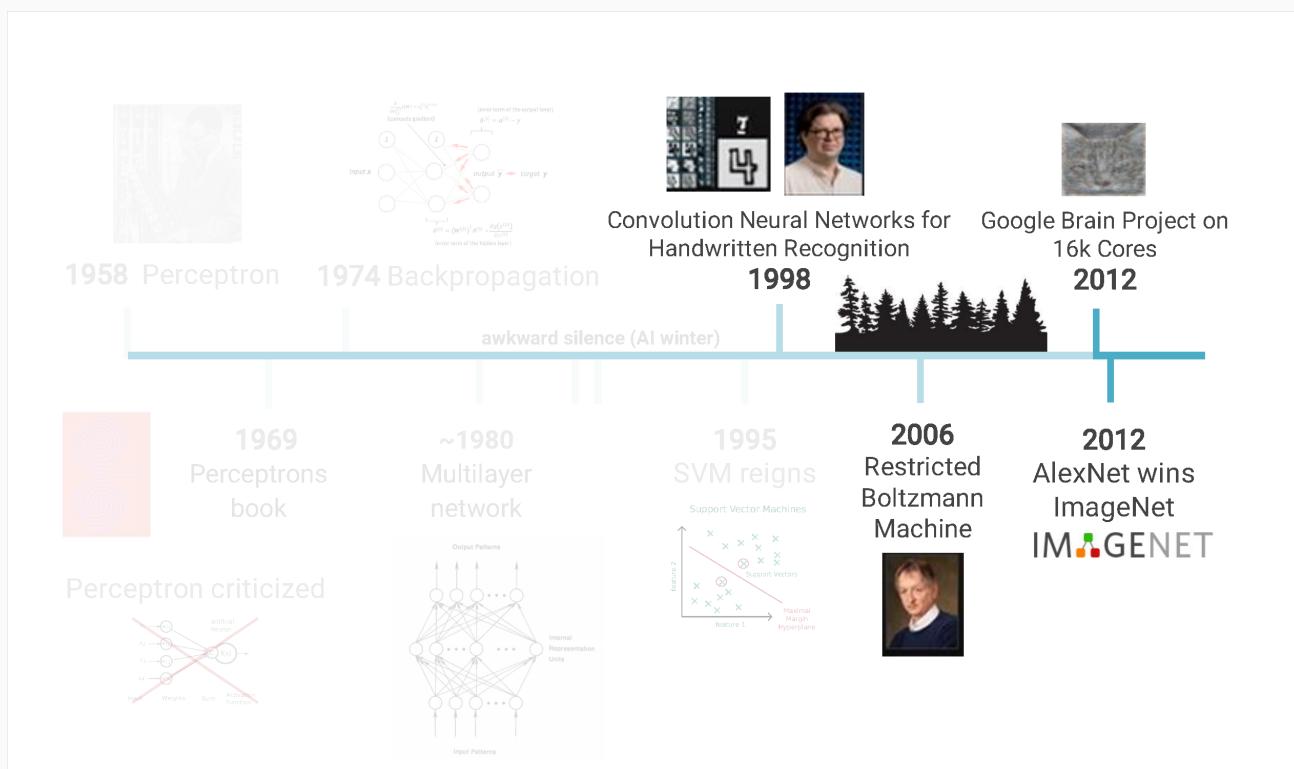
Machine learning – Support Vector Machine

Similarity between SVMs and ANNs (in binary classification)

- Both are linear separators in a suitable feature space,
- Mapping to the feature space is obtained by a non-linear transform,
- For SVM, the non-linear transform is chosen by the user.
Instead, ANNs learn it with backpropagation.
- ANNs can also do it recursively layer by layer:
 - this is called a feature hierarchy,
 - this is the **foundation of deep learning**.

Machine learning – Deep learning

What's next?



(Source: Lucas Masuch & Vincent Lepetit)

Questions?

Next class: Introduction to deep learning

Slides and images courtesy of

P. Gallinari

V. Lepetit

A. O. i Puig

L. Masuch

A. W. Moore

C. Hazırbaş

M. Welling