Course 10

Kernel methods. Classical and deep neural networks.

Kernel methods in similarity-based learning

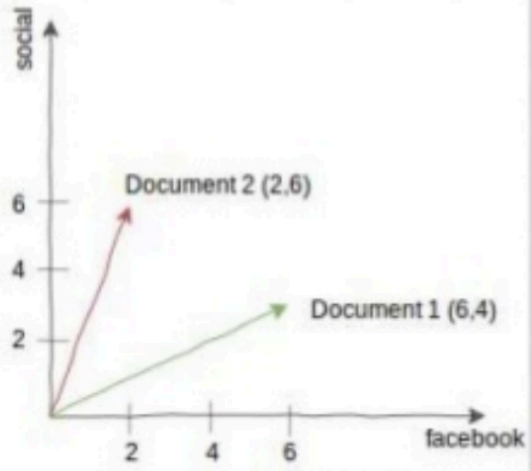
Following (Ionescu, 2018)

Remember

The Vector Space Model

- The representation of a set of objects as vectors in a common vector space is known as the **vector space model**.
  - dimensions are features (words, similarities between a sample and a training sample, TF-IDF, etc.)
  - queries are also vectors

	doc1	doc2
facebook	6	2
social	3	6



From (Verberne, 2018)

Remember

Dot product based similarity

- A vector $\vec{V}(d)$ – derived from the object *d*, with one component for each feature
  - the value of a component being a number, or the *tf-idf* weighting score, or anything else
- How do we quantify the similarity between two objects in this vector space?
  - 1st attempt: compute the magnitude of the vector difference between the vectors of the two objects… critics!
  - A better solution: compute the *cosine similarity*:

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

Remember

Properties of the inner (dot) product

- $\langle \cdot, \cdot \rangle$: $V \times V \rightarrow \mathbf{R}$ with the properties:
  - symmetry: $\langle x, y \rangle = \langle y, x \rangle$
  - linearity: $\langle ax, y \rangle = a\langle y, x \rangle$; $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$
  - positive definiteness: $\langle x, x \rangle \geq 0$; $\langle x, x \rangle = 0 \Leftrightarrow x = 0$

$$\left\langle \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\rangle := x^{\mathsf{T}} y = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + \cdots + x_n y_n$$

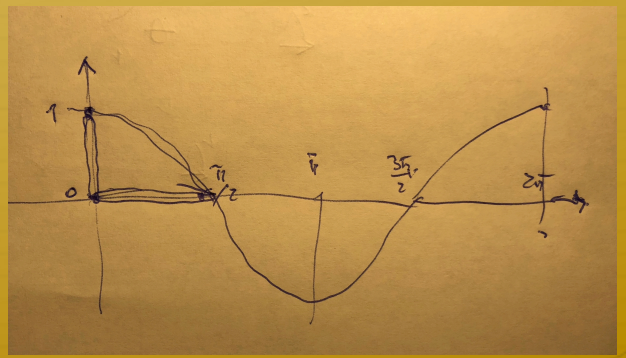
where x^{T} is the transpose of x.

Remember

Computing similarity

- Dot product: $\sum_{i=1}^{M} x_i y_i$
- Euclidean length: $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$
- Unit vectors: $\vec{v}(d_1) = \vec{V}(d_1)/|\vec{V}(d_1)|$ and $\vec{v}(d_2) = \vec{V}(d_2)/|\vec{V}(d_2)|$
- Similarity: $\text{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2)$
- Similarity = cosine of the angle between the two objects

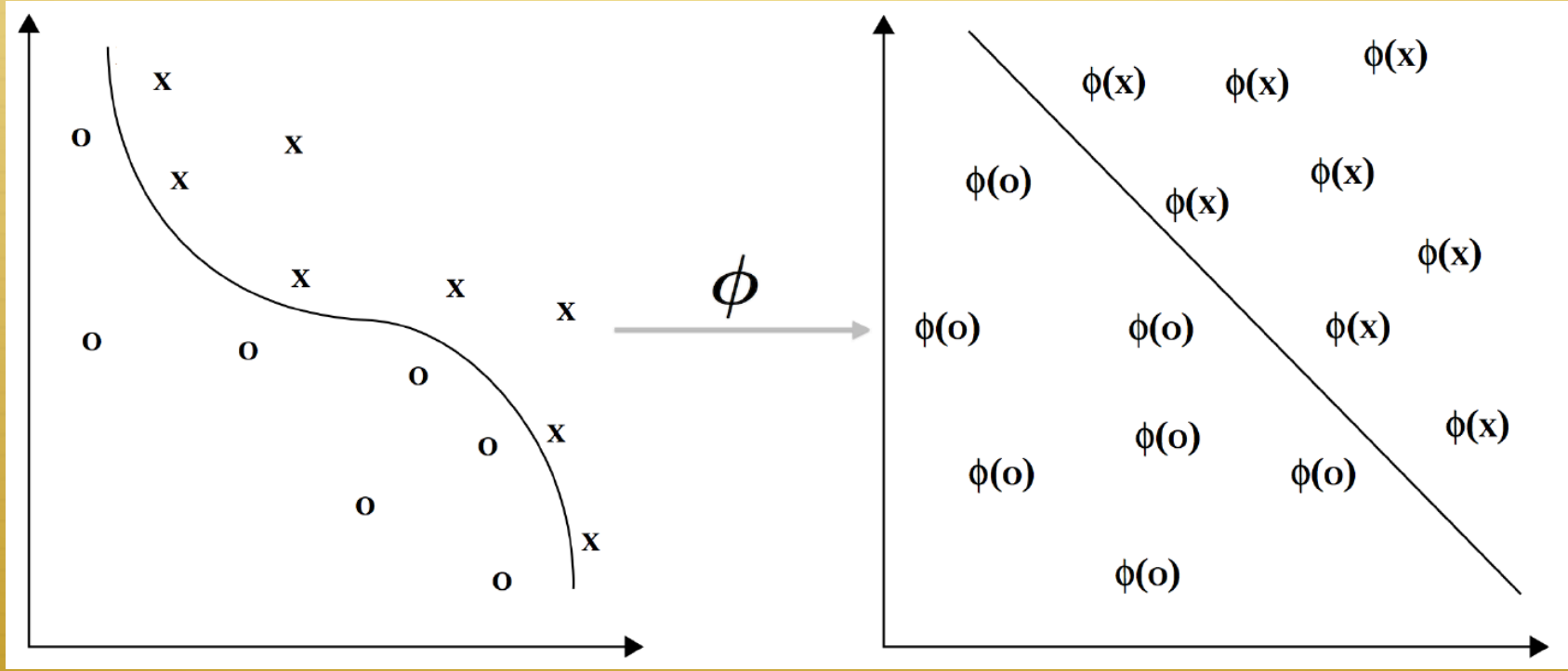
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$



Inner product space (Hilbert space)

- A vector space X over the set of real numbers **R** is an inner product space, if there exists a real-valued symmetric bilinear (linear in each argument) map $\langle \cdot, \cdot \rangle$, that satisfies $\langle x, x \rangle \geq 0$, for all $x \in X$.
  - the bilinear map is known as the inner product, dot product or scalar product
  - inner product space is sometimes referred to as a Hilbert space; $\mathbf{R}^m$ is a Hilbert space ($\mathcal{H}$)

Motivation for using kernels: feature embedding. Nonlinear patterns are converted into linear



The function ϕ embeds the data into a feature space where the nonlinear relations now appear linear. Machine learning methods can easily detect such linear relations.

Kernels in inner product spaces

- A kernel method performs a mapping into an embedding or feature space. An embedding map (or feature map) is a function:

$$\phi : x \in \mathbb{R}^m \longmapsto \phi(x) \in F \subseteq \mathcal{H}$$

  where *F* is an inner product feature space.
- A kernel is a function *k* that for all *x*, z $\in$ *X* satisfies:

$$k(x, z) = \langle \phi(x), \phi(z) \rangle$$

  - the choice of the map ϕ aims to convert the nonlinear relations from *X* into linear relations in the embedding space *F*

Kernel methods

- Kernel-based learning algorithms work by embedding the data into a Hilbert space and by searching for linear relations in that space, using a learning algorithm.
  - to embed data in a Hilbert space = whatever the input data looks like, embed (*transformă*/*încorporează*/*scufundă*) its representation in a multidimensional space (> 3 dimensions)
  - embedding is performed implicitly, i.e. by specifying the inner product between each pair of points rather than by giving their coordinates explicitly
- A kernel = the inner product in a Hilbert space
  - another interpretation: the pairwise similarity between samples

Why is the kernel function important?

- Kernel methods (through kernel functions) offers the power to naturally handle input data that is not in the form of numerical vectors, such as strings, images, or even video and audio files.
- The kernel function captures the intuitive notion of similarity between objects in a specific domain and can be any function defined on the respective domain that is symmetric and positively defined.
  - in computational biology and computational linguistics (Shawe-Taylor and Cristianini, 2004)
  - in image interpretation (Lazebnik *et al*., 2006; Grauman and Darrell, 2005)

The kernel matrix

- Given a set of vectors $\{x_1, \ldots x_n\}$ and a kernel function k employed to evaluate the inner products in a feature space with feature map ϕ, the kernel matrix is defined as the $n \times n$ matrix K with entries given by:

$$K_{ij} = <\phi(x_i), \phi(x_i)\}> = k(x_i, x_j)$$

- A function k: $X \times X \rightarrow \mathbf{R}$ which is either continuous or has a finite domain, can be decomposed by a feature map ϕ into a Hilbert space F applied to both its arguments followed by the evaluation of the inner product in F as follows:

$$k(x, z) = <\phi(x), \phi(z)>,$$

if and only if it is finitely positive semi-definite.

Example and properties of kernel functions

- Linear kernel: obtained by computing the inner product of two vectors. The map function in this case is $\phi(x) = x$.
  - Example: If $x = (1, 2, 4, 1)$ and $z = (5, 1, 2, 3)$ are two vectors in R^4, then $k(x, z) = \langle x, z \rangle = 1 \cdot 5 + 2 \cdot 1 + 4 \cdot 2 + 1 \cdot 3 = 18$
- Let k_1 and k_2 be two kernels over $X \times X$, $X \subseteq \mathcal{H}$, $a \in \mathbf{R}^+$, $f(.)$ - a real-valued function on X, and M a symmetric positive semi-definite $n \times n$ matrix. Then the following functions are also kernels:

$k(x, z) = k_1(x, z) + k_2(x, z)$
$k(x, z) = a\, k_1(x, z)$
$k(x, z) = k_1(x, y) \cdot k_2(x, z)$
$k(x, z) = f(x) \cdot f(z)$
$k(x, z) = x^T M z$

See (Ionescu, 2018), pag. 44-45, for more examples of kernel functions.

M is called positive semi-definite if $x^T M x \geq 0$ for all x in $\mathbf{R}^n$.

Kernel classifiers

- In binary classification problems, kernel-based learning algorithms look for a discriminant function, which assigns +1 to examples that belong to one class and -1 to examples that belong to the other class.
- A linear function: $f(x) = \text{sign}(\langle w, \phi(x) \rangle + b)$

Kernel normalisation

- Given a kernel *k*(*x*; *z*) that corresponds to the feature map ϕ, the normalized kernel ^*k*(*x*; *z*) corresponds to the feature map given by:

$$x \longmapsto \phi(x) \longmapsto \frac{\phi(x)}{\|\phi(x)\|}$$

- Normalization helps to improve machine learning performance.
- Examples:
  - normalization makes the values of each feature in the data have zero-mean and unit-variance
  - divide each component by the square root of the product of the two corresponding diagonal components:

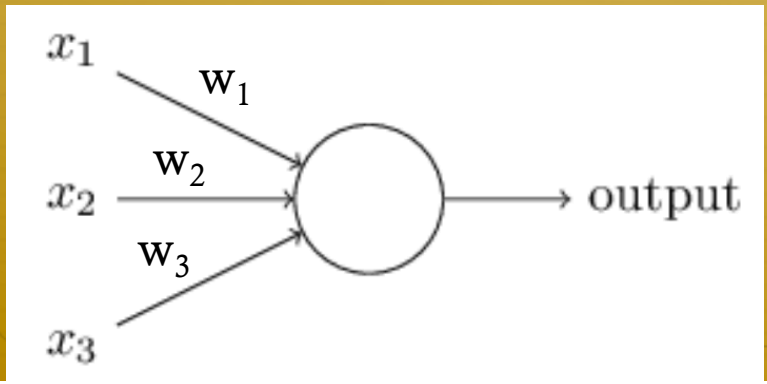
$$\hat{k}(x_i, x_j) = \frac{k(x_i, x_j)}{\sqrt{k(x_i, x_i) \cdot k(x_j, x_j)}}$$

O scurtă revedere a rețelelor neurale

Following: (Nielsen, 2015).

Neural networks in AI

- Inspired from medicine, where it should refer to **neuronal networks**
- Artificial neuron - called **perceptron**
  - concept introduced in the 1950s and 1960s (by Frank Rosenblatt, inspired by a prior work of Warren McCulloch and Walter Pitts)
  - more modern: the **sigmoid neuron**



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Networks with and without feedback

- Neural networks where the output of a layer is used as input to the next layer => **feedforward** neural networks.
  - there are no loops in the network - the information is always fed forward, not backwards
- If the output is fetched to internal nodes => **recursive networks**

Another way to write the network ecuation

$\sum_j w_j x_j$ = w . x (inner product of two vectors: of weights and thresholds)

Noring b = - threshold =>

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Algorithm 2: Generic Algorithm of Kernel Methods

Input:

$\mathcal{S} = \{(x_i, t_i) \mid x_i \in \mathbb{R}^m, t_i \in \{+1, -1\}, i \in \{1, 2, ..., n\}\}$ – the set of n training samples and labels;

$Z = \{z_i \mid z_i \in \mathbb{R}^m, i \in \{1, 2, ..., l\}\}$ – the set of l test samples;

k – a kernel function;

$\mathcal{C}$ – a binary kernel classifier.

Initialization:

$K \leftarrow \mathbf{0}_n$;

$K^{test} \leftarrow \mathbf{0}_{l,n}$;

Computation:

for $x_i \in X$ **do**
    for $x_j \in X$ **do**
        $K_{ij} \leftarrow k(x_i, x_j)$;

for $z_i \in Z$ **do**
    for $x_j \in X$ **do**
        $K^{test}_{ij} \leftarrow k(z_i, x_j)$;

$(\alpha, b) \leftarrow$ the dual weights of $\mathcal{C}$ trained on K with the labels T;

$Y \leftarrow K^{test}\alpha + b$;

biases

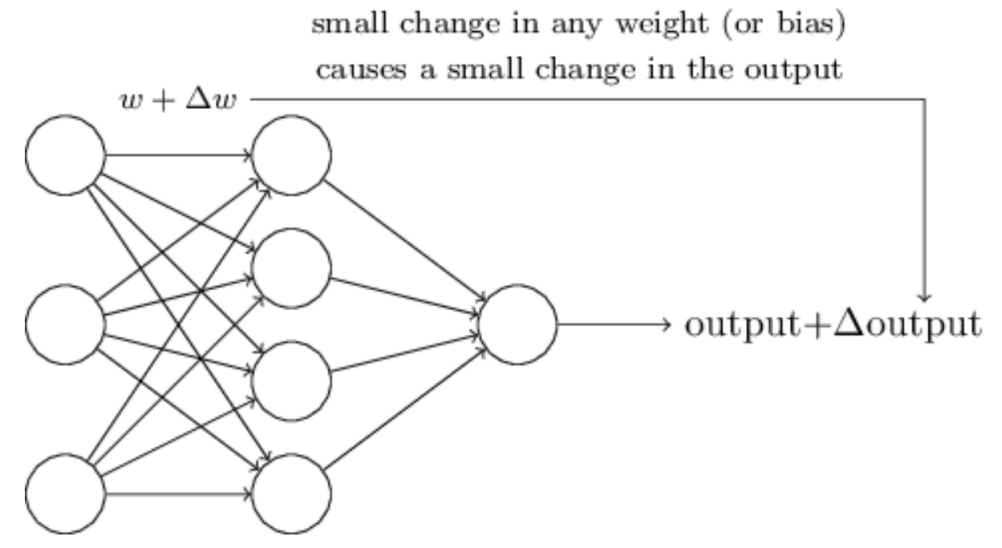
weights

Output:

$Y = \{y_i \mid y_i \in \{+1, -1\}, i \in \{1, 2, ..., l\}\}$ – the set of predicted labels for the test samples in Z.

From (Ionescu, 2018), pag. 47

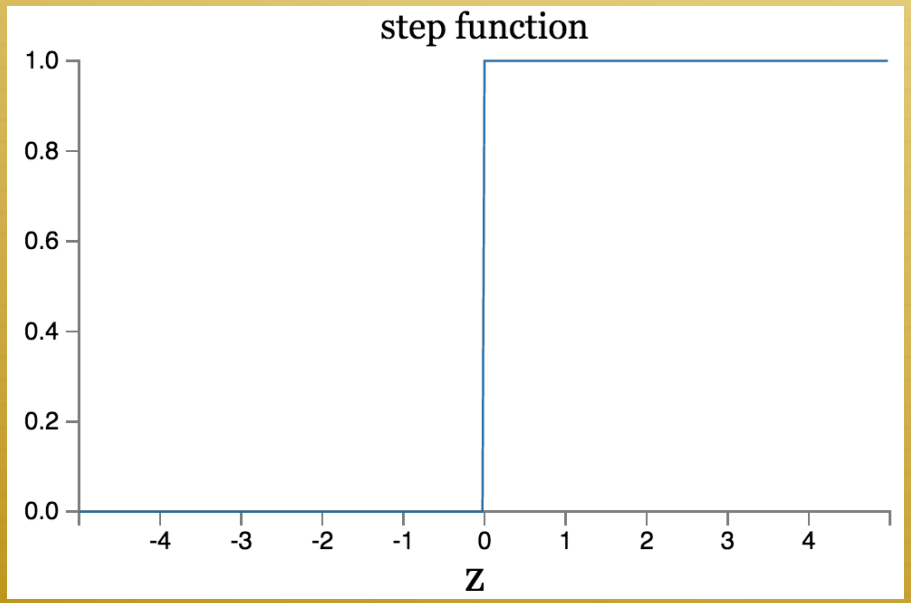
How can a neural network learn?



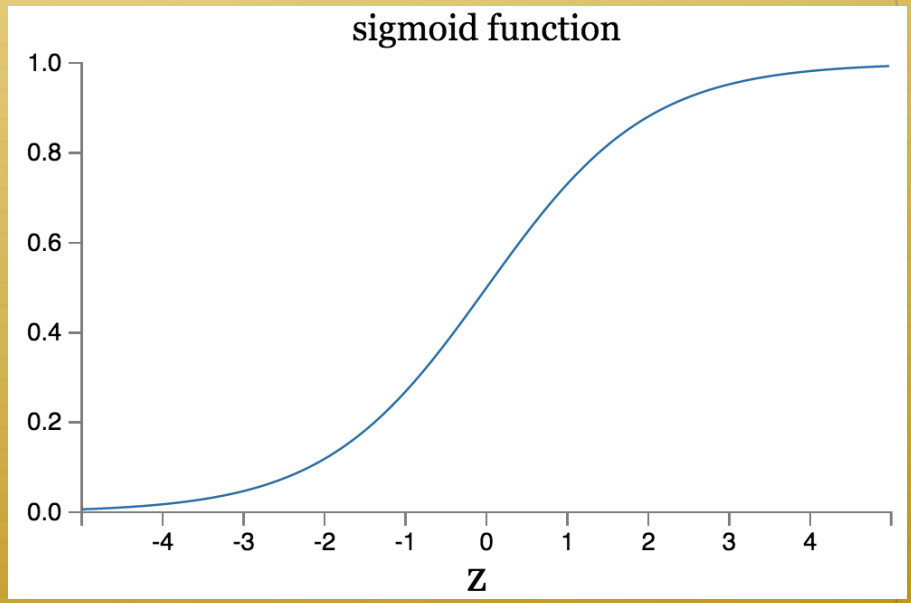
- A small change in the weight of an arc of the network ...
  - can cause complete switching of the output, e.g. from 0 to 1
  - if a certain exit would now be corrected in the way we want, it is possible that the rest of the entries be severely affected in a way that is difficult to control
- => We need a way to gradually change weights and polarities so the network gets closer to the desired behavior
  - a network where switching is not radical, as the one given by a threshold function ...

The sigmoid function

- output = $\sigma(w \cdot x + b)$, where: $\sigma(z) \equiv \frac{1}{1+e^{-z}}$



perceptron



sigmoid neuron

To learn = to minimize an error function (cost, objective, loss)

$$C(w, b) \equiv \frac{1}{2n} \sum_{x} \|y(x) - a\|^2$$

where:

- w = weights matrix
- b = biases vector
- n = dimension of input (length of x)
- x = input
- $y(x)$ = desired output
- $a(w, b, x)$ = computed output of the network

Gradient descent rule

- Assuming a C that depends on a set of input variables $v = (v_1, \ldots v_m)$, a variation ΔC of C, produced by a small variation of the input:

  $\Delta v = (\Delta v_1, \ldots \Delta v_m)^T$ is: $\Delta C \approx \nabla C \,.\, \Delta v$,

  where: $\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T$ is the gradient.

- If we choose: $\Delta v = -\eta \nabla C$, with $\eta > 0$, then we assure that the ΔC variation will be negative (because we want the error to be smaller and smaller at each step).

- Update rule: $v \rightarrow v' = v - \eta \nabla C$

algoritmul de *gradient descent*

How to apply gradient descent?

- Decreasing the gradient can be seen as a way of making small steps in the direction that makes *C* fall faster.
- The gradient vector has the components $\partial C/\partial w_k$ and $\partial C/\partial b_l$, such that:

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l}$$

the rules that make the network learn

But this means that the output influences the network. Still this is not a recurrent network (because only weights and biases are influenced, not the inputs of the neurons).

Stochastic descent of gradient

- In practice, the number of input variables and biases can be extremely high, so iterating gradient descent on each of them may be unrealistic.
- At each step in the training we randomly choose a number of inputs X_1, ... X_m and we estimate ΔC only based on them, by mediating the results <= a process similar to voting polls:

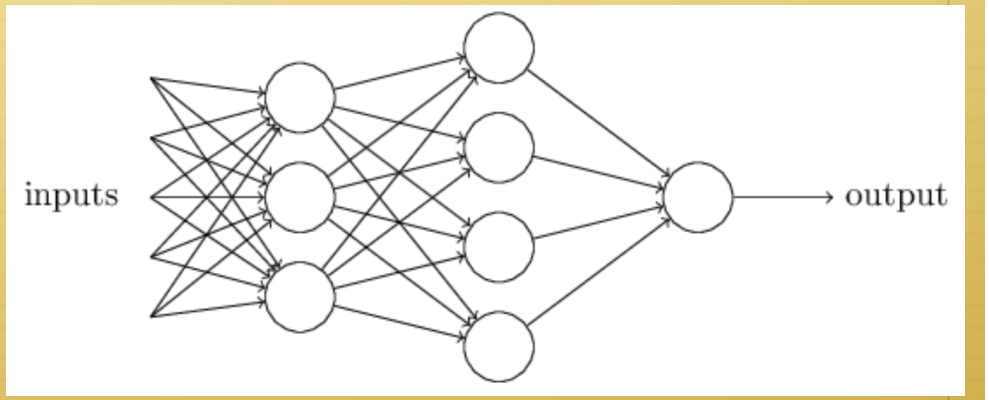
$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

- Summarizing:

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k} \qquad b_l \to b_l' = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

and at each step – another sample.

Multilayer (deep) networks



- Layer 2 perceptrons make decisions by weighing the results of the first layer
  - decisions at a more complex and abstract level than those in the first layer
  - even more complex decisions can be taken by perceptrons in the third layer
  - a network of many layers of perceptrons can engage in a sophisticated decision

Feedforward through a deep neural network

- The loss $L(\hat{y}, y)$ depends on the output $\hat{y}$ and the goal y.
- To obtain the loss J, a regularizer $\Omega(\theta)$, where θ contains all parameters (weights and biases) can be added to the loss.

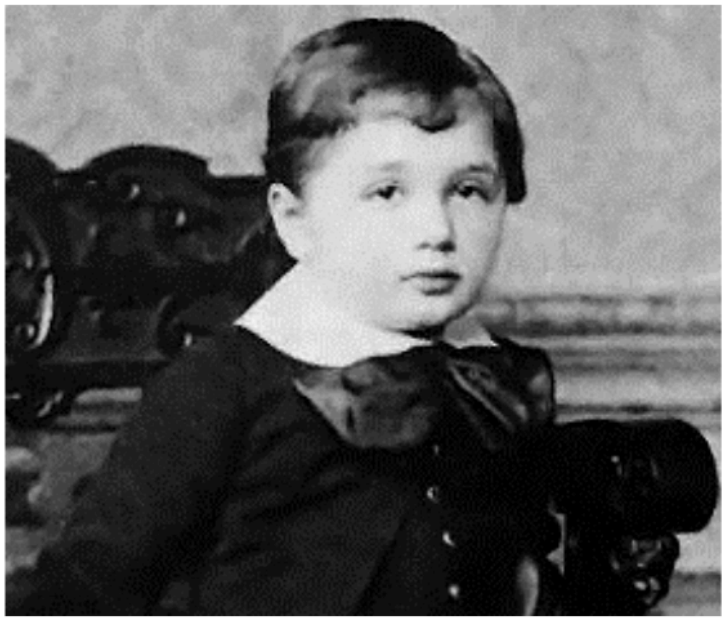
Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

From: (Goodfellow *et al.*, 2016)

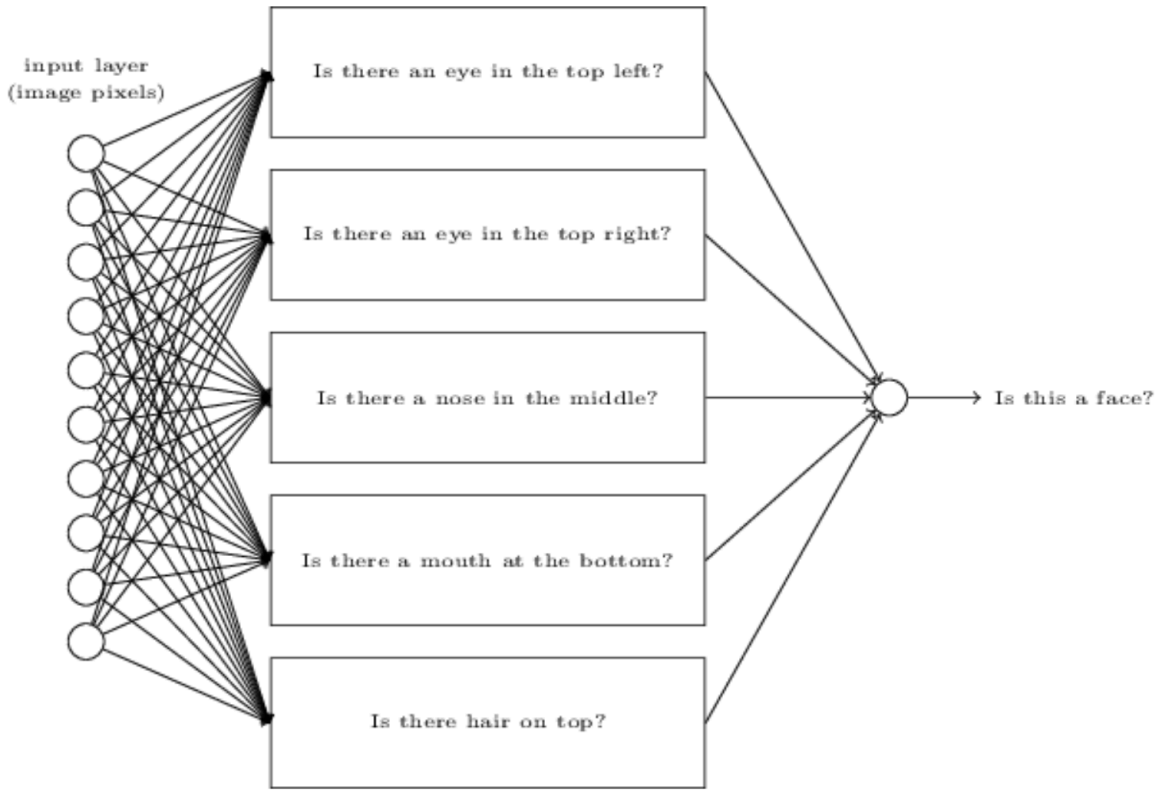
Deep learning

- pwe want to find out if a picture contains a human face:

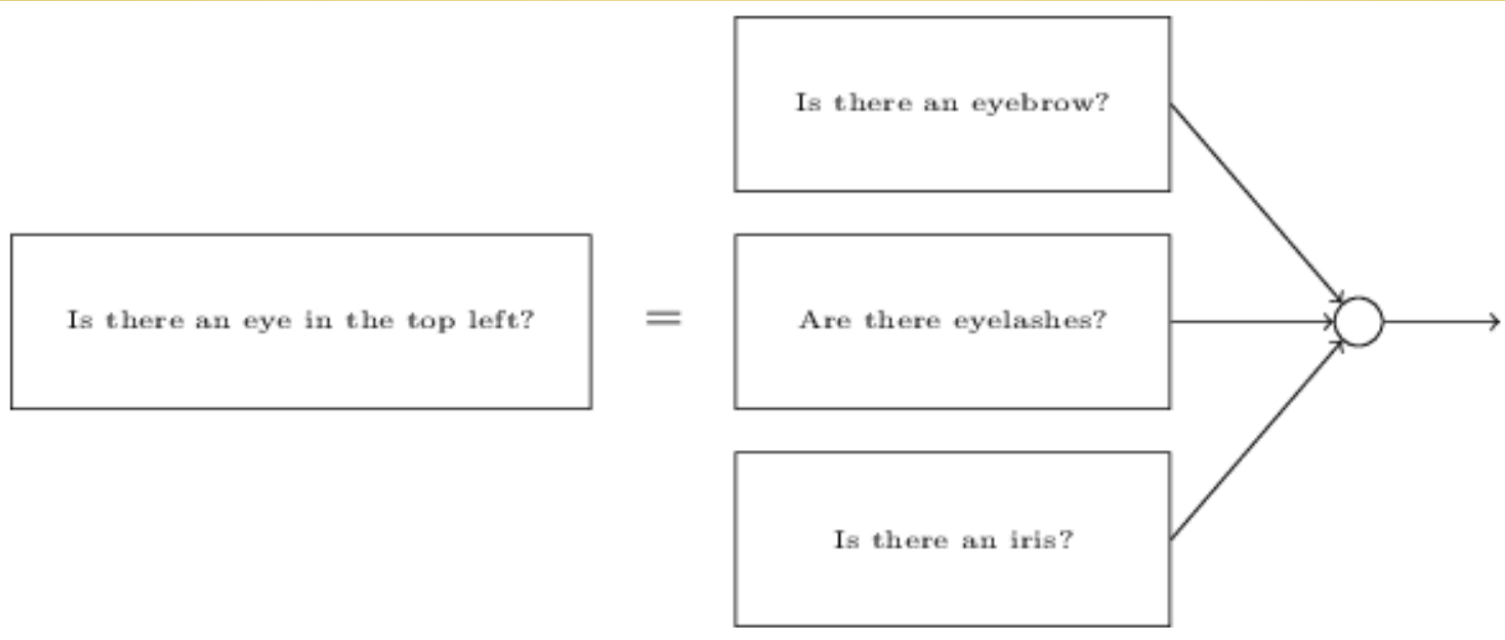




Decompose the problem in simpler sub-problems

- Is there an eye on the top left? Is there an eye on the top right? Is there a nose in the middle? Is there a mouth down in the middle? Is hair above? And so on.



And these... in others even simpler



- But how do you solve the problem automatically by proposing 5-10 intermediate (hidden) layers of neurons that address increasingly general concepts?...

Bibliography

- Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron (2016). Deep Learning, MIT Press (http://www.deeplearningbook.org)
- Grauman, Kristen and Darrell, Trevor (2005). The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features. In Proceedings of ICCV, volume 2, pp. 1458{1465.
- Ionescu, Radu Tudor (2018). Habilitation thesis. Knowledge Transfer between Computer Vision, Text Mining and Computational Biology: New Chapters, University of Bucharest.
- Lazebnik, Svetlana, Schmid, Cordelia, and Ponce, Jean (2006). Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In Proceedings of CVPR, volume 2, pp. 2169{2178, Washington, IEEE Computer Society.
- Nielsen, Michael A. (2015). Neural networks and deep learning, Determination Press. http://neuralnetworksanddeeplearning.com/chap1.html
- Shawe-Taylor, John and Cristianini, Nello (2004). Kernel Methods for Pattern Analysis, Cambridge University Press.
- Verberne, Suzan (2018). Word2Vec Tutorial, Leiden Univ., March. https://twitter.com/suzan/status/977158060371316736