CMC MSU Department of Algorithmic Languages
Samsung Moscow Research Center

Neural Networks for Natural Language Processing

Нейронные сети в задачах автоматической обработки текстов

Lecture 3. Introduction to Neural Networks

Arefyev Nikolay
*CMC MSU Department of Algorithmic Languages &
Samsung Moscow Research Center*

Inspired by the brain

Artificial NNs are insipred by the brain, but **are very different** (like airplanes are inspired by birds, but are different).

The brain consists of ~10^11 neurons, each connected to ~10^3-10^4 other neurons, a few are connected to receptors.

- Very high level of parallelism
- Neurons communicate using spikes - binary signals (charge)
- Learning algorithm?

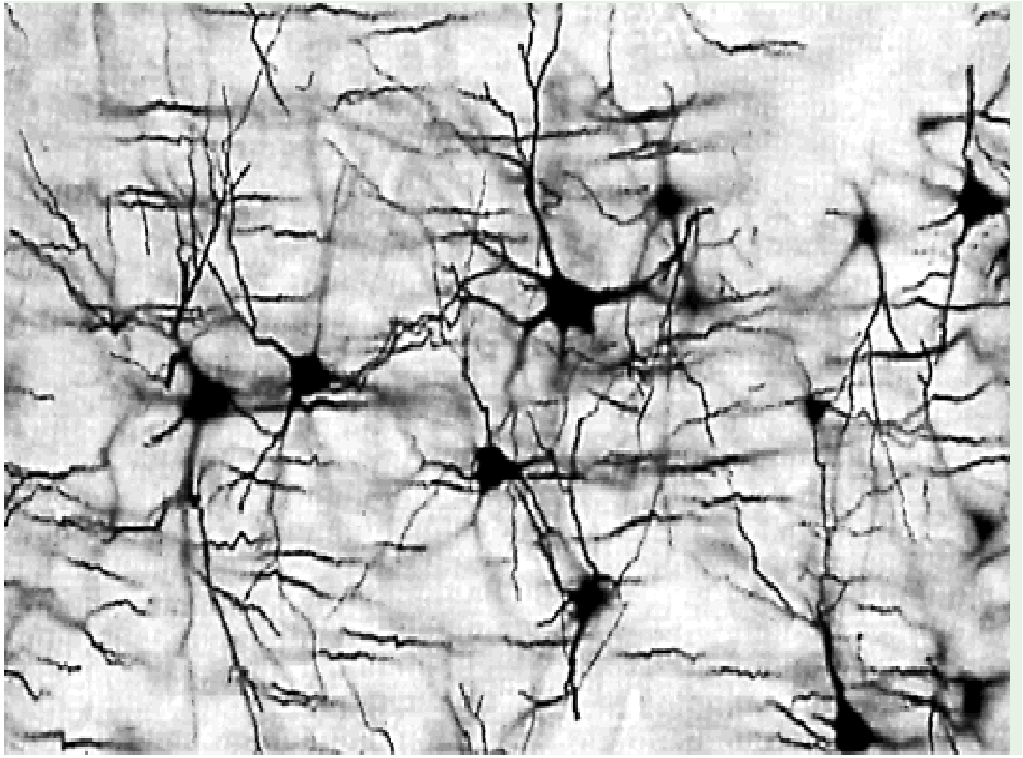


Figure from Yaser Abu-Mostafa. Machine learning video library (https://work.caltech.edu/library/)

Cortical neuron

Dendrites receive signals from other neurons (inputs), **axon** sends signals to other neurons (outputs). They are connected with special structures – **synapses** (~10^14).

- Axon hillock generates spike when enough charge is in
- 2 types of synapses: excitatory / inhibitory (increase / decrease probability of a spike)
- Synaptic weights (strength of influence on probability of a spike) are adapted to do computations useful for real-life problems

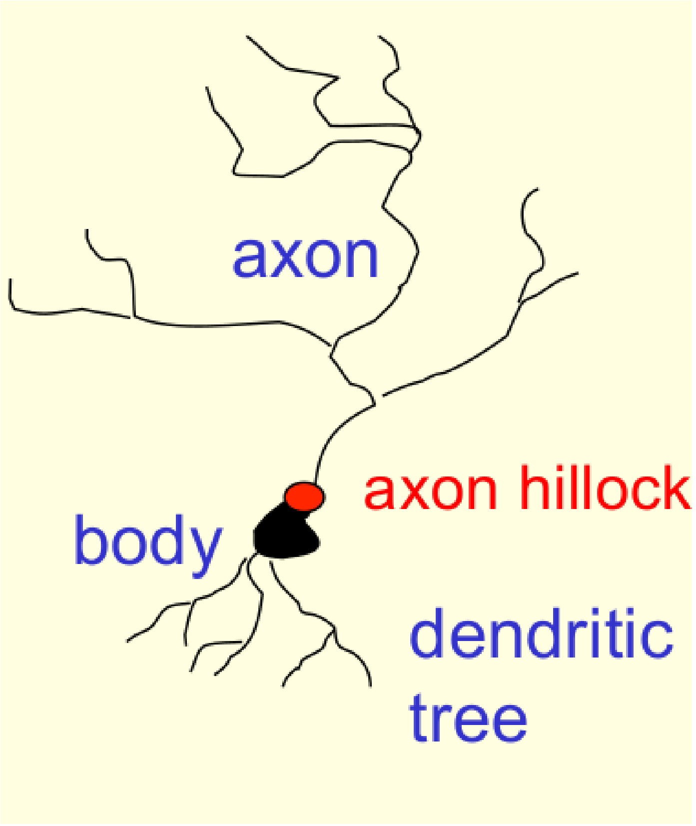


Figure from Hinton. Neural Networks for Machine Learning course, 2012 @ coursera.org

Artificial neurons

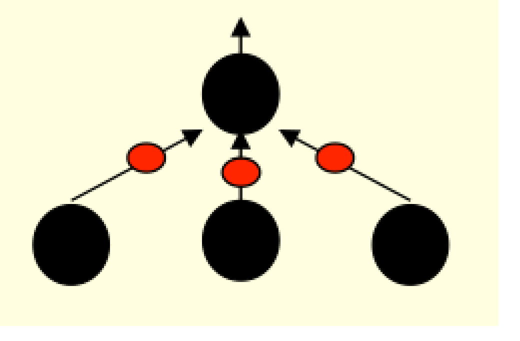
Most of them are wrong models of real neurons ...

- For instance, use **real values** to communicate

… but allow to apply effective algorithms to do useful things.

Artificial neurons first compute weighted sum of inputs (**preactivation**):

$$z = b + \sum_i x_i * w_i$$

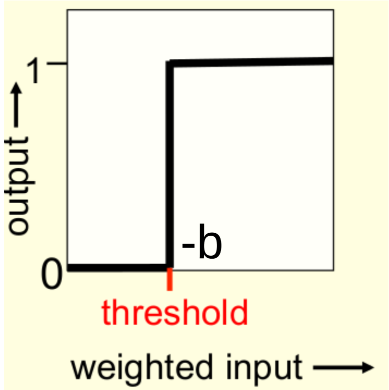


Figures from Hinton. Neural Networks for Machine Learning course, 2012 @ coursera.org

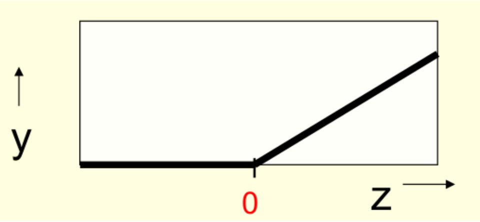
Artificial neuron activations

Then apply activation function:

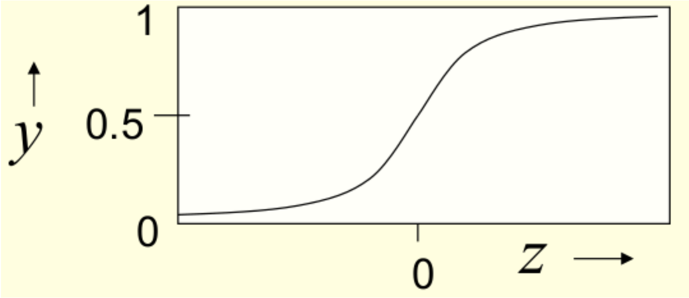
- Linear neuron: $y = z$
- Binary threshold neuron: $y = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$



- Rectified Linear neuron: $y = max(0, z)$



- Sigmoid neuron: $y = \frac{1}{1+e^{-z}}$



- Tanh neuron: $y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

Figures from Hinton. Neural Networks for Machine Learning course, 2012 @ coursera.org

A machine learning algorithm usually corresponds to a combination of the following 3 elements:

(either explicitly specified or implicit)

- ✓ the choice of a specific function family: F
  (often a parameterized family)
- ✓ a way to evaluate the quality of a function $f \in F$
  (typically using a cost (or loss) function L
  mesuring how wrongly f prédicts)
- ✓ a way to search for the «best» function $f \in F$
  (typically an optimization of function parameters to minimize the overall loss over the training set).

Pascal Vincent, Introduction to Machine Learning. Deep Learning Summer School, Montreal 2015

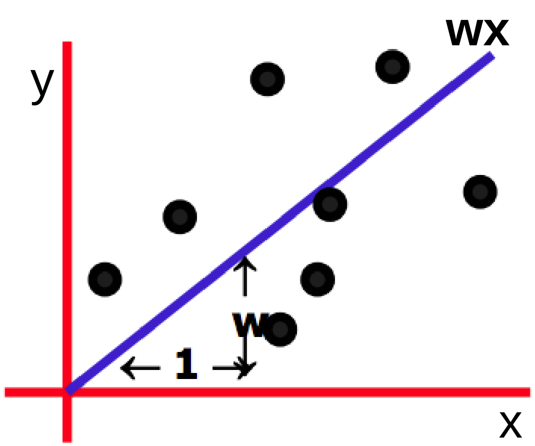
1D Linear Regression

Regression: for input vector ***x*** predict real value *y* (cf. Classification)

- Approximate dependence with linear function:

$$\hat{y} = h_w(x) = wx$$ → Hypothesis

- Learn *w* from data: Loss function? Optimization algorithm?



inputs	outputs
$x_1 = 1$	$y_1 = 1$
$x_2 = 3$	$y_2 = 2.2$
$x_3 = 2$	$y_3 = 2$
$x_4 = 1.5$	$y_4 = 1.9$
$x_5 = 4$	$y_5 = 3.1$

1D / Simple Regression: Regression with 1 input variable (feature)

Images from Andrew W. Moore tutorials (http://www.cs.cmu.edu/~awm/tutorials)

1D Linear Regression Loss

Loss (Cost, Objective) function:

- how well hypothesis *h(x)* approximates real function *y=f(x)*
- minimize loss on train set and hope it will generalize

For regression use sum of squared **residuals/errors:**

$$E = \sum_i \left(y_i - w x_i \right)^2$$

- Natural
- Justified by Maximum Likelihood principle under normal distribution of residuals ← **DERIVE!**

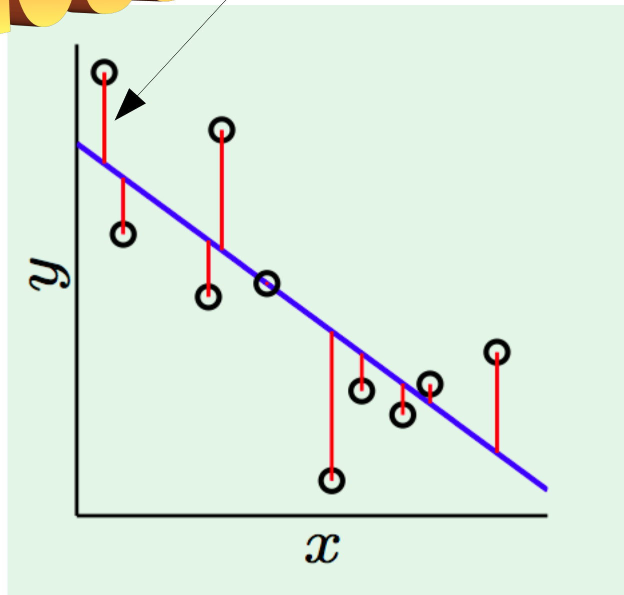


Image from Yaser Abu-Mostafa. Machine learning video library (https://work.caltech.edu/library/)

1D Linear Regression Optimization

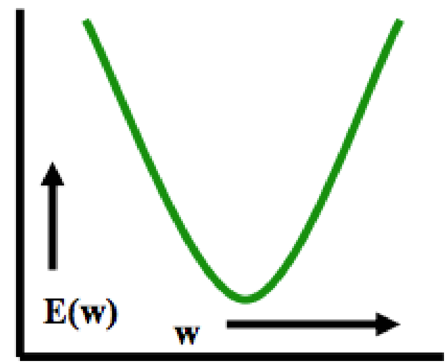
E can be minimized analytically:

– not the case for more sophisticated models

1. E(w) is convex => single optimum=global minimum
2. E`(w) = 0 ← **SOLVE**

$$w = \frac{\sum x_i y_i}{\sum x_i^{\ 2}}$$

Optimization



$$E = \sum_i (y_i - wx_i)^2$$

$$= \sum_i y_i^{\ 2} - \left(2 \sum x_i y_i\right) w + \left(\sum x_i^{\ 2}\right) w^2$$

Images from Andrew W. Moore tutorials (http://www.cs.cmu.edu/~awm/tutorials)

Multivariate Linear Regression

$$\hat{y} = h_w(x) = w^T x = \sum_{j=1}^{m} w_j * x_j$$

Hypothesis

$$E_{\text{in}}(w) = \frac{1}{N} \sum_{i=1}^{N} \left(w^T x_{\{i\}} - y_{\{i\}} \right)^2$$

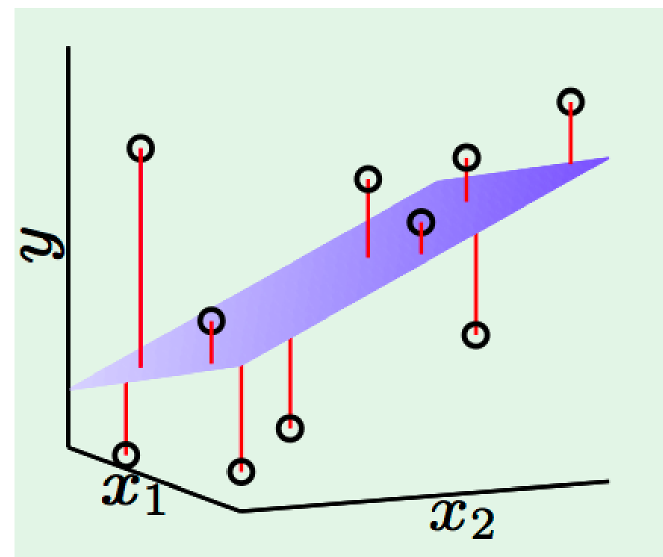
Loss

Mean squared error (MSE):
Multiplying by constant doesn't change argmin, but easier to compare

$$E_{\text{in}}(w) = \frac{1}{N} \| Xw - y \|^2$$

Matrix-vector product: much faster than loop! **always VECTORIZE computations**

$$\mathrm{X} = \begin{bmatrix} \text{—}\mathbf{x}_1{}^{\intercal}\text{—} \\ \text{—}\mathbf{x}_2{}^{\intercal}\text{—} \\ \vdots \\ \text{—}\mathbf{x}_N{}^{\intercal}\text{—} \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$



Images from Yaser Abu-Mostafa. Machine learning video library (https://work.caltech.edu/library/)

Multivariate Linear Regression Optimization

Loss is convex, single optimum=global minumum

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N}\mathrm{X}^{\intercal}(\mathrm{X}\mathbf{w} - \mathbf{y}) = \mathbf{0}$$

$$\mathrm{X}^{\intercal}\mathrm{X}\mathbf{w} = \mathrm{X}^{\intercal}\mathbf{y}$$

$$\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y} \quad \text{where} \quad \mathrm{X}^{\dagger} = (\mathrm{X}^{\intercal}\mathrm{X})^{-1}\mathrm{X}^{\intercal}$$

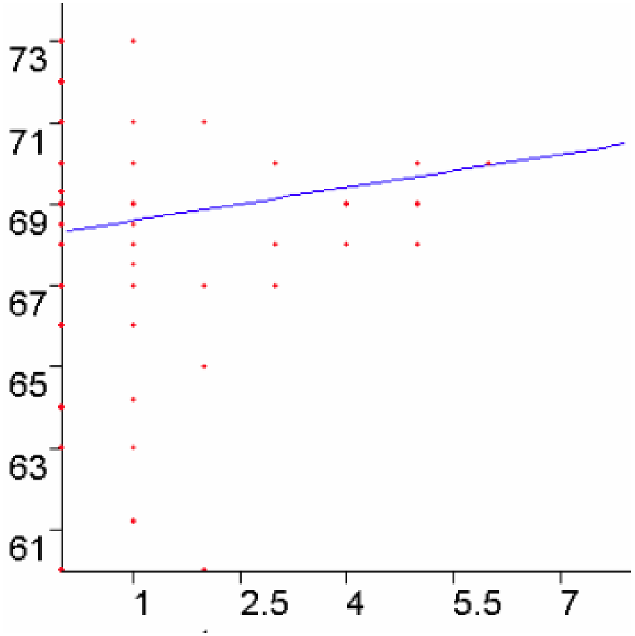
$\mathrm{X}^{\dagger}$ is the '**pseudo-inverse**' of X

Image from Yaser Abu-Mostafa. Machine learning video library (https://work.caltech.edu/library/)

Bias / intercept

What if f(0)!=0 ?

- Add bias / intercept: h(x)=wx+b
- Concatenatate "always 1" input to simplify implementation



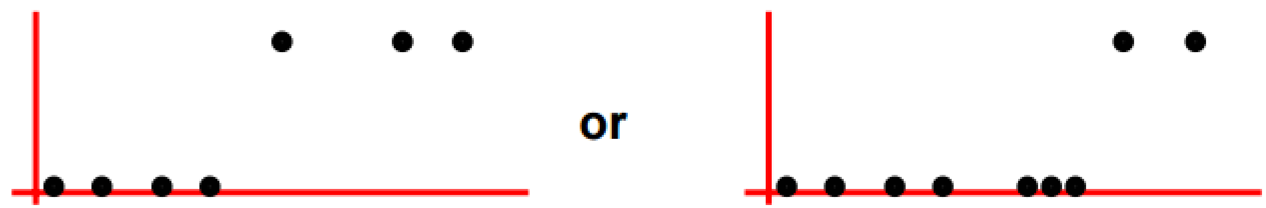
$$\hat{y}=h_w(x)=b+\sum_{j=1}^{m} w_j * x_j=[1;x]^T w \text{, } where\, w_0=b$$

X_0	X_1	X_2	Y
1	2	4	16
1	3	4	17
1	5	5	20

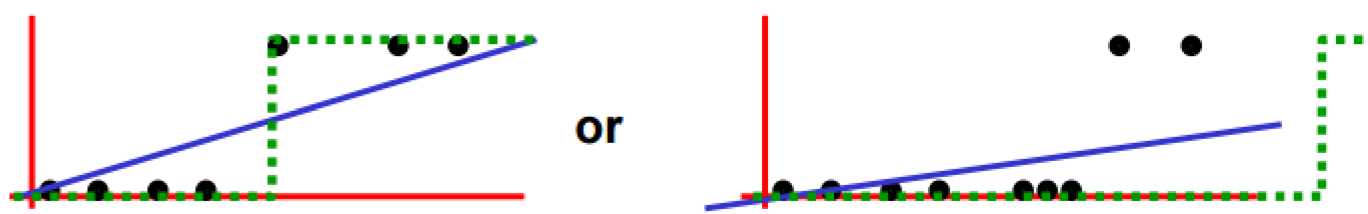
Images from Andrew W. Moore tutorials (http://www.cs.cmu.edu/~awm/tutorials)

Classification with linear regression

Classification as regression where y is 0 or 1



- Train linear regression
- Predict $\begin{cases} 1, \hat{y} \geq 0.5 \\ 0, \hat{y} < 0.5 \end{cases}$



Very unstable decision boundary!

- Least squares tries hard to minimize residuals for points which are already classified correctly!

Images from Andrew W. Moore tutorials (http://www.cs.cmu.edu/~awm/tutorials)

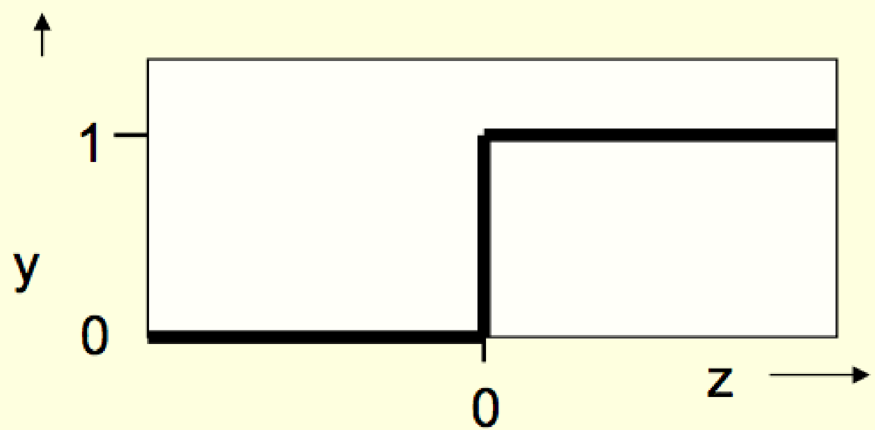
Classification with perceptron

Binary threshold neuron – early mathematical model a biological neuron

- Proposed by McCulloch & Pitts in 1943
- Biological neuron output was supposed to represent truth of some logical proposition

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } & z \geq 0 \\ 0 \text{ otherwise} & \end{cases}$$



Hypothesis

Figure from Hinton. Neural Networks for Machine Learning course, 2012 @ coursera.org

Classification with perceptron

In 1960s Frank Rosenblatt proposed a learning algorithm for perceptron:

Use -1 / +1 instead of 0 / 1 as labels

until convergence:

- pick next (xi, yi)
- If correctly classified: do nothing
- If incorrectly classified as 1: ***w := w – x***
- *If incorrectly classified as -1:* ***w := w + x***

Optimization

Pros: If dataset is linearly separable, this algorithm is guaranteed to converge to some solution ← PROOF in Hinton’s lecture2@coursera

Cons: very slow convergence, no guarantees for non linearly separable data

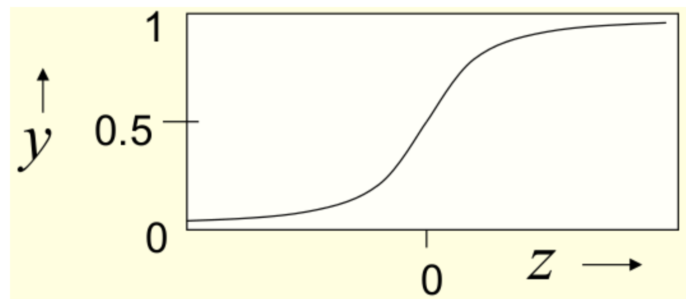
It was (incorrectly) shown that perceptrons can classify photos of tanks vs. trucks obscured in the forest

Photos of trucks were taken on a sunny day, photos of tanks – in a cloudy day, perceptron simply learned to estimate brightness.

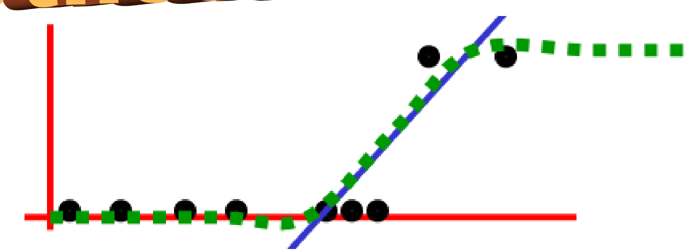
Logistic regression (LR)

- Use sigmoid neuron instead of linear

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



$$\hat{y} = h_w(x) = \sigma([1;x]^T w)$$ Hypothesis



  - more stable decision boundary
  - Outputs value in (0,1) – can be interpreted as probability of a positive class

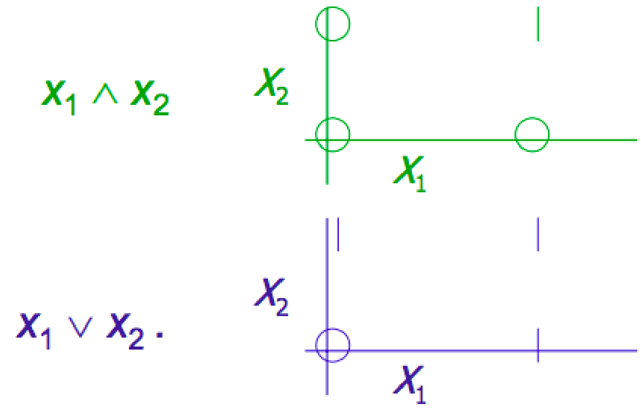
Logistic regression (LR) Loss

- Use binary cross-entropy loss (CE)
  - LR+CE is convex w.r.t. weights (unlike LR+MSE)
  - Justified by Maximum Likelihood ← DERIVE

$$E(w) = -\frac{1}{N}\sum_{i=1}^{N} y_{\{i\}} \log\left(h_w\left(x_{\{i\}}\right)\right) + \left(1 - y_{\{i\}}\right) \log\left(\left(1 - h_w\left(x_{\{i\}}\right)\right)\right)$$

- The same loss is commonly used for classification using NNs!

XOR problem



Linear models can learn:

- **AND**, **OR** functions ← FIND W?
- Any conjunction or disjunction of literals and their negations
  - x1 AND NOT x2 AND x3
  - x1 OR NOT x2 OR x3

Linear models cannot learn **XOR** function!

- Linearly non-separable
- People incorrectly concluded that NNs are not suited for real tasks
- NN with 1 hidden layer can solve XOR ← SHOW

Images from Andrew W. Moore tutorials (http://www.cs.cmu.edu/~awm/tutorials)

Modeling non-linear relations

- Linear models can be used for non-linear relations
  - but needs hand feature engineering
- Linear models = linearity w.r.t. weights, not features

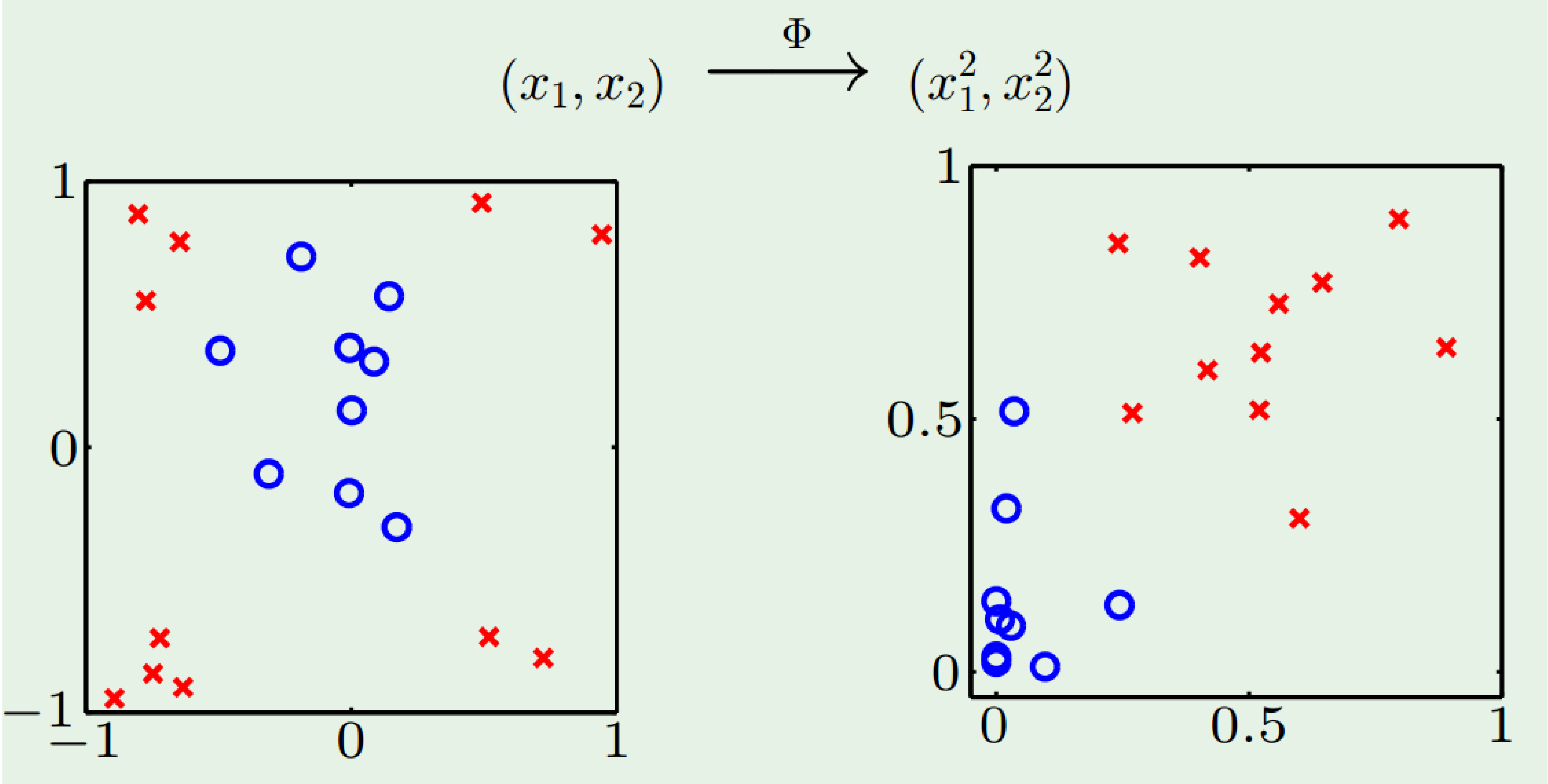


Image from Yaser Abu-Mostafa. Machine learning video library (https://work.caltech.edu/library/)

Feed-forward NN (FFNN)

FFNN – simply a composition of logistic regressions!

Even 1 hidden layer FFNN (given enough hidden units) can represent:

- any boolean function (exactly)

←BUILD 1 hidden layer NN with binary threshold neurons for XOR (manually)!

- any continuous function defined on compact subsets of R^n (approximately with any precision)

Appropriate weights exist, but no guarantees they will be learned!

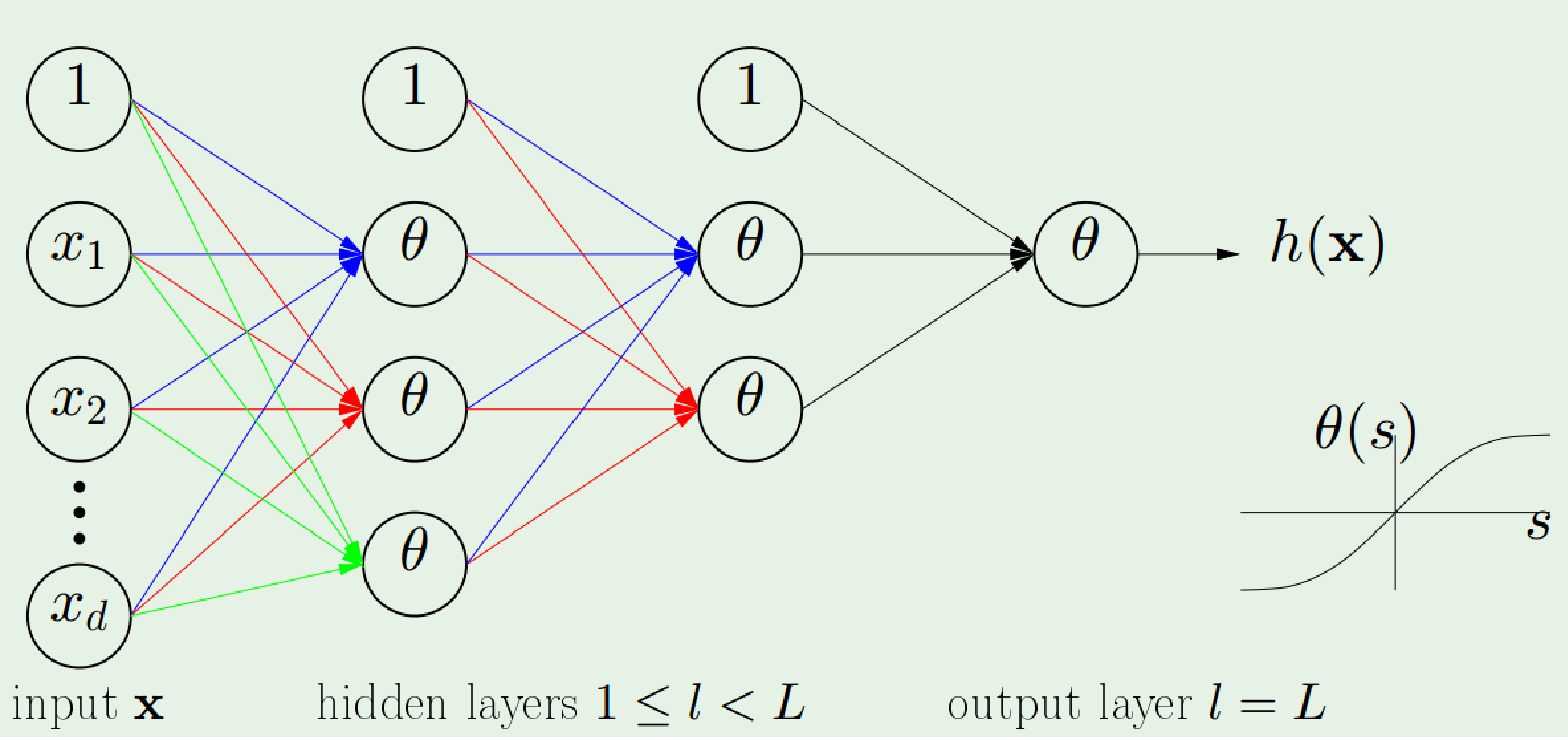
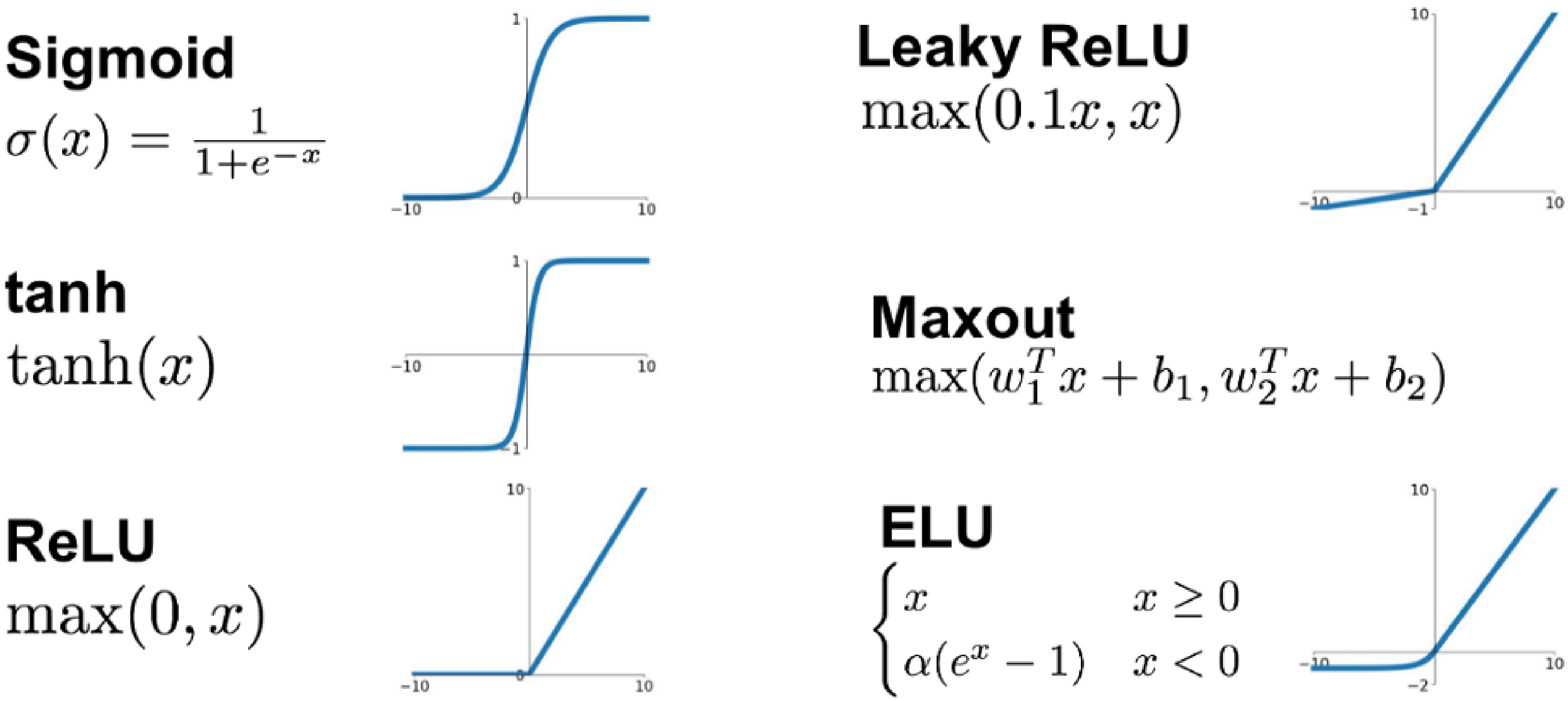


Image from Yaser Abu-Mostafa. Machine learning video library (https://work.caltech.edu/library/)

Popular activation functions

- Don’t use Sigmoid FFNN (not centered in 0)!
- Tanh is good default



Shruti Jadon. Introduction to Different Activation Functions for Deep Learning @ medium.com

LR/FFNN Optimization

- Optimized using iterative optimization algorithms (not analytically)
  - Most commonly used is Stochastic Gradient Descent (SGD) ← next lecture

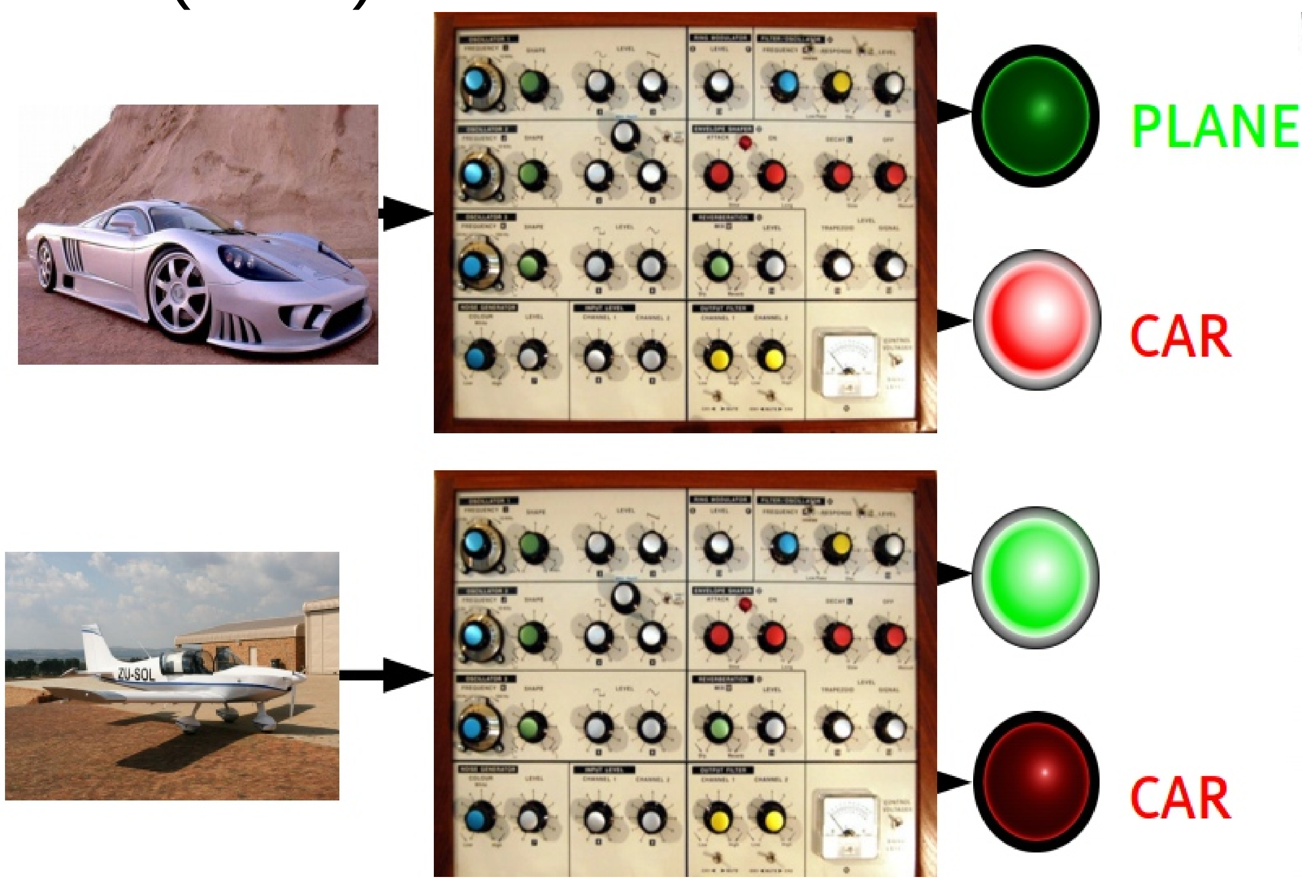


Image from Yann Le Cun, Deep Learning and the Future of AI, 2016