Week 04

Multilayer Perceptron 1

Prof. Ph.D. Woo Youn Kim

Chemistry, KAIST

Sources

- The perceptron: http://natureofcode.com/book/chapter-10-neural-networks/#chapter10_figure3
- MLP: https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks
- MLP: http://norman3.github.io/prml/docs/chapter05/
- Andrew Ng: http://cs229.stanford.edu/syllabus.html
- Forward and Back propagation: http://alinlab.kaist.ac.kr/resource/Lec1_Introduction_to_NN.pdf
- History of deep learning: watch before lecture https://www.youtube.com/watch?v=n7DNueHGkqE&feature=youtu.be

https://www.youtube.com/watch?v=AByVbUX1PUI

Contents

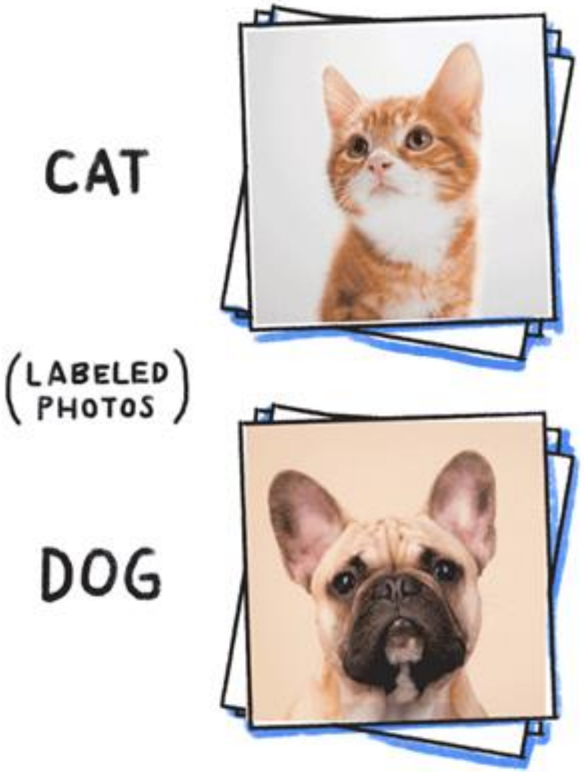
- Concept of deep learning
  - Neuron
  - Perceptron
  - Activation function
- Multilayer perceptron
  - Nonlinearity
  - Universal approximation
- Back propagation
  - Example
  - Vanishing gradient
  - ReLU

Concept of deep learning

Concept of machine learning

“easy-for-a-human, difficult-for-a-machine” tasks,

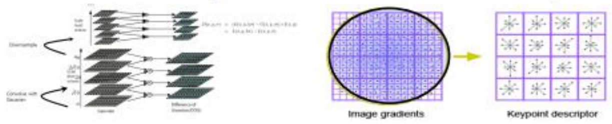
often referred to as pattern recognition.



Why deep learning?

Classical ML (or shallow learning)

Input → Feature Extraction → Other ML → Output

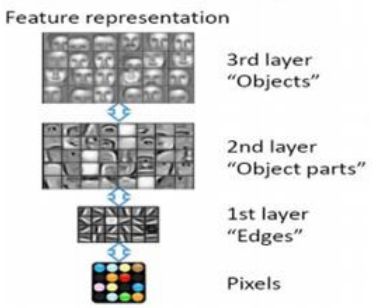


SIFT

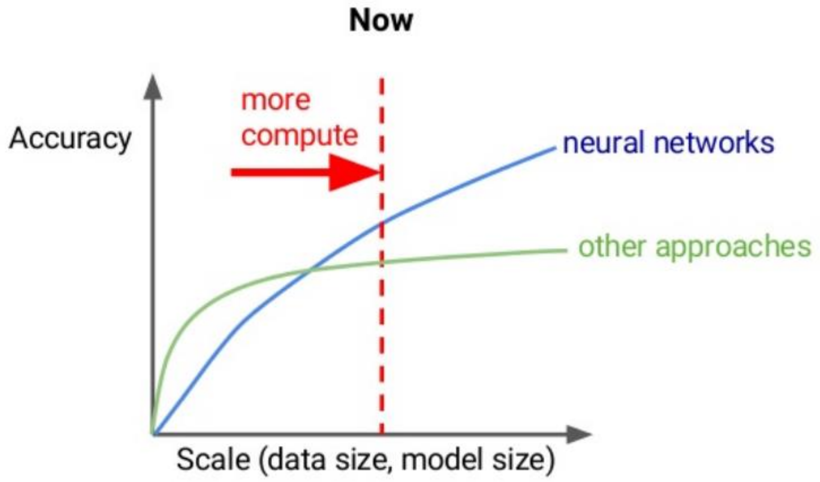
hand-crafted features by experts

Deep learning

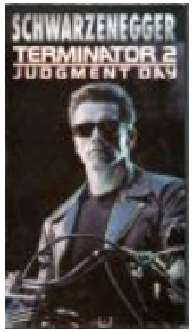
Input → Deep Network → Output



automated feature extraction by AI



Terminator 2 (1991)



JOHN: Can you learn? So you can be... you know. More human. Not such a dork all the time.

TERMINATOR: My CPU is a neural-net processor... a learning computer. But **Skynet** presets the switch to "read-only" when we are sent out alone.

We'll learn how to **set** the neural net

. . .

TERMINATOR Basically. (starting the engine, backing out) The **Skynet** funding bill is passed. The system goes on-line August 4th, 1997. Human decisions are removed from strategic defense. **Skynet** begins to learn, at a geometric rate. It becomes **self-aware** at 2:14 a.m. eastern time, August 29. In a panic, they try to pull the plug.

SARAH: And **Skynet** fights back.

TERMINATOR: Yes. It launches its ICBMs against their targets in Russia.

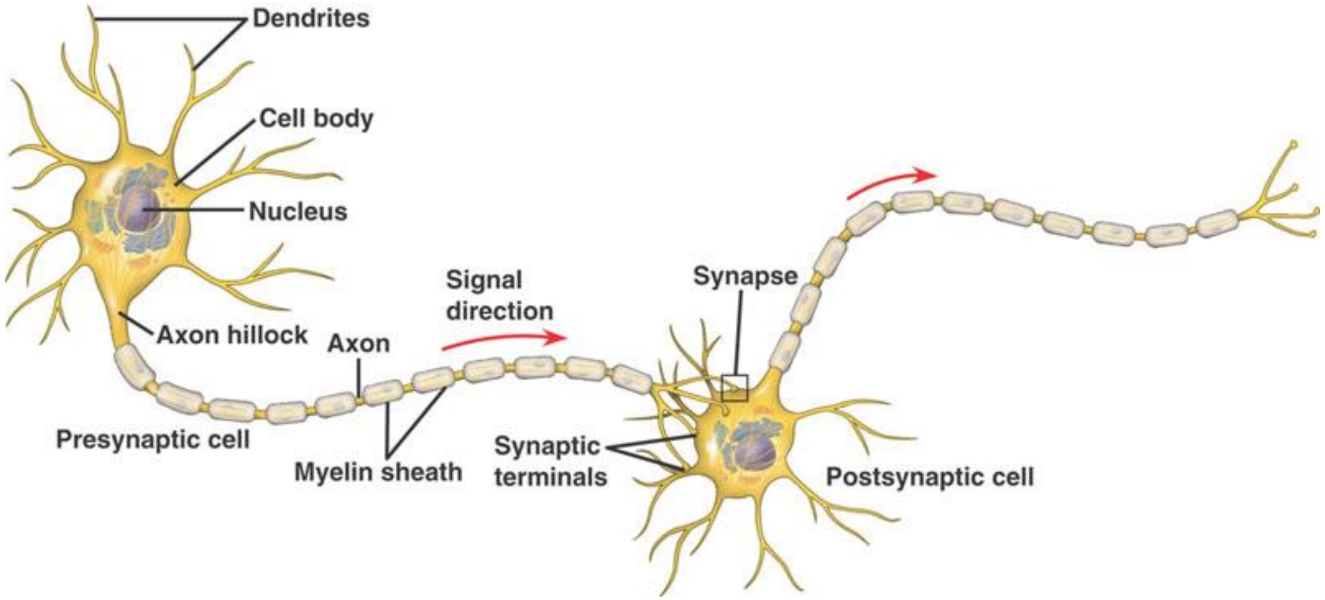
SARAH: Why attack Russia?

TERMINATOR: Because **Skynet** knows the Russian counter-strike will remove its enemies here.

http://pages.cs.wisc.edu/~jerryzhu/cs540/handouts/neural.pdf

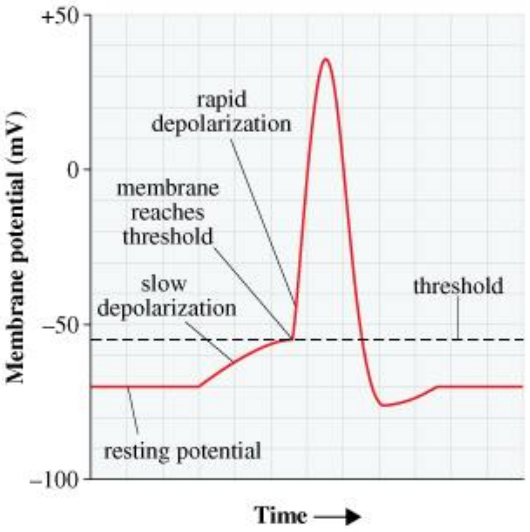
Neuron & synapse

Electrical and chemical signal transmission



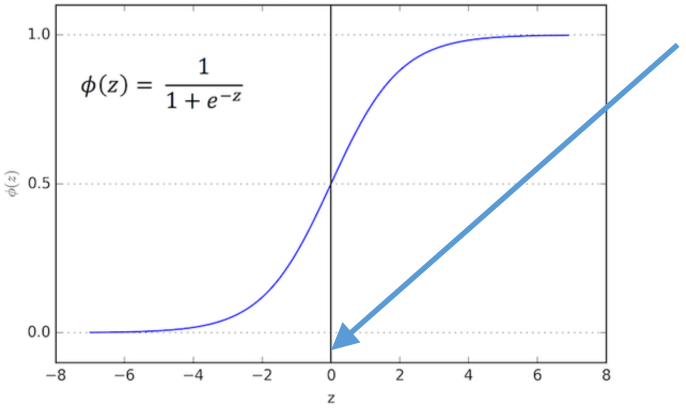
Input **Output**

Analog signal transmission



Digital signal: on-off

threshold



Logistic (sigmoid) function

Neural network

Hyper-connections

Complexity of human brain

- 10^{11} neurons
- each neuron has ~7,000 synaptic connections
- 3 years old: 10^{15} synapses
- adult: 10^{14}~$5X10^{14}$ synapses

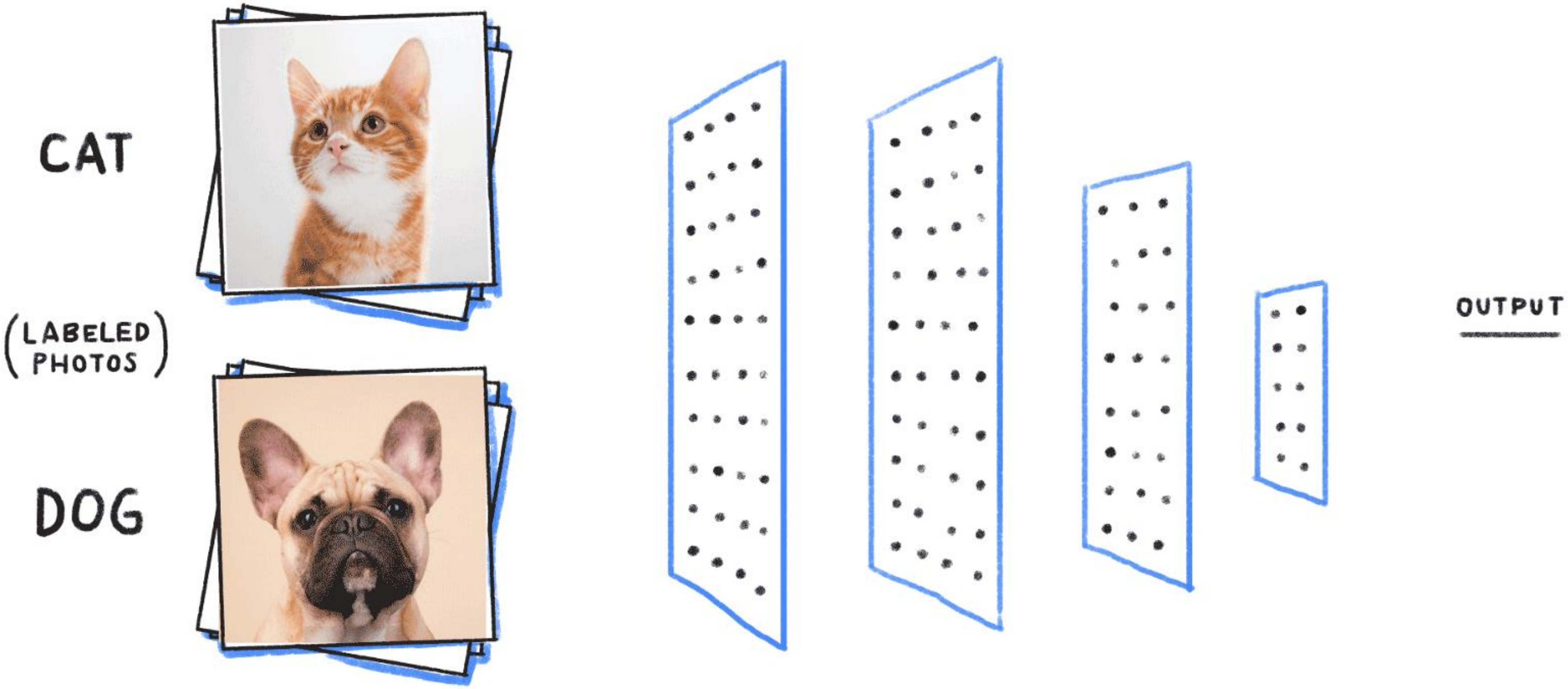
➔ complex functions

One of the key functions of a neural network is its ability to *learn*.

A neural network is a complex ***adaptive*** system; it can change its internal structure based on the information flowing through it by adjusting the amplitude of signals (***weight***).

Perception

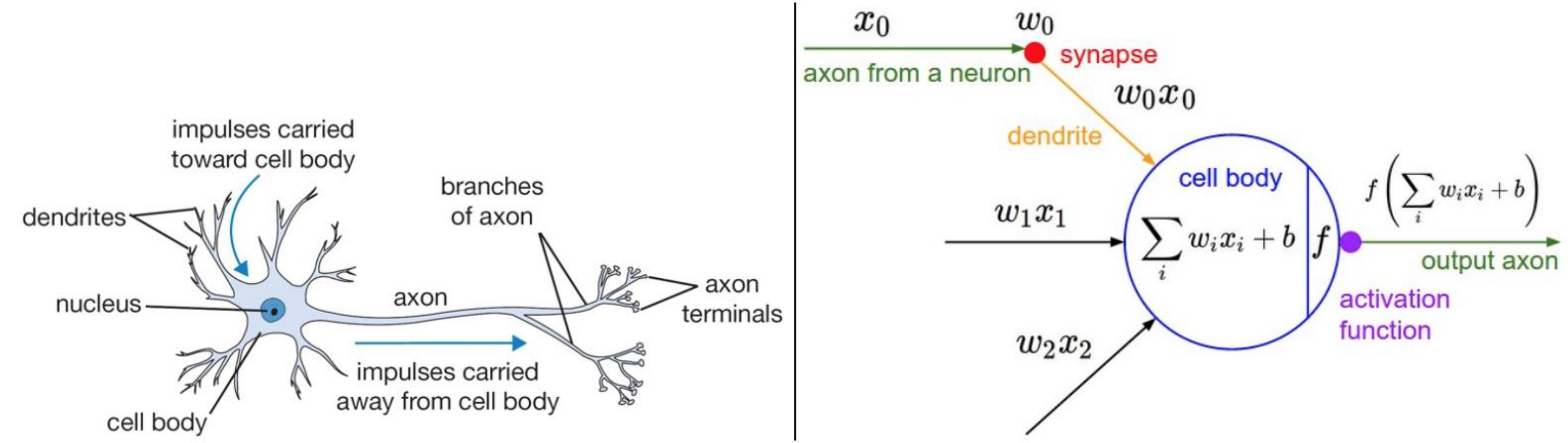
"easy-for-a-human, difficult-for-a-machine" tasks, often referred to as pattern recognition.



Unique signal pattern

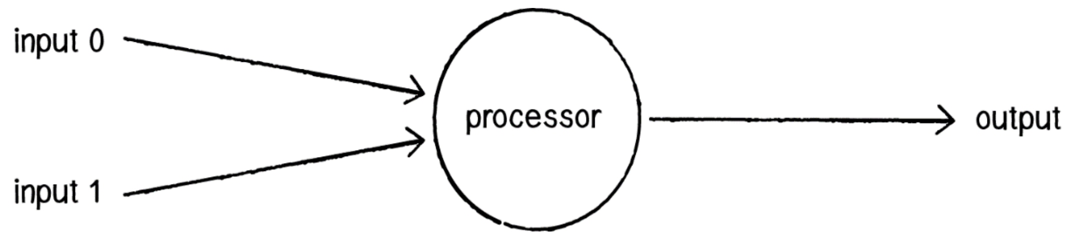
Perceptron

Invented in 1957 by Frank Rosenblatt at the Cornell Aeronautical Laboratory, a perceptron is the simplest neural network possible: a computational model of a single neuron.



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Principle of perceptron



inputs processor single output

$$\text{output} = \begin{cases} 0 \text{ if } f(\boldsymbol{w}\cdot\boldsymbol{x}) < \text{threshold} \\ 1 \text{ if } f(\boldsymbol{w}\cdot\boldsymbol{x}) \geq \text{threshold} \end{cases}$$

$$\boldsymbol{w}\cdot\boldsymbol{x} = \sum_j w_j x_j$$

Step 1: receive inputs

Input 0: $x_1 = 12$
Input 1: $x_2 = 4$

Step 2: weight inputs

$x_1 \times w_1 = 12 \times 0.5 = 6$
$x_2 \times w_2 = 4 \times -1 = -4$

Step 3: sum inputs

Sum = 6 + -4 = 2

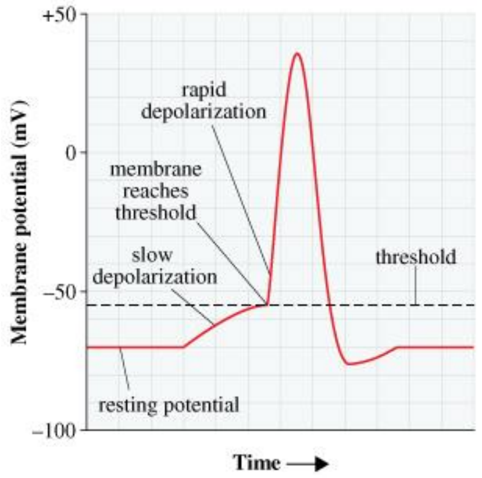
Step 4: generate output

Output = sign(sum)
= sign(2) = +1

Bias

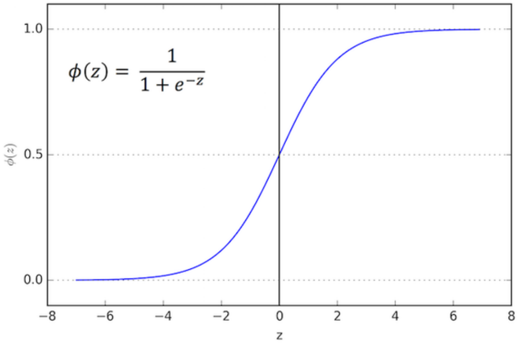
$$\text{output} = \begin{cases} 0 \text{ if } f(\boldsymbol{w}\cdot\boldsymbol{x}) < \text{threshold} \\ 1 \text{ if } f(\boldsymbol{w}\cdot\boldsymbol{x}) \geq \text{threshold} \end{cases} \quad \Rightarrow \quad \text{output} = \begin{cases} 0 \text{ if } f(\boldsymbol{w}\cdot\boldsymbol{x}+b) < 0 \\ 1 \text{ if } f(\boldsymbol{w}\cdot\boldsymbol{x}+b) \geq 0 \end{cases}$$

move the threshold inside the perceptron
and then make it a learning parameter



a measure of how easy it is to get the perceptron to output 1.

or a measure of how easy it is to get the perceptron to *fire*.



Linear problems

$$\text{output} = \begin{cases} 0 \text{ if } f(\boldsymbol{w} \cdot \boldsymbol{x} + b) < 0 \\ 1 \text{ if } f(\boldsymbol{w} \cdot \boldsymbol{x} + b) \geq 0 \end{cases}$$

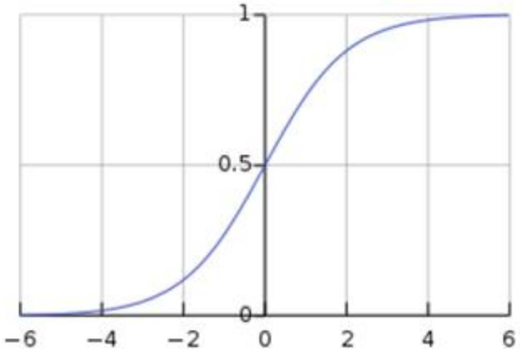
$$\Rightarrow$$

$$\text{output} = \begin{cases} 0 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \\ 1 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \end{cases}$$

Activation functions

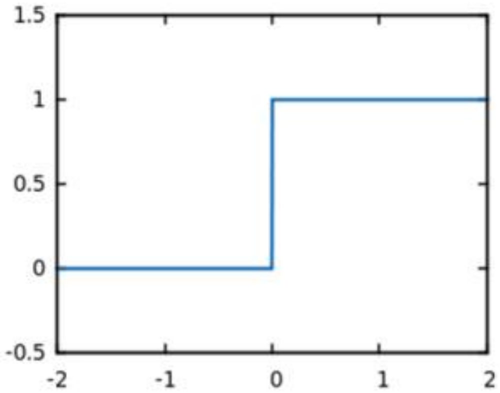
$$\text{output} = \begin{cases} 0 \text{ if } f(\boldsymbol{w} \cdot \boldsymbol{x} + b) < \text{threshold} \\ 1 \text{ if } f(\boldsymbol{w} \cdot \boldsymbol{x} + b) \geq \text{threshold} \end{cases}$$

✓ **Sigmoid Function**

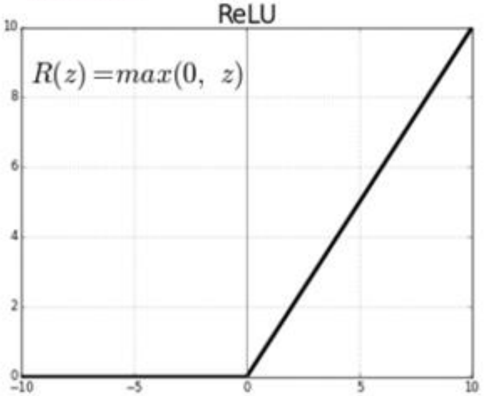


$$h(x) = \frac{1}{1 + \exp(-x)}$$

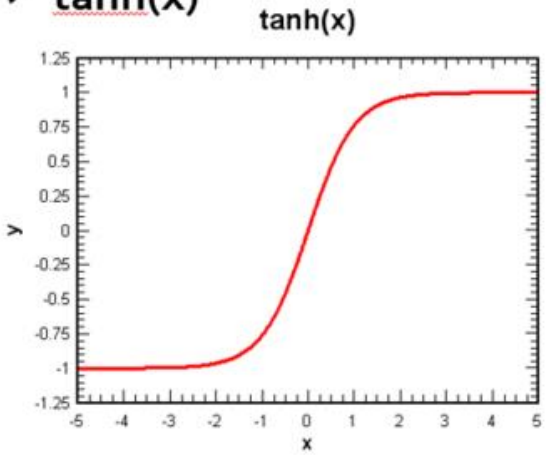
✓ **Step Function**



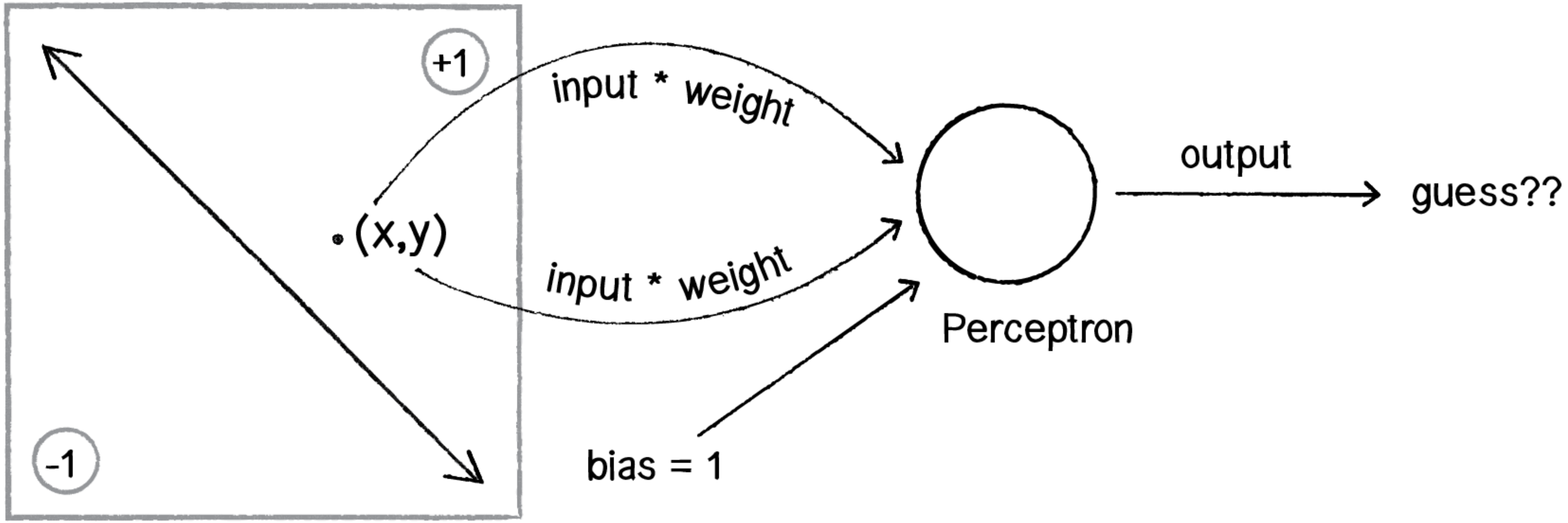
✓ **ReLU**



✓ **tanh(x)**



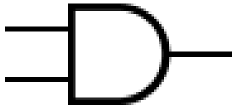
Linear classification

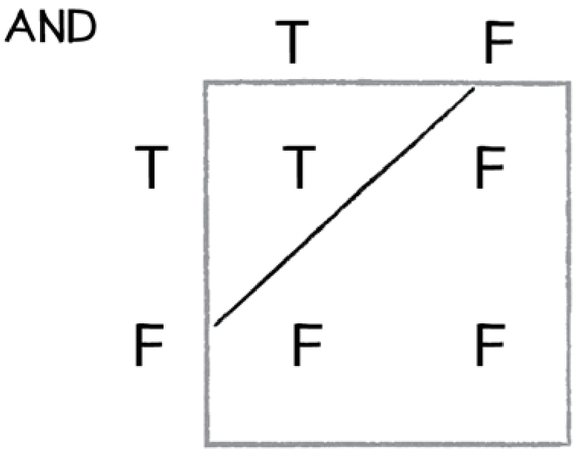


By varying the weights and the threshold, we can get different models of decision-making.

Learning ➔ determining appropriate weights for best output with a given training data set

Logic gate

✓ **AND Gate** 



x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

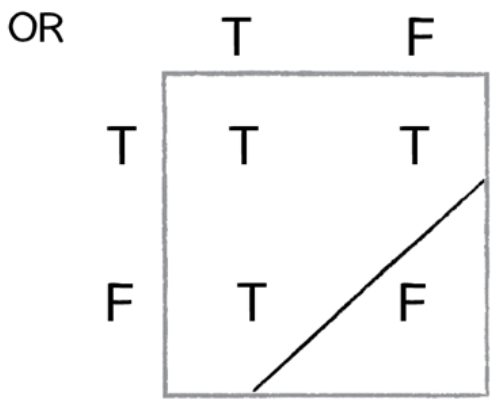
$$\text{output} = \begin{cases} 0 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \\ 1 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \end{cases}$$

```python
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = 0.7

    val = np.sum(w*x) - b
    if tmp <= 0:
        return 0
    else:
        return 1
```

Logic gate

✓ **Logic gate - OR Gate**



x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

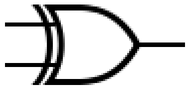
$$\text{output} = \begin{cases} 0 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \\ 1 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \end{cases}$$

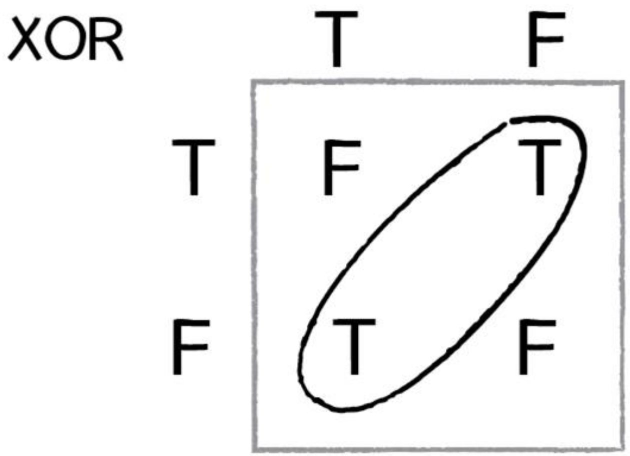
```
def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = 0.2

    val = np.sum(w*x) - b
    if tmp <= 0:
        return 0
    else:
        return 1
```

Logic gate

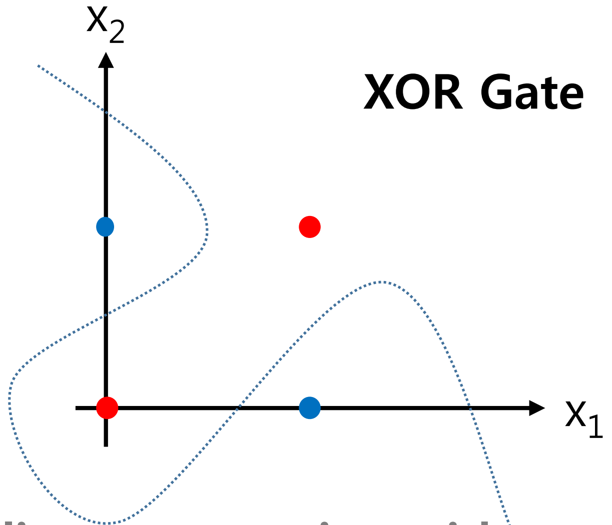
✓ **Logic gate- XOR Gate**





x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

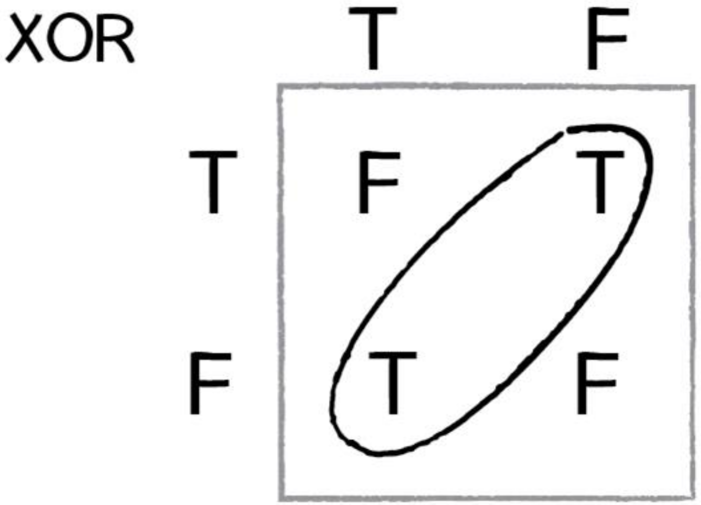
$$\text{output} = \begin{cases} 0 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \\ 1 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \end{cases}$$



Non-linear expression with w and x

Drawback of a single perceptron

The single perceptron approach has one major drawback: it can only learn linearly separable functions. Take XOR, a relatively simple function, and notice that it can't be classified by a linear separator.



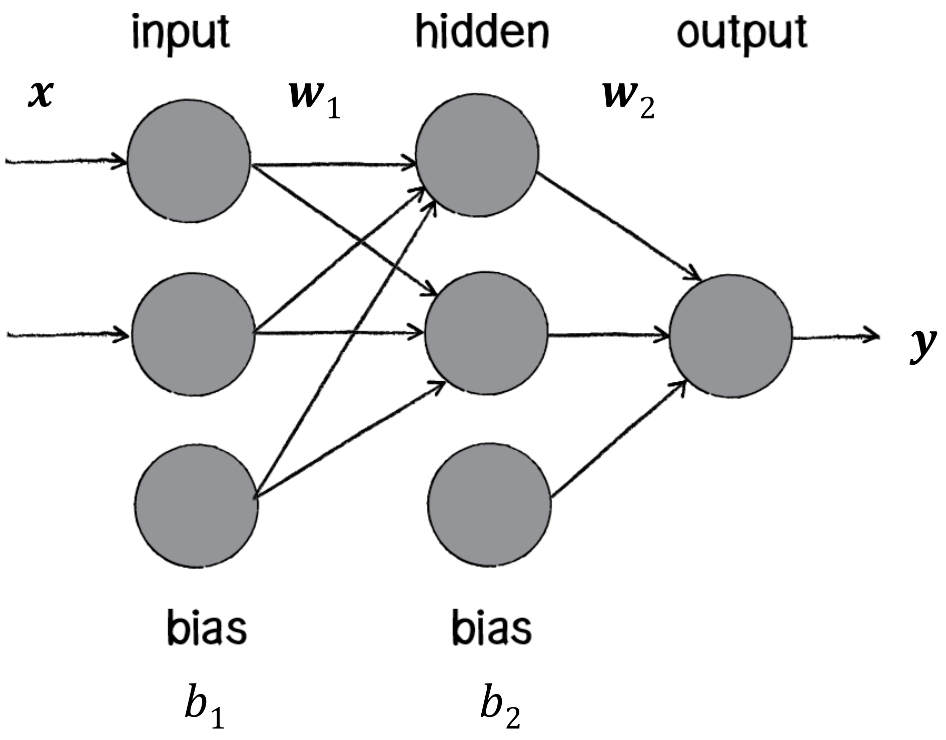
But what if we made a network out of more perceptrons? If one perceptron can solve *OR* and one perceptron can solve *NOT AND*, then two perceptrons combined can solve *XOR*.

➔ Many perceptrons to go beyond linearity

Multilayer perceptron

Multilayer perceptron (MLP)

MLP, a.k.a a forward-feed network



$$\boldsymbol{x} = \{x_1, x_2\}$$

$$h_j = f(\boldsymbol{w}_{j1} \cdot \boldsymbol{x} + b_1)$$

$$y = f(\boldsymbol{w}_2 \cdot \boldsymbol{h} + b_2)$$

Marvin Minsky 1969, founder of MIT AI lab
no way to train the neural network

Nonlinearity

Why does deep and cheap learning work so well?*

Henry W. Lin, Max Tegmark, and David Rolnick
Dept. of Physics, Harvard University, Cambridge, MA 02138
Dept. of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139 and
Dept. of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139

(Dated: July 21 2017)

We show how the success of deep learning could depend not only on mathematics but also on physics: although well-known mathematical theorems guarantee that neural networks can approximate arbitrary functions well, the class of functions of practical interest can frequently be approximated through "cheap learning" with exponentially fewer parameters than generic ones. We explore how properties frequently encountered in physics such as symmetry, locality, compositionality, and polynomial log-probability translate into exceptionally simple neural networks. We further argue that when the statistical process generating the data is of a certain hierarchical form prevalent in physics and machine-learning, a deep neural network can be more efficient than a shallow one. We formalize these claims using information theory and discuss the relation to the renormalization group. We prove various "no-flattening theorems" showing when efficient linear deep networks cannot be accurately approximated by shallow ones without efficiency loss; for example, we show that n variables cannot be multiplied using fewer than 2^n neurons in a single hidden layer.

https://arxiv.org/pdf/1608.08225v4.pdf

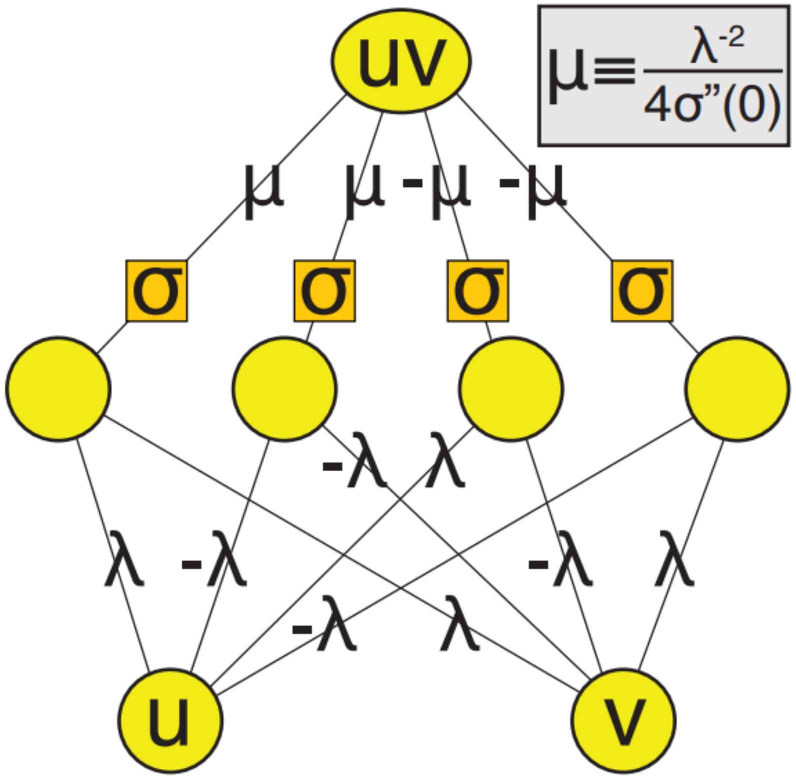
Nonlinearity

Hamiltonians can be expanded as a power series

$$H_y(\mathbf{x}) = h + \sum_i h_i x_i + \sum_{i \le j} h_{ij} x_i x_j + \sum_{i \le j \le k} h_{ijk} x_i x_j x_k + \cdots$$

Nonlinearity

Continuous multiplication gate:



Output

$$m(u,v) = \mu \cdot \{ \sigma[\lambda(u+v)] + \sigma[-\lambda(u+v)] - \sigma[\lambda(u-v)] - \sigma[\lambda(-u+v)] \}$$

The nonlinear activation function can be expanded as

$$\sigma(u) \approx \sigma_0 + \sigma_1 \cdot u + \sigma_2 \cdot \frac{u^2}{2}$$

Substituting it into the output function to obtain

$$m(u,v) = \mu \cdot (\sigma_1\lambda \cdot [(u+v) + (-u-v) - (u-v) - (-u+v)] + \sigma_2\lambda^2 \cdot [(u+v)^2 + (-u-v)^2 - (u-v)^2 - (-u+v)^2)$$

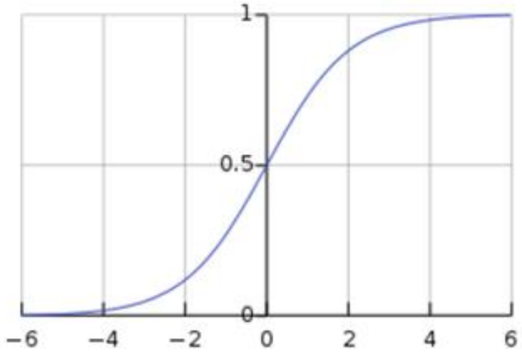
$$= 4\mu \cdot \sigma_2 \cdot \lambda^2 \cdot uv$$

The result is not a linear combination of inputs but multiplication (or nonlinear)!

KAIST

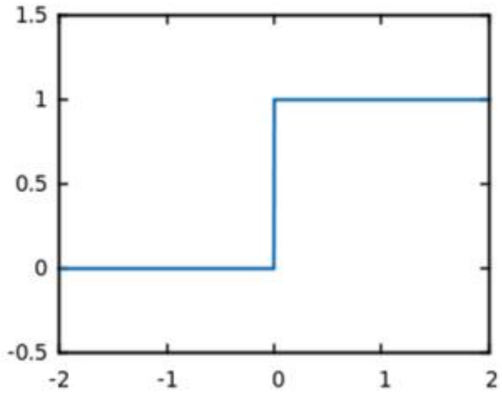
Nonlinearity

✓ **Sigmoid Function**

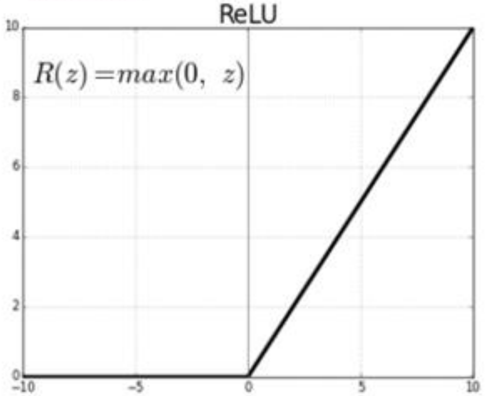


$$h(x) = \frac{1}{1 + \exp(-x)}$$

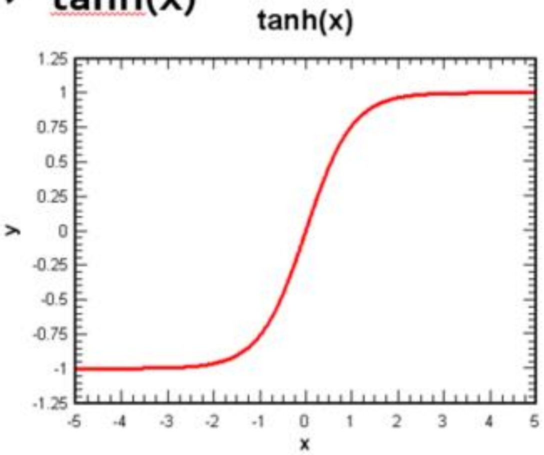
✓ **Step Function**



✓ **ReLU**



✓ **tanh(x)**



The nonlinear activation functions can be expanded as

$$\sigma(u) \approx \sigma_0 + \sigma_1 \cdot u + \sigma_2 \cdot \frac{u^2}{2}$$

The nonlinearity comes from the activation functions and many perceptrons

Universal approximation

Neural Networks, Vol. 2, pp. 359–366, 1989
Printed in the USA. All rights reserved.

0893-6080/89 $3.00 + .00

ORIGINAL CONTRIBUTION

Multilayer Feedforward Networks are Universal Approximators

KURT HORNIK

Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBERT WHITE

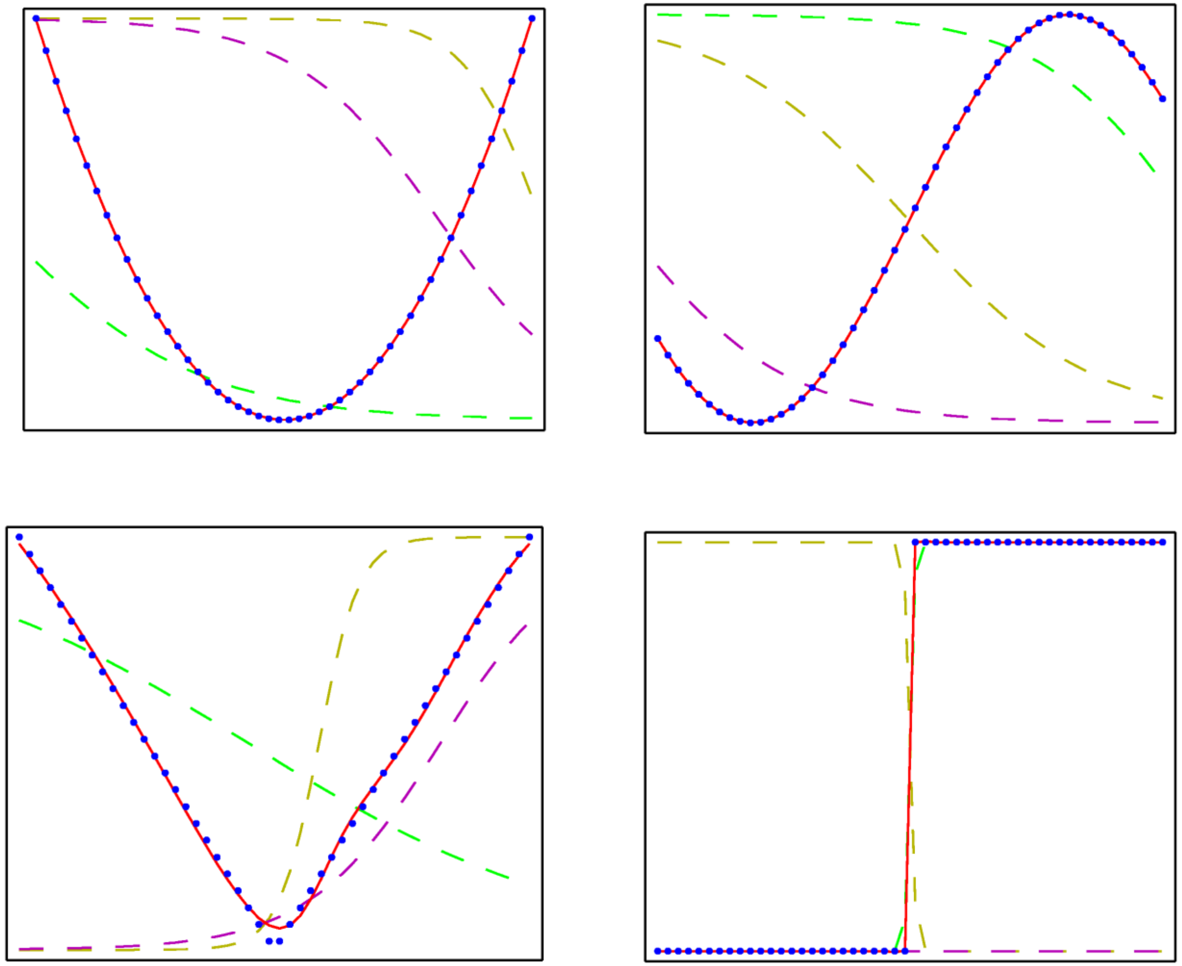
University of California, San Diego

(*Received 16 September 1988; revised and accepted 9 March 1989*)

Abstract—*This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

Universal approximation

Two layer approximation of nonlinear functions: x^2, sinx, |x|, and H(x)

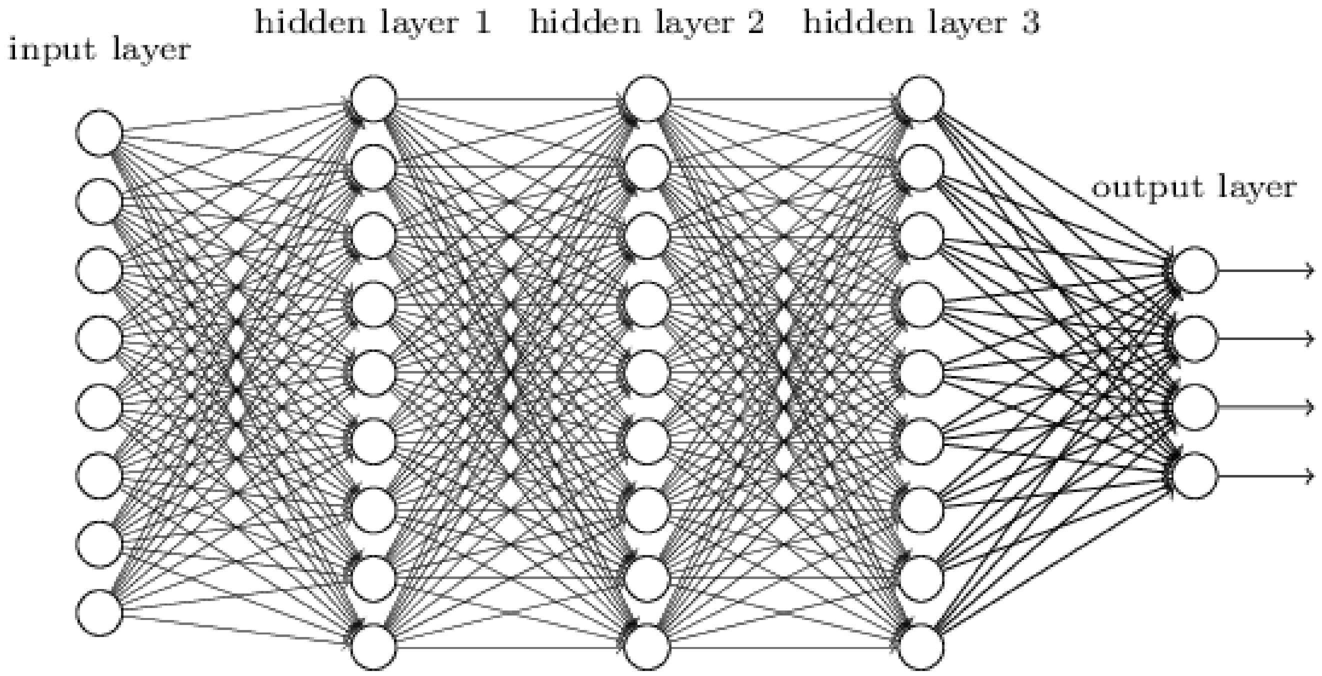


Good bases for spanning a nonlinear function space

Why deeper?

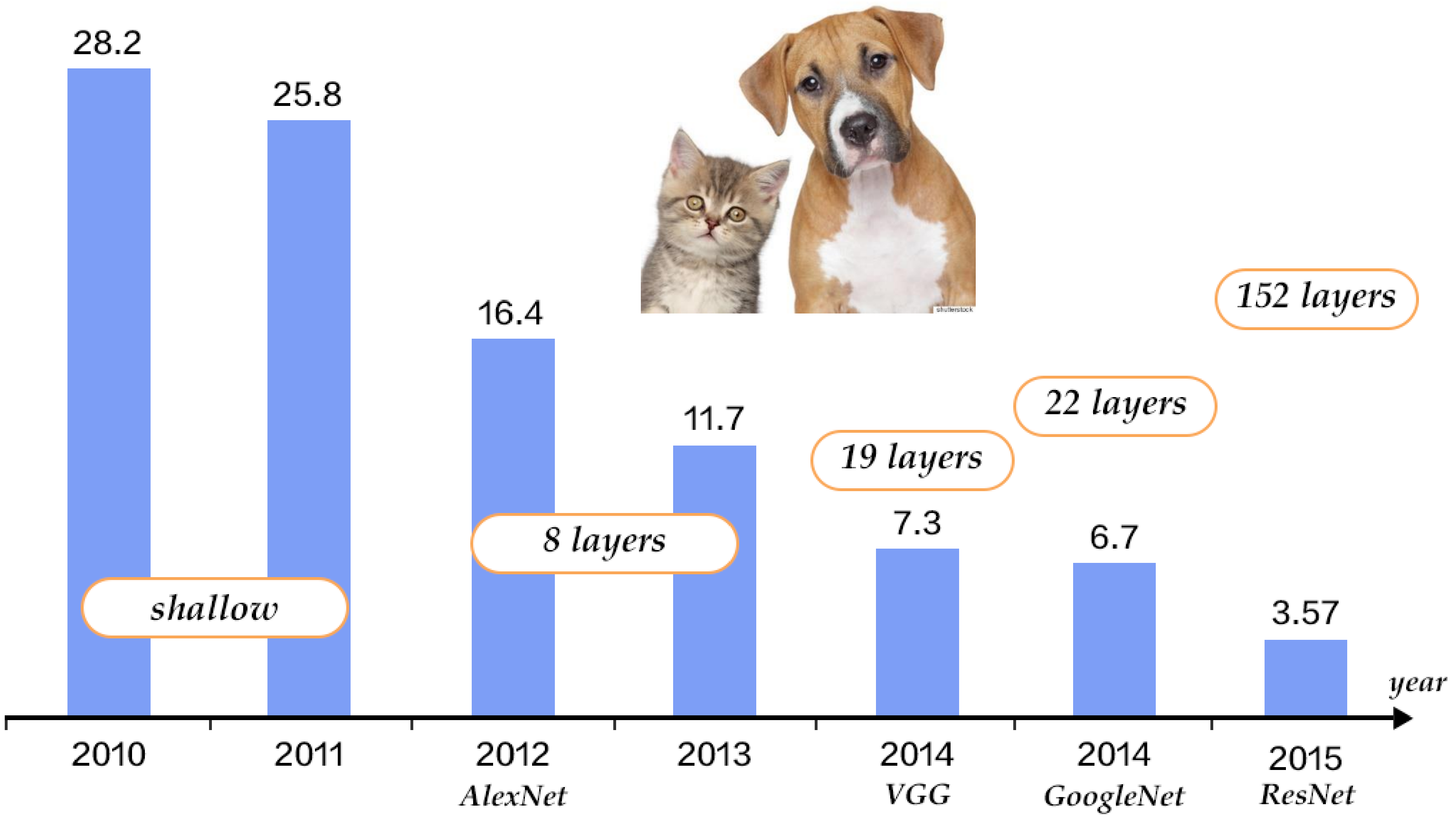
Theorem. For any given multivariate polynomial and any tolerance $\varepsilon > 0$, there exists a neural network of fixed finite size N (independent of ε) that approximates the polynomial to accuracy better than ε. Furthermore, N is bounded by the complexity of the polynomial, scaling as the number of multiplications required times a factor that is typically slightly larger than 4.

https://arxiv.org/pdf/1608.08225v4.pdf



Why deeper?

- **ILSVRC (ImageNet Large Scale Visual Recognition Challenge) Winners**

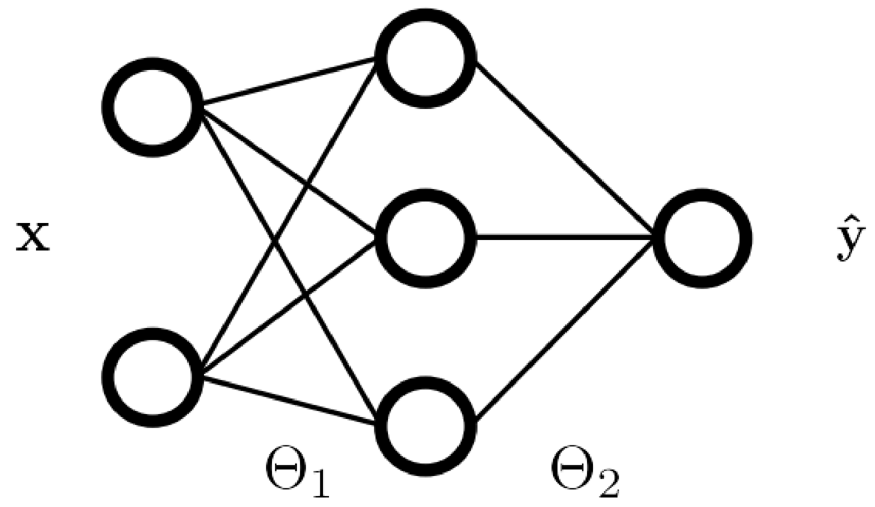


Example: forward propagation

- **Forward propagation**: calculate the output $\hat{\mathbf{y}}$ of the neural network

$$\hat{\mathbf{y}} = \sigma\left(\Theta_k^\top \sigma\left(\Theta_{k-1}^\top \sigma(\cdots \sigma(\Theta_1^\top \mathbf{x}))\right)\right)$$

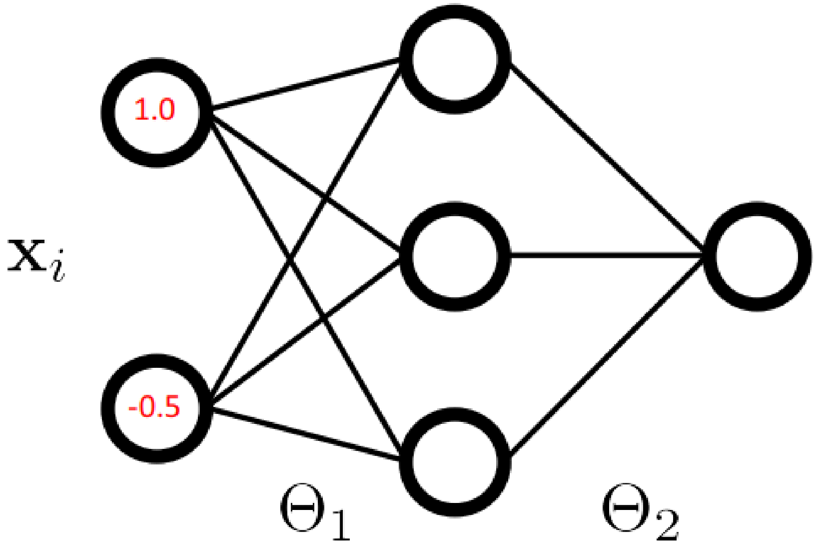
where $\sigma(\cdot)$ is activation function (e.g., sigmoid function) and k is number of layers



Example: forward propagation

$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \qquad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Input data $\mathbf{x}_i$



Example: forward propagation

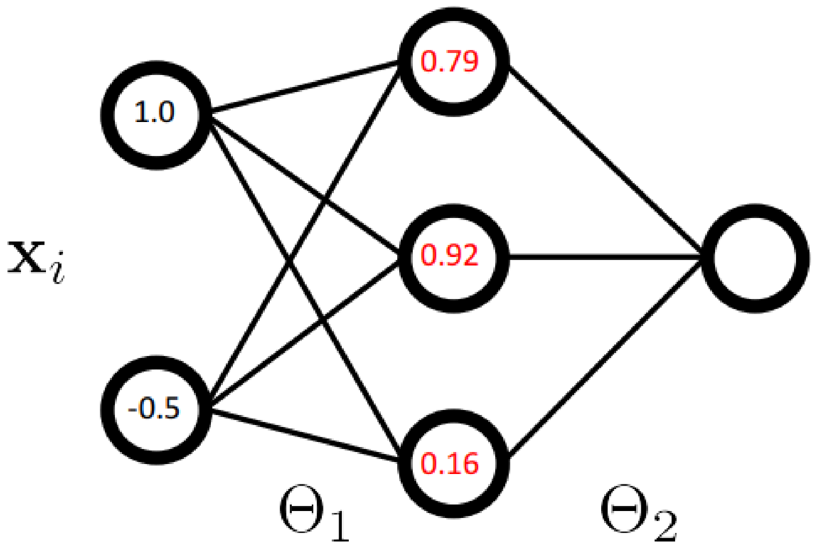
$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \qquad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute hidden units $\mathbf{h}_1$

$$\Theta_1^\top \mathbf{x}_i = \begin{pmatrix} 1.2 & -0.3 \\ 2.1 & -0.7 \\ -1.5 & 0.3 \end{pmatrix} \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} = \begin{pmatrix} 1.35 \\ 2.45 \\ -1.65 \end{pmatrix}$$

$$\mathbf{h}_1 = \sigma(\Theta_1^\top \mathbf{x}_i) = \begin{pmatrix} \sigma(1.35) \\ \sigma(2.45) \\ \sigma(-1.65) \end{pmatrix} = \begin{pmatrix} 0.79 \\ 0.92 \\ 0.16 \end{pmatrix}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$



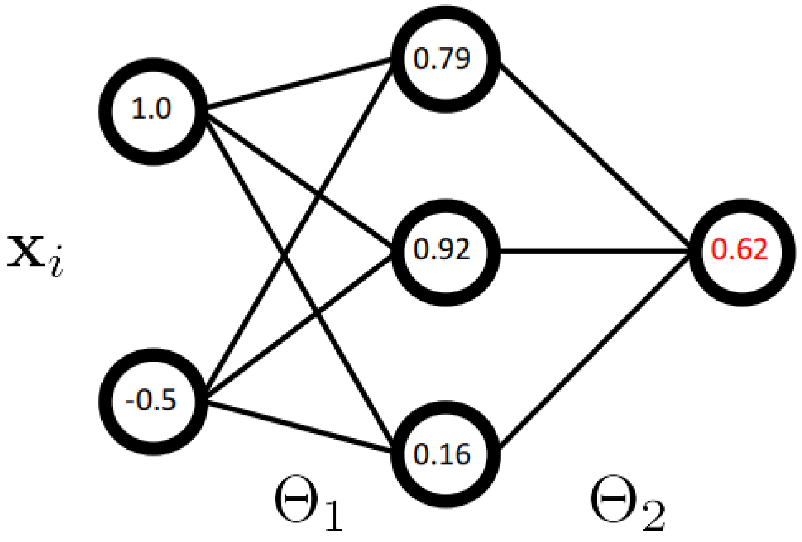
Example: forward propagation

$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \qquad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute output $\hat{y}_i$

$$\Theta_2^\top \mathbf{h}_1 = \begin{pmatrix} -0.2 & 0.5 & 1.3 \end{pmatrix} \begin{pmatrix} 0.79 \\ 0.92 \\ 0.16 \end{pmatrix} = 0.51$$

$$\hat{y}_i = \sigma(\Theta_2^\top \mathbf{h}_1) = \sigma(0.51) = 0.62$$

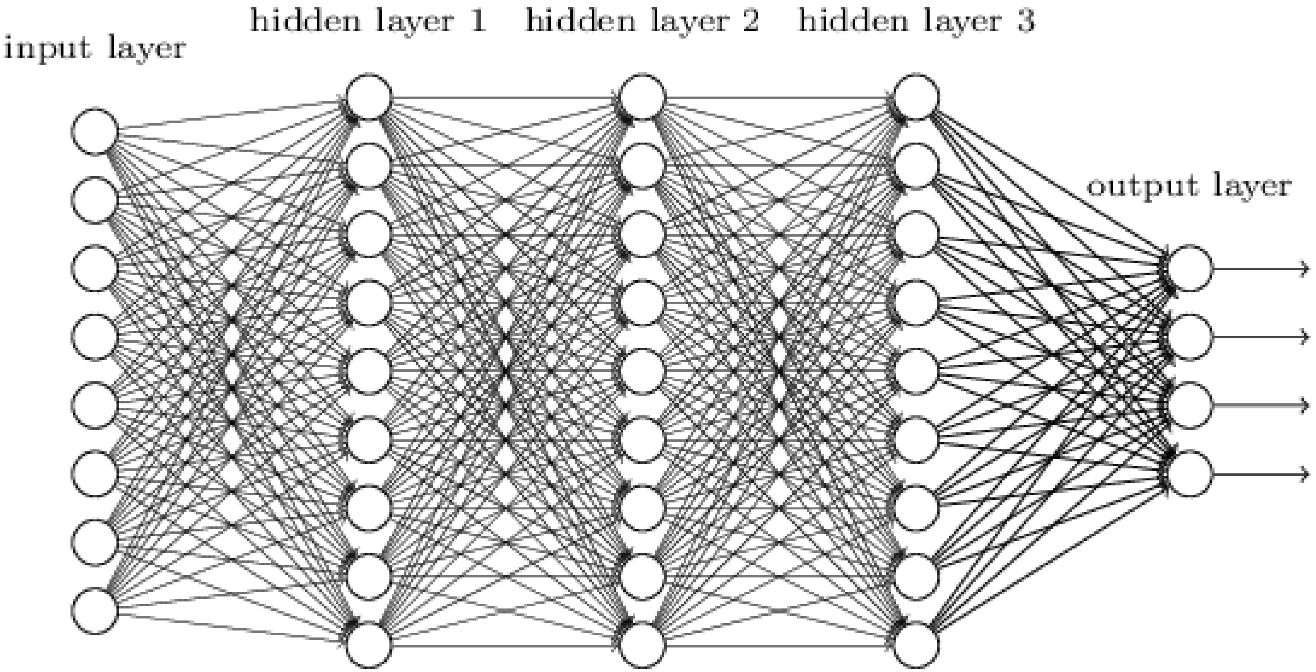


KAIST

Back propagation

How to train?

$\mathbf{x}$



$\mathbf{y}$

$$h_i^{(i+1)} = f(\textstyle\sum_j \boldsymbol{w}_{ij} h_j^{(i)} + b^{(i)})$$

One has to determine {**w**, b} so as to minimize the error, E($\mathbf{y}_{pred}$, $\mathbf{y}_{true}$)

Minsky (1969): no one on earth had found a viable way to train MLPs good enough to learn such simple functions (Winter of NN1 until 1986)

Loss functions

Loss function = Cost function = Error function

Regression

L2 loss function (or MSE) $= \frac{1}{2}\sum_i (y_i - h(x_i))^2$

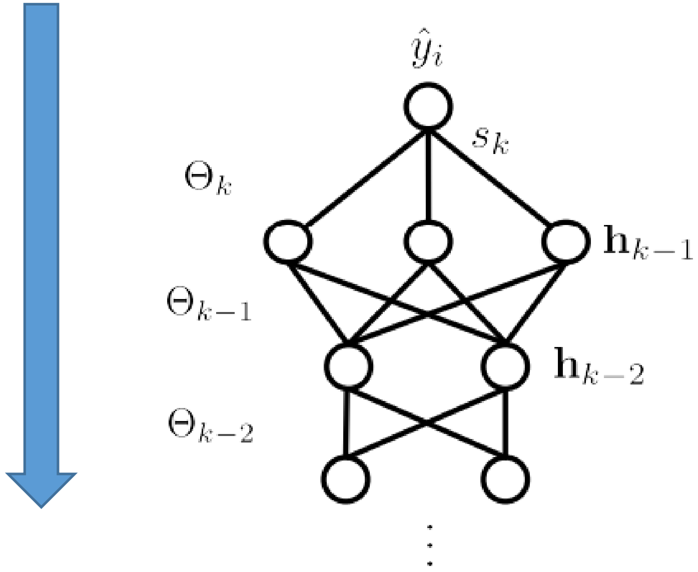
Classification

Cross entropy $= -\sum_i y_i \ln h(x_i)$

For binary classification, $L = -(y_i \ln h(x_i) + (1 - y_i)\ln(1 - h(x_i))$

KAIST

Gradient descent



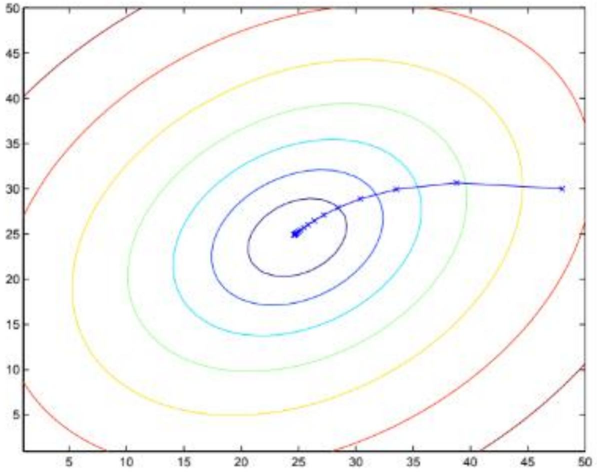
Back propagation

1974, 1982 by Paul Werbos,
1986 by Hinton, rediscovery

parameters　　loss function

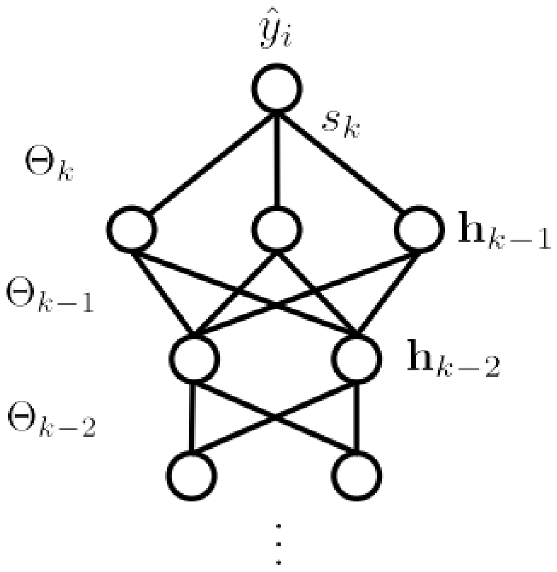
$$\Theta^{(t+1)} = \Theta^{(t)} - \gamma \underbrace{\nabla L\left(\Theta^{(t)}\right)}_{:= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\mathbf{x}_i, y_i; \Theta^{(t)})}$$

learning rate



Back propagation

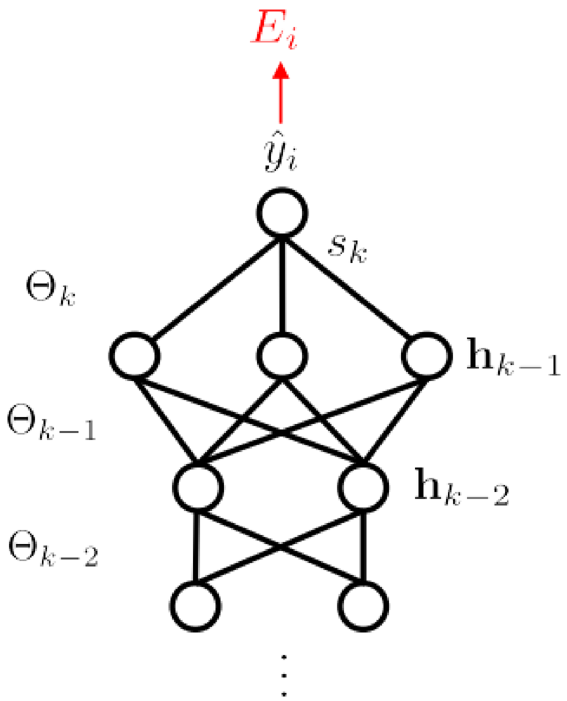
- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$



Back propagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- **Compute error** $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot,\cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$



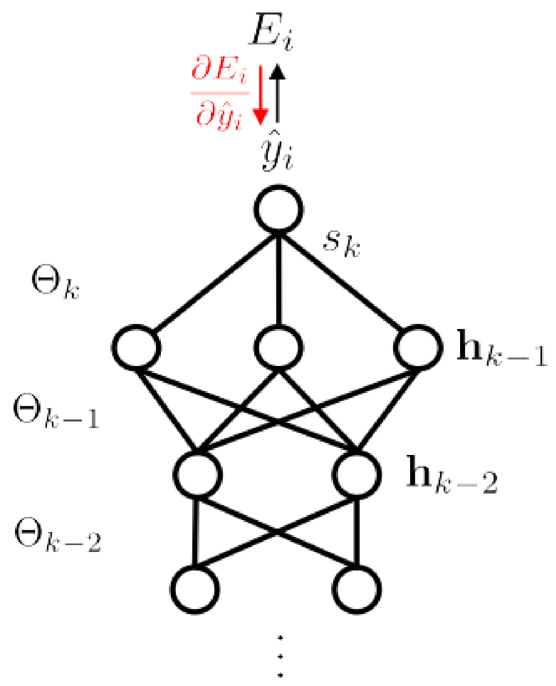
Back propagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot, \cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$

- **Compute derivative of E_i with respect to $\hat{y}_i$**

$$\frac{\partial E_i}{\partial \hat{y}_i} = \frac{\partial}{\partial \hat{y}_i}\frac{1}{2}(y_i - \hat{y}_i)^2 = -(y_i - \hat{y}_i)$$



KAIST

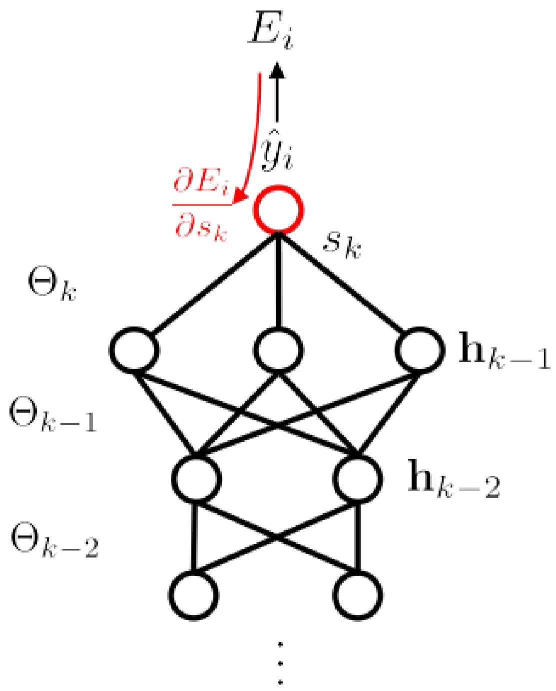
Back propagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot,\cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$

- **Compute derivative of E_i with respect to s_k**

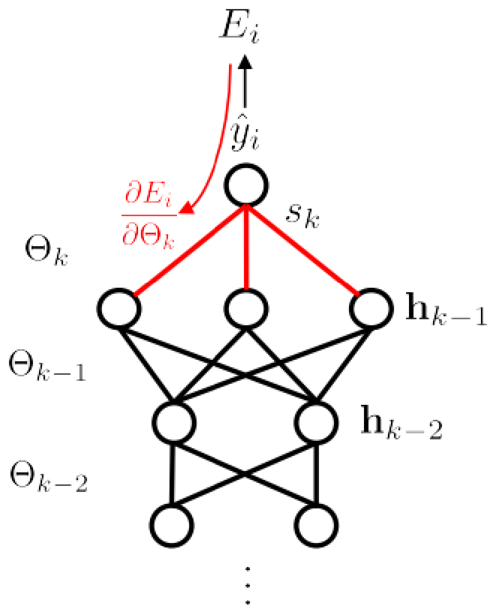
$$\frac{\partial E_i}{\partial s_k} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial}{\partial s_k}\sigma(s_k) = (\hat{y}_i - y_i)\sigma'(s_k)$$



Back propagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot,\cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$



- **Compute derivative of E_i with respect to Θ_k**

$$\frac{\partial E_i}{\partial \Theta_k} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k}\frac{\partial s_k}{\partial \Theta_k} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k}\frac{\partial}{\partial \Theta_k}(\Theta_k^\top \mathbf{h}_{k-1}) = (\hat{y}_i - y_i)\sigma'(s_k)\mathbf{h}_{k-1}$$

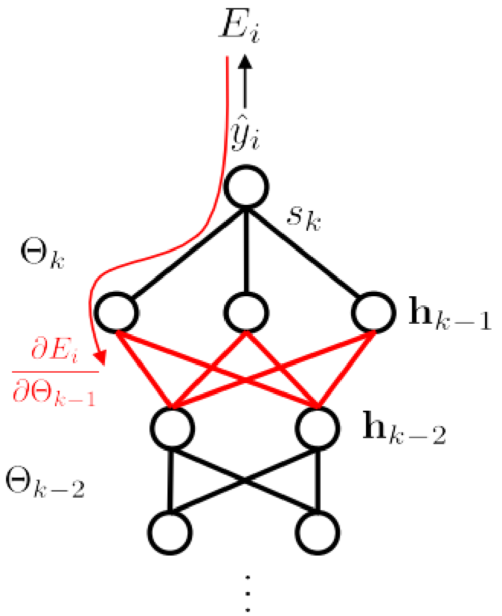
- Parameter update rule

$$\Theta_k \leftarrow \Theta_k - \underbrace{\gamma}_{\text{learning rate}} \frac{\partial E_i}{\partial \Theta_k}$$

Back propagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot,\cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$



- **Compute derivative of E_i with respect to Θ_{k-1}**

$$\frac{\partial E_i}{\partial \Theta_{k-1}} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k}\frac{\partial s_k}{\partial \mathbf{h}_{k-1}}\frac{\partial \mathbf{h}_{k-1}}{\partial \mathbf{s}_{k-1}}\frac{\partial \mathbf{s}_{k-1}}{\partial \Theta_{k-1}} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k}\frac{\partial s_k}{\partial \mathbf{h}_{k-1}}\frac{\partial \mathbf{h}_{k-1}}{\partial \mathbf{s}_{k-1}}\frac{\partial}{\partial \Theta_{k-1}}(\Theta_{k-1}^\top \mathbf{h}_{k-2})$$

- Parameter update rule

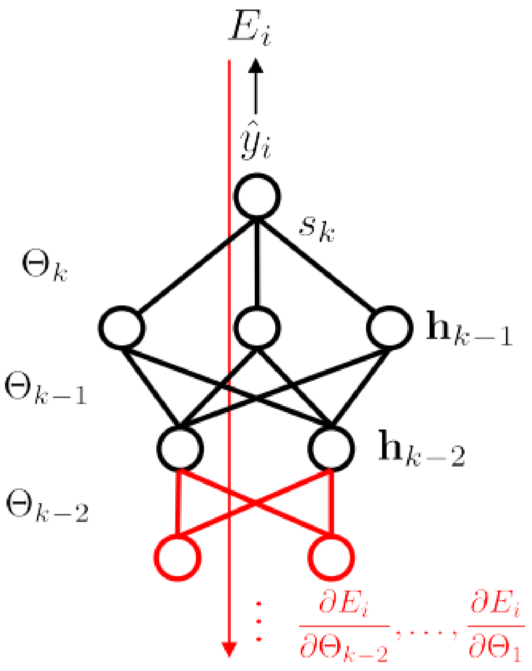
learning rate

$$\Theta_{k-1} \leftarrow \Theta_{k-1} - \gamma \frac{\partial E_i}{\partial \Theta_{k-1}}$$

Back propagation

- Consider the input $(\mathbf{x}_i, y_i)$
- Forward propagation to compute output $\hat{y}_i = f(\mathbf{x}_i; \Theta)$
- i^{th} layer intermediate output $s_i = \Theta_i^\top \mathbf{h}_{i-1}$
- Compute error $\ell(\hat{y}_i, y_i)$ (where $\ell(\cdot, \cdot)$ is MSE loss)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 := E_i$$



- Similarly, we can compute gradients with respect to $\Theta_{k-2}, \ldots, \Theta_1$
  - And update using the same update rule

Example: back propagation

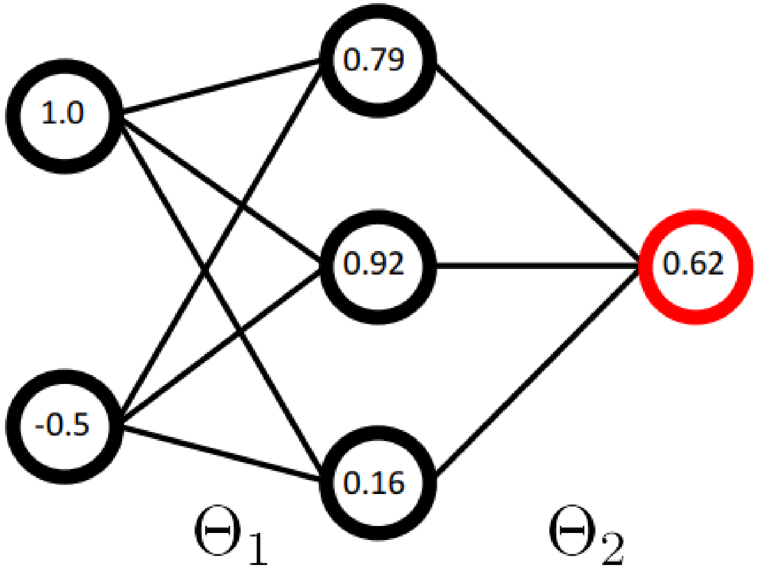
$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \quad y_i = (1.0) \quad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute the error $\ell(\hat{y}_i, y_i)$

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 = 0.072$$

- Compute $\frac{\partial E_i}{\partial \hat{y}_i}$

$$\frac{\partial E_i}{\partial \hat{y}_i} = (\hat{y}_i - y_i) = -0.38$$



Example: back propagation

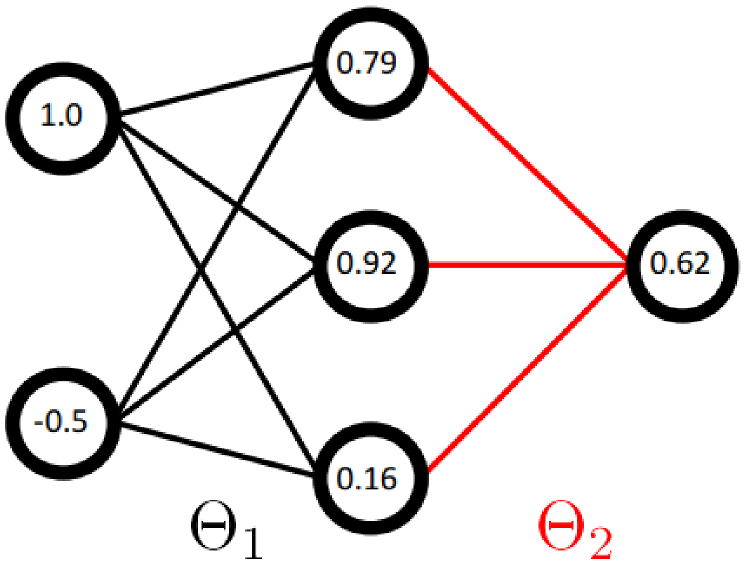
$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \quad y_i = (1.0) \quad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute $\frac{\partial E_i}{\partial \Theta_2}$

$$\frac{\partial E_i}{\partial \Theta_2} = (\hat{y}_i - y_i)\sigma'(s_2)\mathbf{h}_1 = \begin{pmatrix} 0.02 \\ -0.05 \\ -0.12 \end{pmatrix}$$

- Update Θ_2 with $\gamma = 1$

$$\Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix} - 1 \begin{pmatrix} 0.02 \\ -0.05 \\ -0.12 \end{pmatrix} = \begin{pmatrix} -0.22 \\ 0.55 \\ 1.42 \end{pmatrix}$$



Example: back propagation

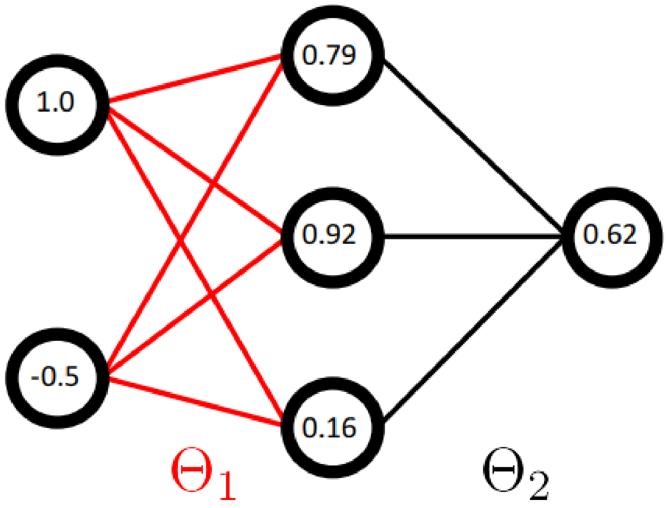
$$\mathbf{x}_i = \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix} \quad y_i = (1.0) \quad \Theta_1 = \begin{pmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{pmatrix} \quad \Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix}$$

- Compute $\dfrac{\partial E_i}{\partial \Theta_2}$

$$\frac{\partial E_i}{\partial \Theta_2} = (\hat{y}_i - y_i)\sigma'(s_2)\mathbf{h}_1 = \begin{pmatrix} 0.02 \\ -0.05 \\ -0.12 \end{pmatrix}$$

- Update Θ_2 with $\gamma = 1$

$$\Theta_2 = \begin{pmatrix} -0.2 \\ 0.5 \\ 1.3 \end{pmatrix} - 1 \begin{pmatrix} 0.02 \\ -0.05 \\ -0.12 \end{pmatrix} = \begin{pmatrix} -0.22 \\ 0.55 \\ 1.42 \end{pmatrix}$$

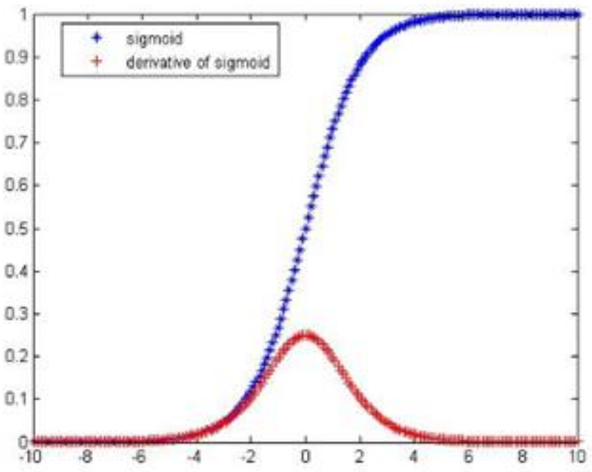


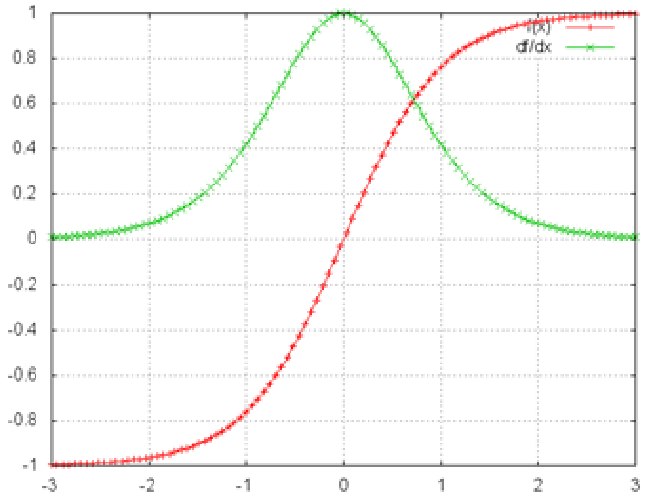
- Similarly, we can update Θ_1

Vanishing gradient

Vanishing gradient

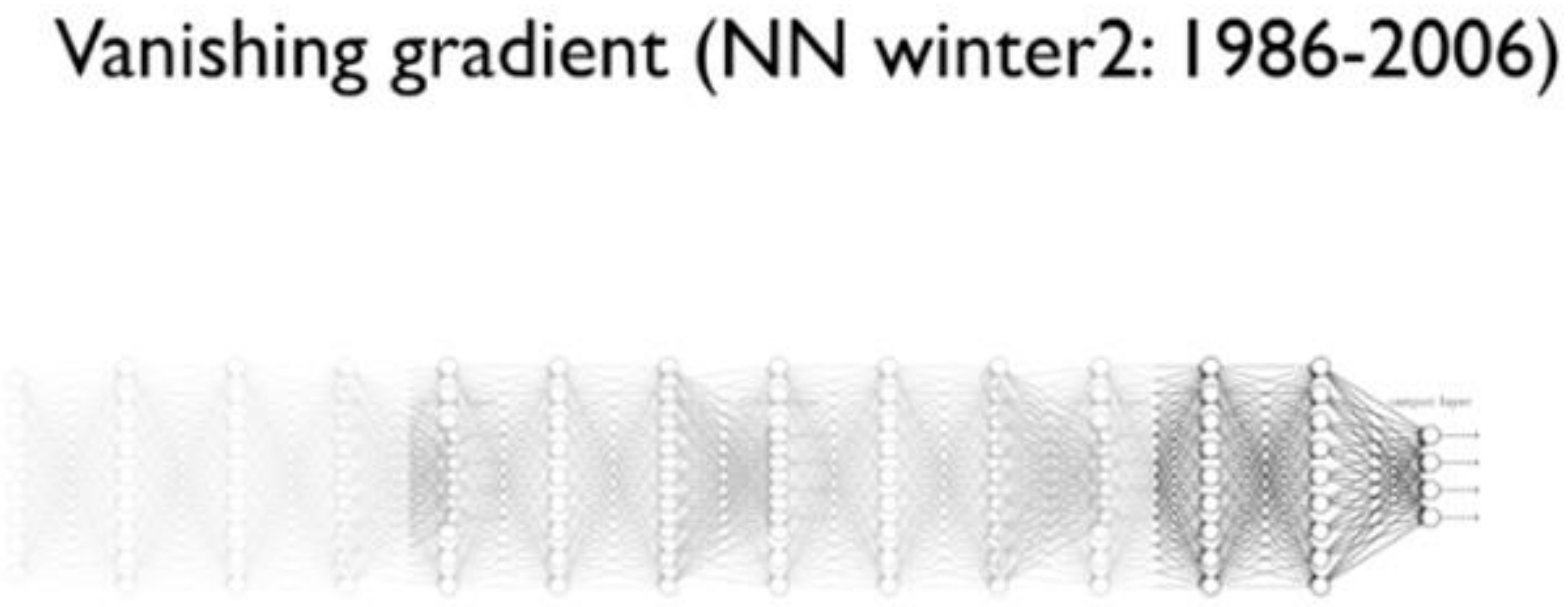
$$\frac{\partial E_i}{\partial \Theta_{k-1}} = \frac{\partial E_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_k}\frac{\partial s_k}{\partial \mathbf{h}_{k-1}}\frac{\partial \mathbf{h}_{k-1}}{\partial \mathbf{s}_{k-1}}\frac{\partial \mathbf{s}_{k-1}}{\partial \Theta_{k-1}} = \frac{\partial E_i}{\partial \hat{y}_i}\boxed{\frac{\partial \hat{y}_i}{\partial s_k}}\frac{\partial s_k}{\partial \mathbf{h}_{k-1}}\boxed{\frac{\partial \mathbf{h}_{k-1}}{\partial \mathbf{s}_{k-1}}}\frac{\partial}{\partial \Theta_{k-1}}(\Theta_{k-1}^\top \mathbf{h}_{k-2})$$





Successive multiplication of the derivatives of the activation function ➔ zero update value

Vanishing gradient

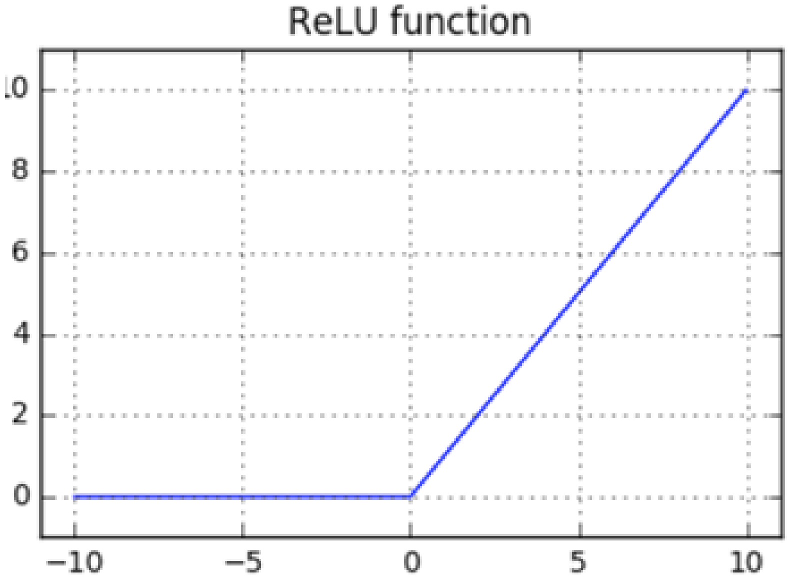


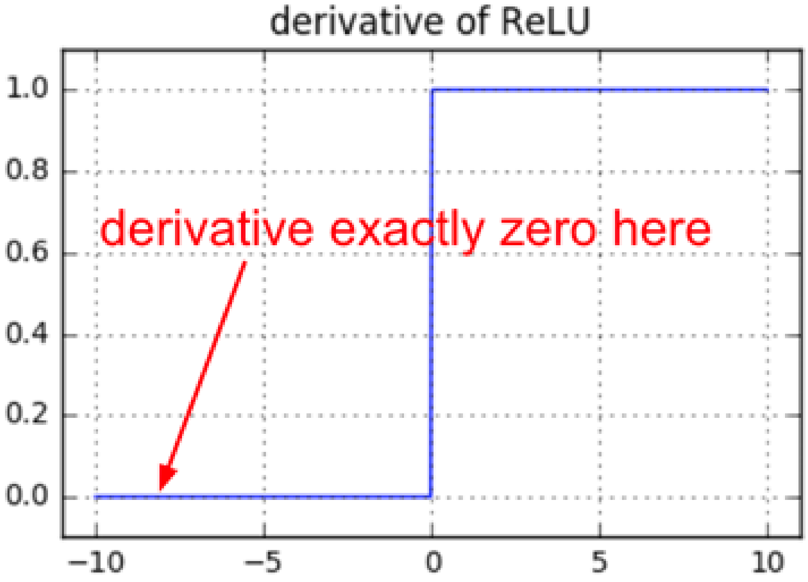
Winter of neural network 2, CIFAR

[Canadian Institute For Advanced Research]

ReLU

ReLU = Rectified Linear Unit





Nature **405**, 947 (2000).

Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit

Richard H. R. Hahnloser*‡, Rahul Sarpeshkar†§, Misha A. Mahowald*, Rodney J. Douglas* & H. Sebastian Seung†‡

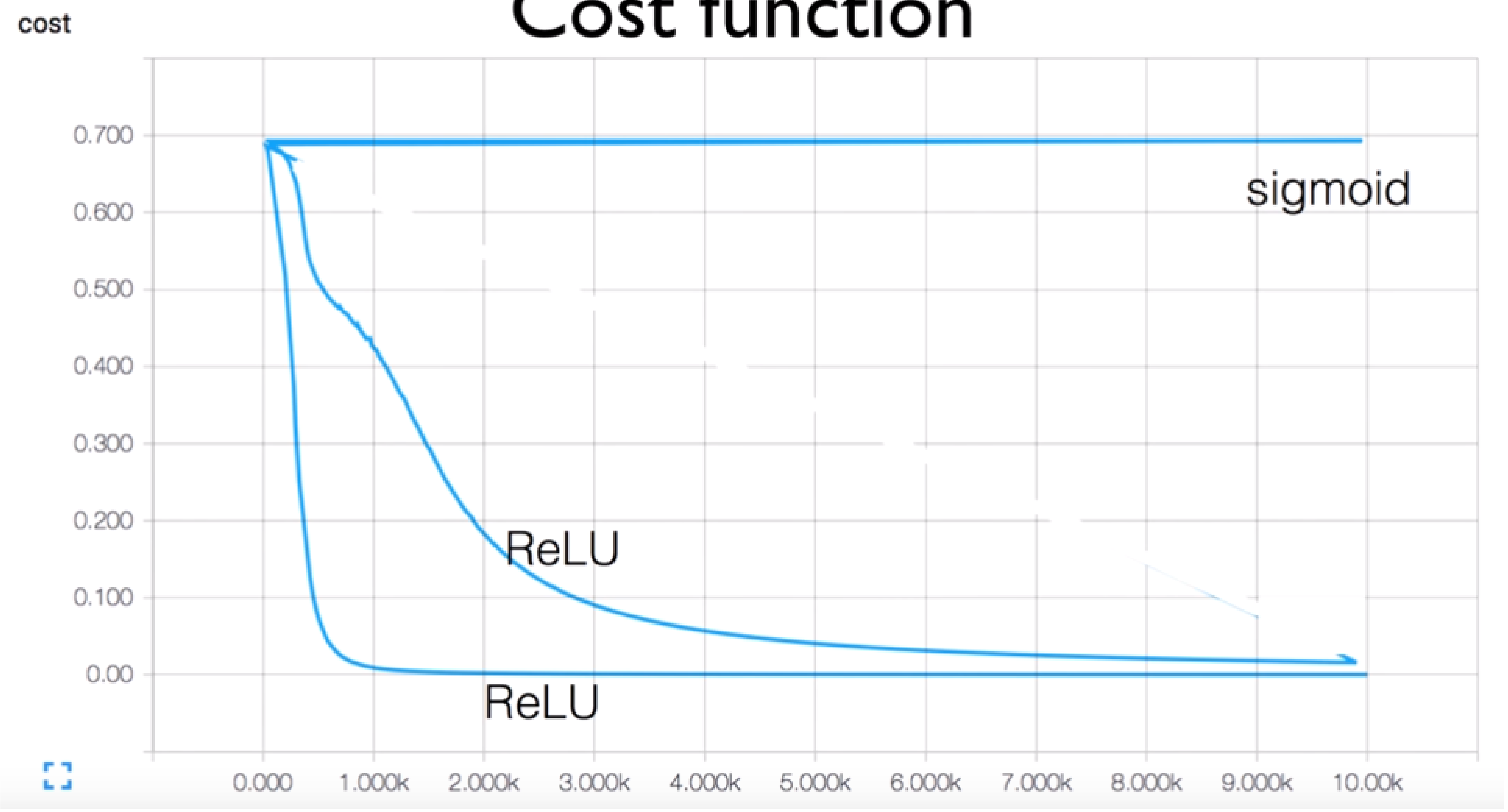
* *Institute of Neuroinformatics ETHZ/UNIZ, Winterthurerstrasse 190, 8057 Zürich, Switzerland*
† *Bell Laboratories, Murray Hill, New Jersey 07974, USA*
‡ *Department of Brain and Cognitive Sciences and § Department of Electrical Engineering and Computer Science, MIT, Cambridge, Massachusetts 02139, USA*

Nair and Hinton, ICML (2010)

Go deeper

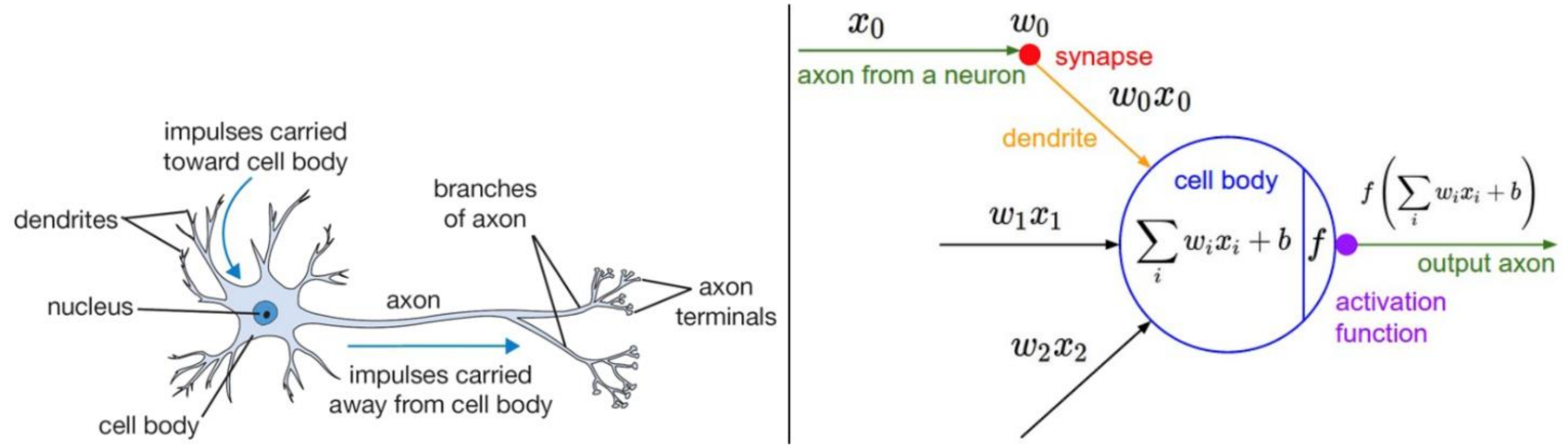
KAIST

Cost function



Summary

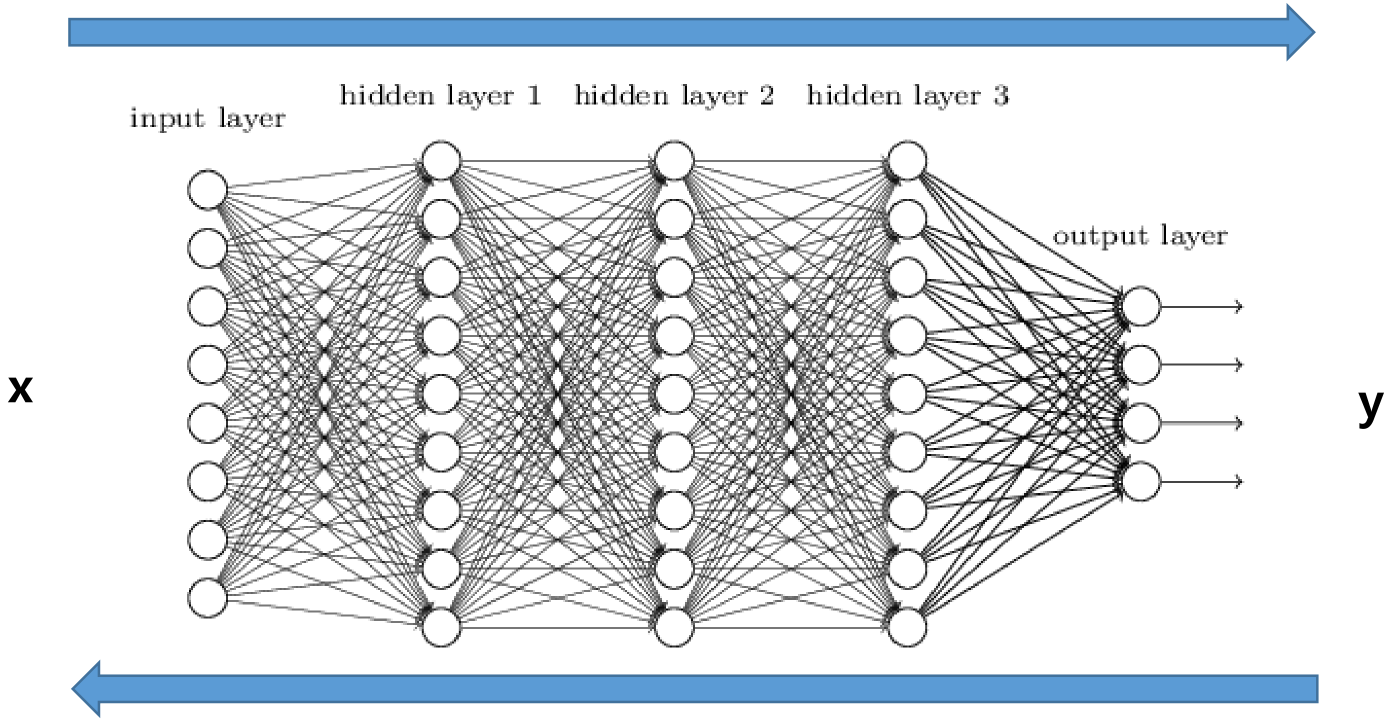
Neuron vs Perceptron: 1957 by Frank Rosenblatt



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Summary

Feed-forward MLP (or DNN)

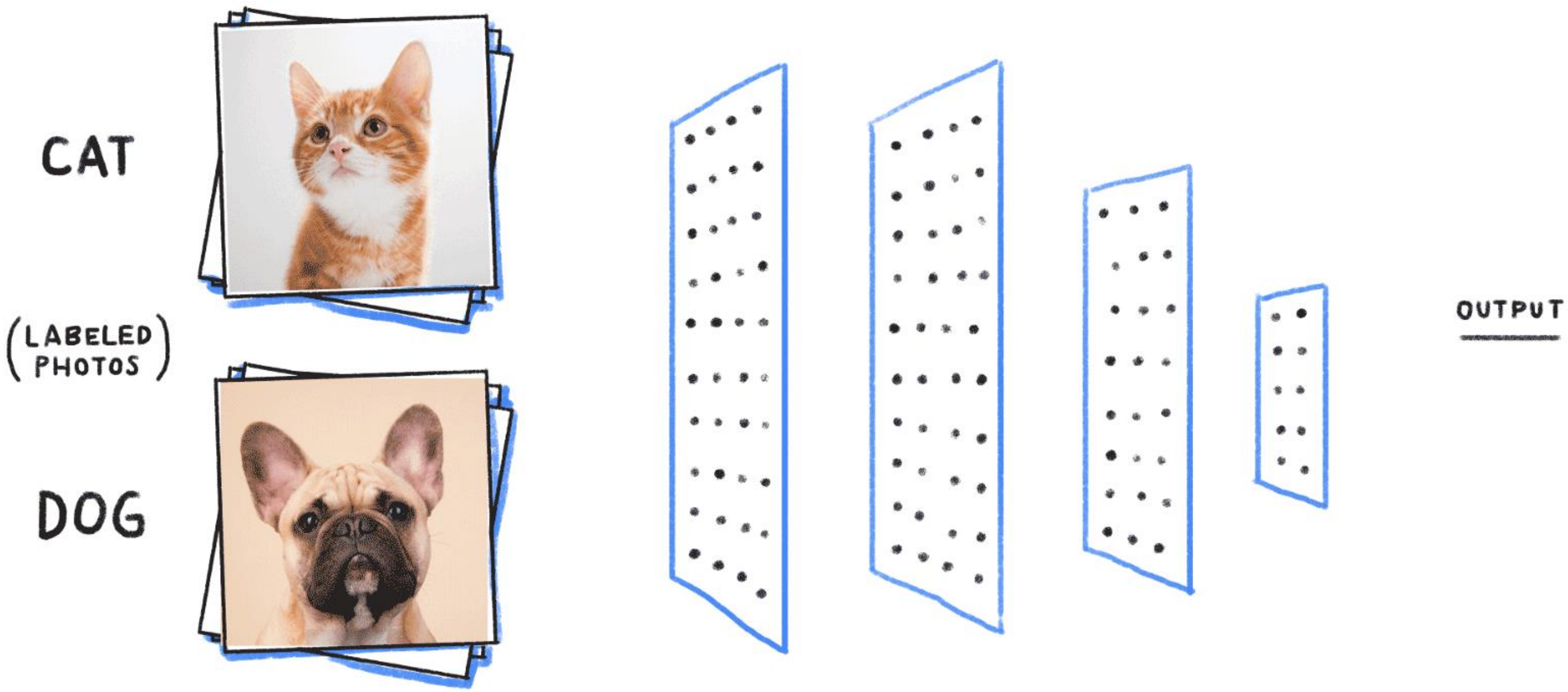


Back propagation (Hinton, 1986)

Vanishing gradient & ReLU (Hinton, 2010)

Perception

"easy-for-a-human, difficult-for-a-machine" tasks, often referred to as pattern recognition.



Unique signal pattern

New terms

- Neuron & synapse
- Perceptron
- Activation function: sigmoid, tanh, ReLU, etc
- Multilayer perceptron (MLP) = Deep neural network (DNN) = Artificial neural network (ANN)
- Nonlinearity
- Universal approximation
- Loss function = Cost function = Error function
- Feed forward
- Back propagation
- Vanishing gradient
- ReLU: rectified linear unit