Uskoro počinjemo!?!

We are starting soon!?!

Inizieremo presto!?

Мы скоро начнется!?!

我們將很快開始

Zaczniemy wkrótce !?!

Empezaremos el trabajo pronto!?

Nous allons bientôt commencer

हम जल्द ही शुरू करेंगे

سنبدأ قريبا

Wir werden bald beginnen !?! ☺

VsiTe College of Information Technology

Workshop

Deep Learning

A Basic Tool of Artificial Intelligence

Vojislav KECMAN's Part

Fundamentals of Machine Learning and Neural Networks

Zagreb

May 10 – May 11, 2022

Contents

- Few general remarks on AI and ML
- Basics of Learning from Data
  - Linear model, Nonlinear Model, Neural Networks (NNs), Support Vector Machines (SVMs)
- Supervised, Semi-Supervised, Unsupervised Learning
- Learning Algorithms – **Gradient Method**
- Error Back Propagation
- Few words only on the very hot extension of NN

  the so called Deep Learning
- No conclusions – things are getting all of the following wilder, hotter and less predictable ☹

In the workshop, we're going to deal with two topics

- Artificial Intelligence

&

- Deep Learning

These are challenging, but also **ill-defined topics**.

Why ill-defined?

Because the **notion of intelligence is over-**, sorry, **under-defined**!

The definition of a (human) intelligence is controversial. Period!

Researcher	Quotation
Alfred Binet	Judgment, otherwise called "good sense," "practical sense," "initiative," the faculty of adapting one's self to circumstances ... auto-critique
David Wechsler	The aggregate or global capacity of the individual to act purposefully, to think rationally, and to deal effectively with his environment
Lloyd Humphreys	"...the resultant of the process of acquiring, storing in memory, retrieving, combining, comparing, and using in new contexts information and conceptual skills.
Howard Gardner	To my mind, a human intellectual competence must entail a set of skills of problem solving — enabling the individual to resolve genuine problems or difficulties that he or she encounters and, when appropriate, to create an effective product — and must also entail the potential for finding or creating problems — and thereby laying the groundwork for the acquisition of new knowledge.
Linda Gottfredson	The ability to deal with cognitive complexity.
Sternberg & Salter	Goal-directed adaptive behavior.
Reuven Feuerstein	The theory of Structural Cognitive Modifiability describes intelligence as "the unique propensity of human beings to change or modify the structure of their cognitive functioning to adapt to the changing demands of a life situation."
Legg & Hutter	A synthesis of 70+ definitions from psychology, philosophy, and AI researchers: "Intelligence measures an agent's ability to achieve goals in a wide range of environments,"[6] which has been mathematically formalized.
Alexander Wissner-Gross	$F = T \nabla S_T$

My Working Definition of Intelligence is:

- Intelligence is a human capacity to
- sense, memorize, reason, think abstractly, plan, solve problems, comprehend complex ideas, **learn from experience**, **join with others** when individual efforts don't suffice, **beneficially participate in society** and **hold to moral/ethical norms**.
- Excelling in all capacities leads to the top of intelligence.
- Failing in one, more or, God forbid, all of them is a straight path to the lower level of intelligence.

What is then the Artificial Intelligence?

The **Artificial Intelligence** is connected to the definition of **human intelligence** as follows:

- Whenever **one of** previous **capacities** is, or **more of them** are, **performed** within the **(wo)men** or machine **made artifacts**** such an **intelligence capacity** will be **labeled by the adjective 'artificial'**.

** In silicon, i.e., by computing chips, algebraic logical units, processors, computers, machines, cameras, sound recorders, MRI, …, etc.

Fine, but
what about the **Deep Learning**?

Well,
this is what the next two days will be about
☺

Artificial Intelligence (AI)
&
Machine Learning (ML)
are all of the following

hot, hot, hot
&
important, important, important

How comes? Why is that way?

December 2010, US President's Council of Advisors on Science and Technology submitted the REPORT TO THE PRESIDENT AND CONGRESS on DESIGNING A DIGITAL FUTURE

2011 NSF solicited application for BIG DATA

May 3, 2016, **USA Administration** announced the formation of a new NSTC's Subcommittee on **Machine Learning and Artificial intelligence**

2017 V. V. Putin, Whoever becomes the leader in AI will become the ruler of the world

2018, March, **E. Makron** announced French **€** 1.5 billions investment into AI

2018, 24 out of 27 EU countries announced cooperative efforts in building **European AI initiative** in order to **respond to Chinese** and **American AI challenges**

The things are even hotter in a real industrial/research world where

Giants are coming to the AI party 'spoiling' and redefining everything

- **IBM Watson**
- **Microsoft Oxford**
- **Google DeepMind**
- **Amazon Alexa**
- **Baidu Minwa**
- **Яндекс=Yandex**

All 6 companies are devoted to building powerful AI 'machines'

However, there are many more initiatives all over the planet Earth, one of them being

- **OpenAI** (Elon Musk) – "**counteract** large corporations who may gain **too much power by owning super-intelligence systems devoted to profits**, as well as **governments** which **may use AI to gain power and even oppress their citizenry**"

The giants' tools ready for the **AI** game are

Supercomputers:

thousands of processors & GPGPUs => **millions of cores**, huge memory, immense storage i.e., peta hard drives, ultrafast connections

and

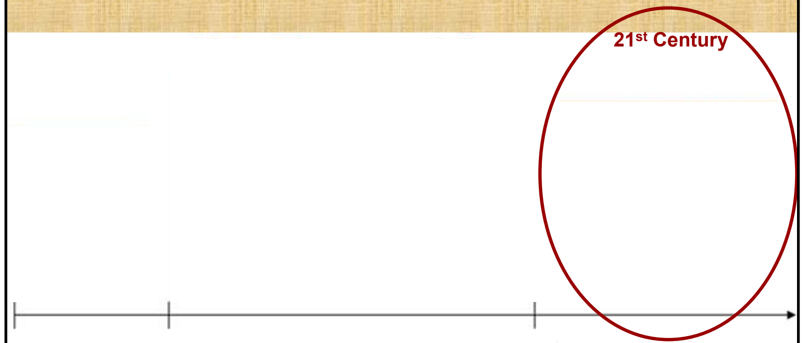
high end **intelligent software** with extreme capacities to **learn from data, (examples, cases, images, videos, reports, papers, web sites)** including abilities for natural language processing.

Now comes the strange, counterintuitive, but true & the most recent news about AI

- Amazon, Facebook, Google, IBM and Microsoft formed The **'Partnership on AI'**. **Hard to believe, competitors i.e., arch enemies, are uniting and joining forces?!?**
  WHY ???
- Their claim is: **The Partnership** is not a lobbying organization i.e., it's not aimed to lobby government but to
- "… conduct **research**, recommend **best practices**, and **publish research under an open license** in areas such as **ethics**, fairness and inclusivity; **transparency, privacy, and interoperability; collaboration between people and AI systems**; and the **trustworthiness, reliability and robustness of the technology** …"
- **if so**, **HUGE IF indeed**, it's a principal news pointing at the **importance & paramount impact AI** will have in future!

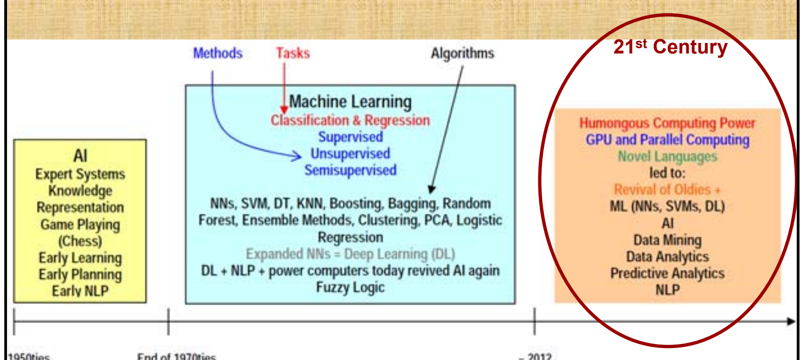
Let's Clean and Clarify the Vocabulary

- AI, ML, DL ? ? ?



Let's Clean and Clarify the Vocabulary

- AI, ML, DL ? ? ?



Let's now go back to our topics and let's present the basics of (supervised) learning

Historically, note that you may find different and classic names for the learning from data:

identification, estimation, data modeling, regression, classification, pattern recognition, statistical inference, function approximation, curve or surface fitting,…, etc…

Learning is the crucial constituent of intelligence.

How does the machine/computer learn?

Well, let's see !!!

Note! We are showing **just one way of learning**, present mostly in **NNs**, **SVMs** & **DL**. There are other similar, or very different, approaches.

Beginning of any **supervised** learning are data and our data is given as:

Columns are known as:

Features, Predictors, Explanatory Variables, Attributes, Inputs

Y-s are known as:

Labels, Desired values (when learning), Outputs (when applying)

1st data pair

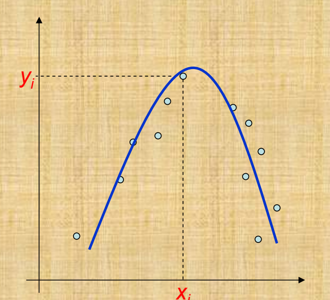
$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ x_{l1} & x_{l2} & \cdots & x_{ln} \end{bmatrix}, \quad \mathbf{Y}_{Class} = \begin{bmatrix} +1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}, \text{ or } \quad \mathbf{Y}_{Regress} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_l \end{bmatrix}$$

l^{st} data pair

To better understand the notions of **x**, y and $o = f(\mathbf{x})$ consider this problem

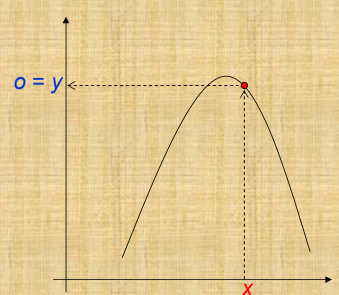
Training

Known: pairs ($\mathbf{x}_i$, y_i),
Assume: $o = x^2w_2 + xw_1 + 1w_0$
Find: $\mathbf{w} = [\ w_1\ \ w_2\ \ w_0\ ]^t$



Test i.e. Application

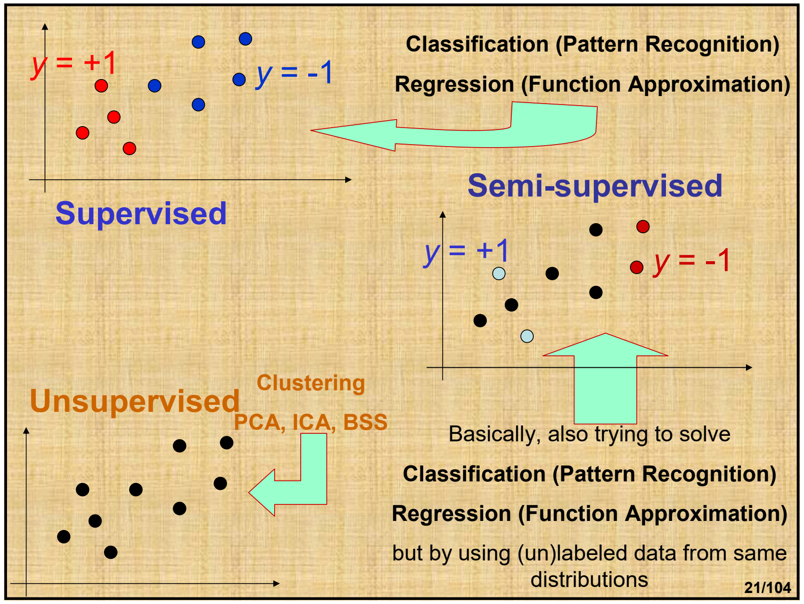
Known: input (**x**), and **w**
Find: $o = y_a = x^2w_2 + xw_1 + 1w_0$



Let's first set the stage: there are three (3) machine learning (ML) settings

- **Supervised** (pairs $\boldsymbol{x}_i$, y_i are given for **all** data pairs, where $\boldsymbol{x}_i$ are the values of the independent variables, features, inputs, attributes and y_i are class labels)
- **Semi-supervised** (pairs $\boldsymbol{x}_i$, y_i are given for **just a fraction** of data pairs)
- **Unsupervised** (only inputs $\boldsymbol{x}_i$, are given and no single label y_i is known)

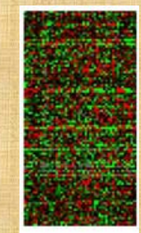
Today, we deal only with **SUPERVISED** ML problems!

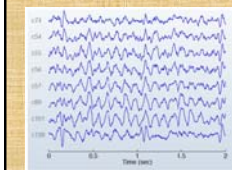


Data sets variety

X and Y, or X only

	Legs	Wings	Fur	Feathers
cat	4	no	yes	no
crow	2	yes	no	yes
frog	4	no	no	no
bat	4	yes	yes	no
barstool	3	no	no	no

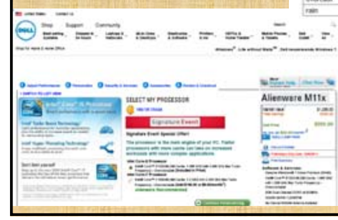


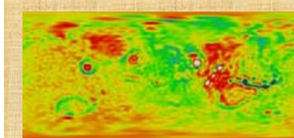


Play golf dataset

Independent variables				Dep. var
OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	FALSE	Don't Play
sunny	80	90	TRUE	Don't Play
overcast	83	78	FALSE	Play
rain	70	96	FALSE	Play
rain	68	80	FALSE	Play
rain	65	70	TRUE	Don't Play
overcast	64	65	TRUE	Play
sunny	72	95	FALSE	Don't Play
sunny	69	70	FALSE	Play
rain	75	80	FALSE	Play
sunny	75	70	TRUE	Play
overcast	72	90	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	80	TRUE	Don't Play

1	-0.66242	-0.03596
2	101.07	100.04
2	100.28	99.772
2	102.26	99.244
2	100.27	99.729
1	-1.2924	0.31328
1	1.4643	0.63647
2	100.24	99.294





Mars magnetic field

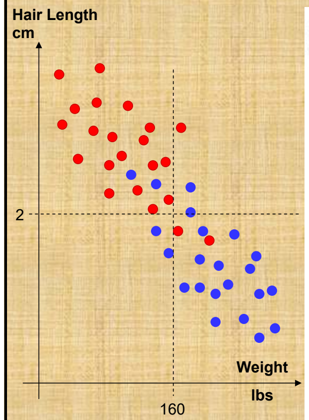
Learning

First, let's present a good old method, born in statistics, known as **DECISION TREE** and then we'll introduce a classic learning method which ends by using **PSEUDOINVERSE** of a data matrix **X** or (after NL mapping in a new space) of matrix **G.**

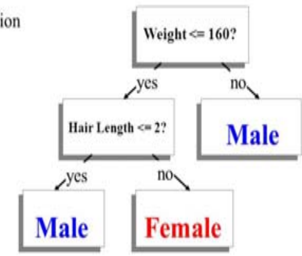
Decision Trees

- A decision tree represents a *decision-making process*.
  - Each **possible "decision point"** or situation is represented by **a node**.
  - Each **possible choice** that could be made at that decision point is represented by **an edge** to a child node.

Example of DT 1 – Gender Recognition



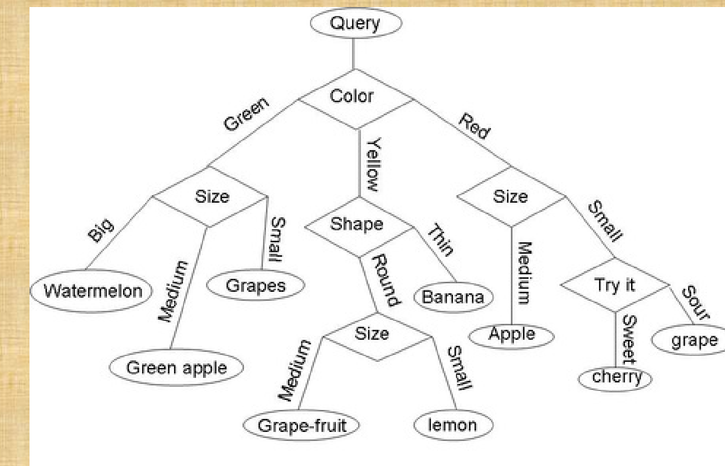
It is trivial to convert Decision Trees to rules…



Rules to Classify Males/Females

If *Weight* **greater than** 160, classify as **Male**
Elseif *Hair Length* **less than or equal** to 2, classify as **Male**
Else classify as **Female**

Example of DT 2 – Fruit Recognition

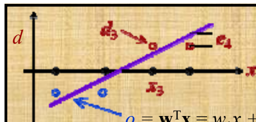


Classic Linear Classification Model Produced by the Use of a

PSEUDOINVERSE

of a

DATA MATRIX X



$o = \mathbf{w}^T\mathbf{x} = w_1x + w_2$

w = ???

$d = \mathbf{w}^T\mathbf{x} + e$

$e_p = d_p - o_p = d_p - \mathbf{w}^T\mathbf{x}_p$

P is the cardinality of dataset and p stands for pattern i.e., data sample

$$R = \sum_{1}^{P} e_p^2$$

R** stands for **RISK** which will be composed of **LOSS** (error, cost, norm, merit, fitness) and **REGULARIZER**. Here, we use only **LOSS.

$$R = \sum_{1}^{P}(\mathbf{w}^T\mathbf{x}_p - d_p)(\mathbf{w}^T\mathbf{x}_p - d_p) = \mathbf{w}^T(\sum_{1}^{P}(\mathbf{x}_p\mathbf{x}_p^T)\mathbf{w} - 2\mathbf{w}^T\sum_{1}^{P} d_p\mathbf{x}_p + \sum_{1}^{P} d_p^2$$

$$\mathbf{X} = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_P],\ \mathbf{D} = [d_1\ d_2\ \ldots\ d_P]^T$$

Remind, we **know** *x* and *d*, we **assume** linear model here and **learning is finding** the weights w_i

$$R = \mathbf{w}^T[\mathbf{X}^T\mathbf{X}]\mathbf{w} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{D} + \mathbf{D}^T\mathbf{D}$$

where, $\mathbf{X} \in \Re^{P,n+1}$, $\mathbf{D} \in \Re^{P,1}$

$$\frac{\partial R}{\partial \mathbf{w}} = 2\,(\mathbf{X}^T\mathbf{X})\mathbf{w} - 2\mathbf{X}^T\mathbf{D} = \mathbf{0}$$

$(\mathbf{X}^T\mathbf{X})\mathbf{w} = \mathbf{X}^T\mathbf{D}$ this is the famous **normal equation**

$$\mathbf{w}^* = (\mathbf{X}\,\mathbf{X}^T)^{-1}\,\mathbf{X}\,\mathbf{D} = \mathbf{X}^+\mathbf{D}$$

$\mathbf{X}^+$ is known as **PSEUDOINVERSE**

Remember**, whenever you use pseudoinverse you are minimizing sum of error squares!!!**

Let's solve an 1-dimensional problem, *m* = 1

Training pairs (x_i, y_i) are given:

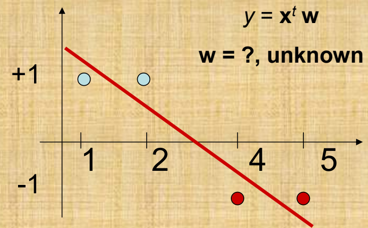
x=[1 2 4 5]', **y**=[1 1 -1 -1]'

Our ASSUMED model is

$y = \mathbf{x}^t \mathbf{w}$

w = ?, unknown

We are after decision function. Assume linear one -> then we are after **weights** (intercept and slope)



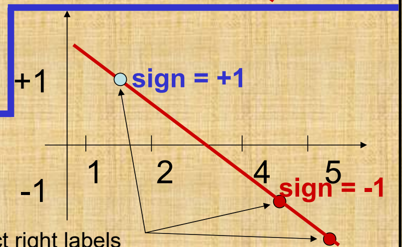
$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 4 & 1 \\ 5 & 1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \quad \mathbf{w} = \mathbf{X}^* \mathbf{y}, \quad \mathbf{w} = \begin{bmatrix} -0.6 \\ 1.8 \end{bmatrix}$$

After, the training given new (test, application) data

$y_{itest} = \mathbf{w}^t \mathbf{x}_{itest}$, say for

x=[1.5 4.5 5.5], our model will predict right labels

y = sign([0.9 -0.9 -1.5]^T)



This was an 1-dimensional case. Modern application use very high dimensional data. Let's show you how the dimensionality increases very fast even in a very simple problem of

GENDER RECOGNITION

Gender recognition problem: Female or Male?

L

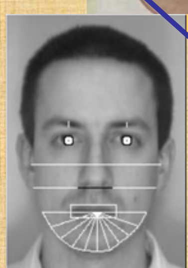
R

F or M?

There must be something in the geometry of our faces. Here, 18 input variables, features, were chosen!

M or F?

A problem from **Brunelli & Poggio**, 1993.



Nr.	Feature
1	pupil to nose vertical distance
2	pupil to mouth vertical distance
3	pupil to chin vertical distance
4	nose width
5	mouth width
6	zygomatic breadth
7	bigonial breadth
8-13	chin radii
14	mouth height
15	upper lip thickness
16	lower lip thickness
17	pupil to eyebrow separation
18	eyebrow thickness

Note!

Today, **18-dimensions is a very low dimensionality.**

Not only this – even bigger problem is that the data set is usually sparse (little, or very little data) in high dimensional data!!!

Data **size,** its **dimensionality** and often its **sparseness** & **complexity** are chasing us indeed

and, we feel their heavy breath down our neck.

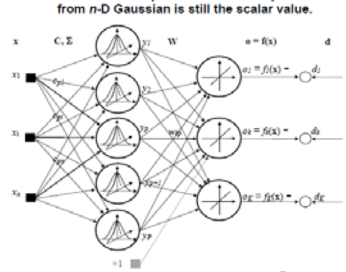
Before, going to nonlinear models let's introduce the idea of nonlinear mapping of a data matrix **X** into a matrix **G** and then a usage of a pseudoinversion. Recently this procedure was reinvented and 'commercially' dubbed the

Extreme Learning Machine (**ELM**)

Well, here it is

The very **idea** is both **simple** and **old**

Finally, the structure of the RBF network for an *n* dimensional input and *K*-dim output looks as shown below. Note that the learning problem is same as for 1-D input because the output from *n*-D Gaussian is still the scalar value.



The architecture of a regularization (i.e., RBFs) network for $\Re^n \rightarrow \Re^K$ mapping. The *n*-dimensional input vector **x** is shown component-wise and the *n*-dimensional Gaussians basis or activation functions are shown as two-dimensional Gaussians bells.

Suppose we have **1000 data pairs**, with **12-dimensional inputs** and **10 dimensional outputs.** We have selected **333 centers** and we have placed them into the input space by some *clever* technique. The RBF model and the matrices involved will be as follow:

$$\mathbf{G}_{train}\,\mathbf{W} = \mathbf{D}$$
$$\mathbf{W} = \mathbf{G}^{+}\mathbf{D}$$
$$\mathbf{y}_a = \mathbf{G}_{test}\,\mathbf{W}$$

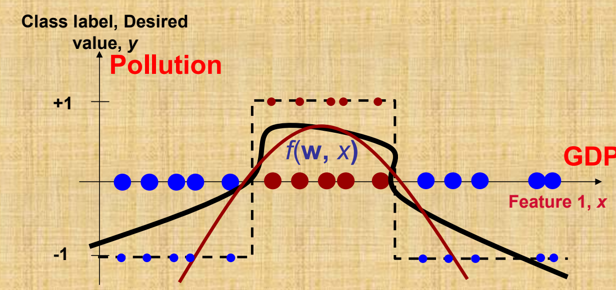
$$\mathbf{G}(1000, 333), \mathbf{W}(333, 10), \mathbf{D}(1000, 10)$$

Note that **G** and **D** are known and, after a pseudoinversion step, **W** can be calculated. Having **W**, for any 12-*dim* input vector x we can calculate the corresponding row of **G matrix** and find the output vector y from the RBF NN by the last equation above!!!

Well, until now it was simple, and many of you know linear models, but how to proceed if the solution must be a

NONLINEAR MODEL?

Example 1



There is an infinite number of solutions
and this is a COMPLEX problem indeed!

How to solve such a complex NONLINEAR problem?

There are two possibilities.

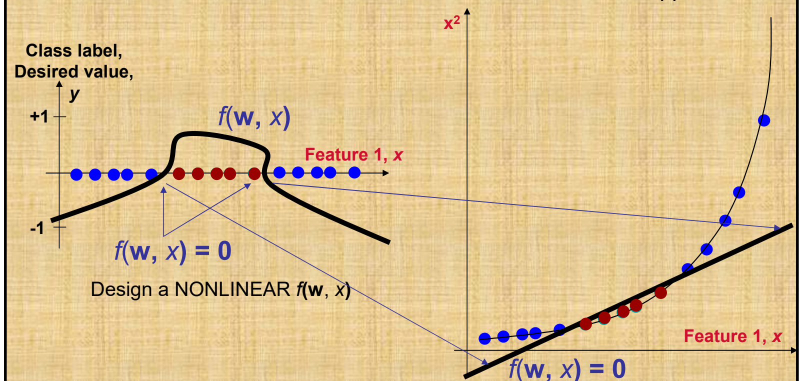
Example 1:

1) Solve in the original *x domain*

This is **not a feasible** approach

2) **Map data into an extended features' *domain***

This is **the RIGHT** approach

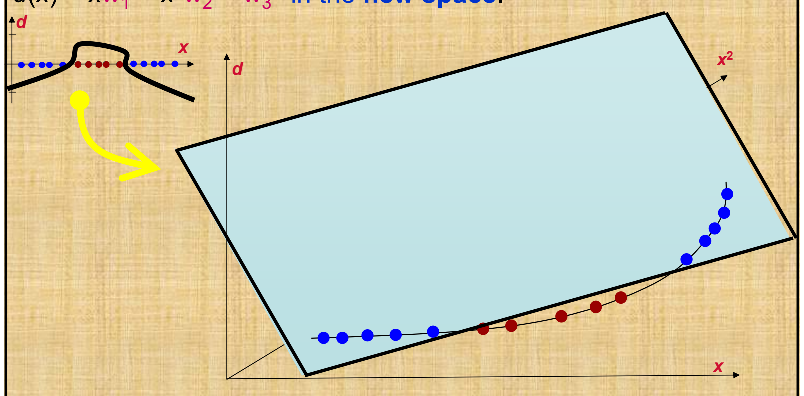


Design a LINEAR decision function in a NEW features space (which is a plane here).

Note that we do not see Class Labels in the right hand graph!

Let's see this new space

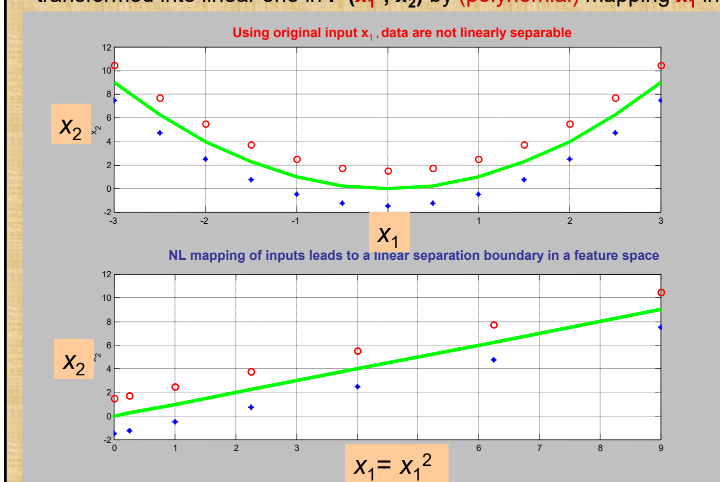
So, instead of a NL decision function in the original 1 dimensional space, we want to create a LINEAR decision function defined as $d(x) = xw_1 + x^2w_2 + w_3$ in the **new space**.



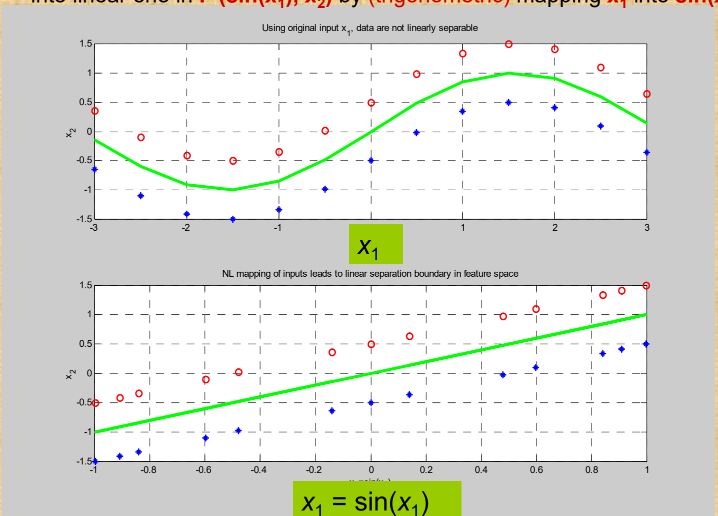
Two more examples of mapping to NEW space

Why may the mapping of input space X (x_1, x_2) into feature space F ($f(x_1)$, $f(x_2)$) be useful?

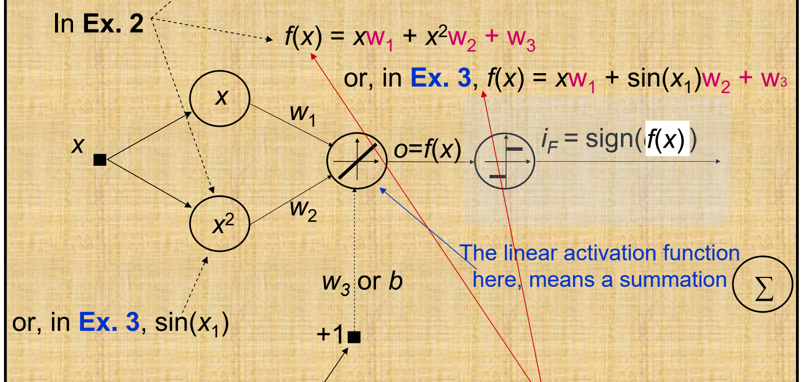
Example 2: Nonlinear (quadratic) separation boundary in **X** (**x_1, x_2**) is transformed into linear one in **F** (**x_1^2, x_2**) by (polynomial) mapping **x_1** into **x_1^2**



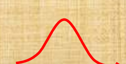
Example 3: NL i.e., sinusoidal separation boundary in **X (x_1, x_2)** is transformed into linear one in **F (sin(x_1), x_2)** by (trigonometric) mapping x_1 into **sin(x_1)**



An extension (i.e., NONLINEAR mapping) of an input space x into the new feature space [x x^2] can be given **the graphical representation in the form of a 'neural' network below**



One Comment!

- I was "cheating" while doing mappings to a hidden layer because I knew the nonlinearities and I used them while mapping.
- In fact, that was not a true cheating because any other NONLINEAR MAPPING would've solved the problem ☺
- **Important is to do a NONLINEAR MAP of ORIGINAL DATA into, so called, HIDDEN SPACE!!!**
- Two the most popular mappings are
  - Sigmoidal Function  & 
  - Radial Basis Function (RBF), in particular, GAUSSIAN
- The mappings were dubbed differently:
- Activation functions (in NNs), Basis functions (in RBF NNs), Kernels (in SVM), Membership functions (in Fuzzy Logic)
- Next page shows few more popular NL mappings.

Classic & Novel (a.k.a. Deep Learning) Activations Functions & Their Derivatives

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \ne 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

$f(x)$, $o(u)$ — x, u

Notice! x in this slide is u in all the other ones. Also note that $f(x) = o$ i.e., $o(u)$

Deep Learning Act. Functions

Finally we arrived at something which looks as a **neural network** (**NN**)!

However, some/many of you have already seen such models, you've worked with them but you were not aware ☺ that those models can be shown as NNs i.e., as SVMs.

Some connections between

NNs (*or* SVMs)

and

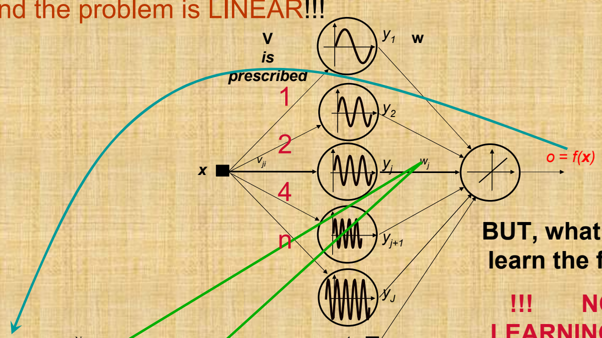
classic techniques such as Fourier series and Polynomial approximations

Classic approximation techniques in NN graphical appearance

FOURIER SERIES

AMPLITUDES and PHASES of sine (cosine) waves are unknown, but frequencies are known because <u>Mr Joseph Fourier has selected frequencies for us</u> -> they are INTEGER multiples of some pre-selected base frequency.

And the problem is LINEAR!!!



$f(x) = \sum_{k=1}^{N} a_k \sin(kx)$, *or* $b_k \cos(kx)$, *or both*

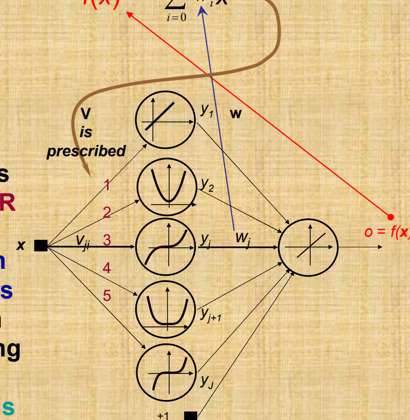
BUT, what if we want to learn the frequencies?

!!! NONLINEAR LEARNING PROBLEM !!!

Another classic approximation scheme is a

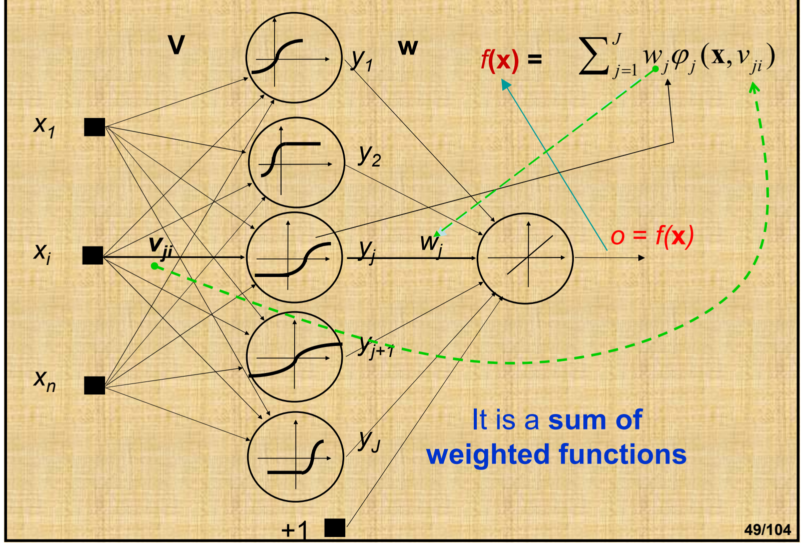
POLYNOMIAL SERIES

$f(x) = \sum_{i=0}^{N} w_i x^i$



With a prescribed (integer) exponents this is again LINEAR APPROXIMATION SCHEME. Linear in terms of parameters to learn and not in terms of the resulting approximation function. This one is NL function for *i* > 1.

A Classic Neural Network Is of The Same Structure



However, before going to details of learning i.e. of training NN, a basic statistical concepts while learning from data must be introduced now. They are

BIAS and VARIANCE

&

CROSS VALIDATION

Bias – Variance Dilemma!

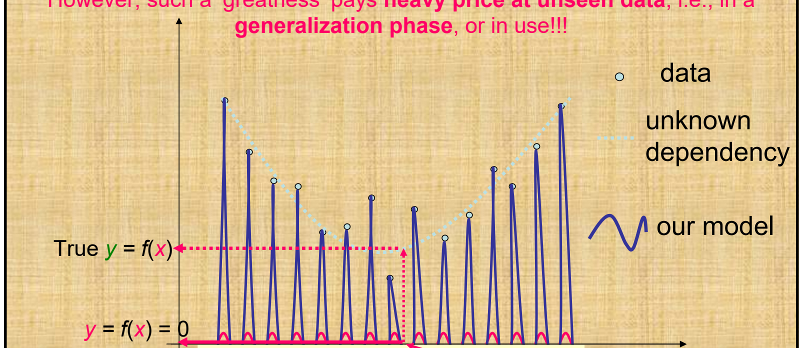
It is **the must piece of the knowledge** in order to get an idea of the relationship between the data, models and errors!

It will be **intuitive**, without math or any equation and it will serve for warming up! Check Kecman's book (there are many others, better and more specialized too) if you prefer math.

Training and Generalization

Today, having powerful computers and good math software it is easy to be **'great** and **perfect'** on the training data set!

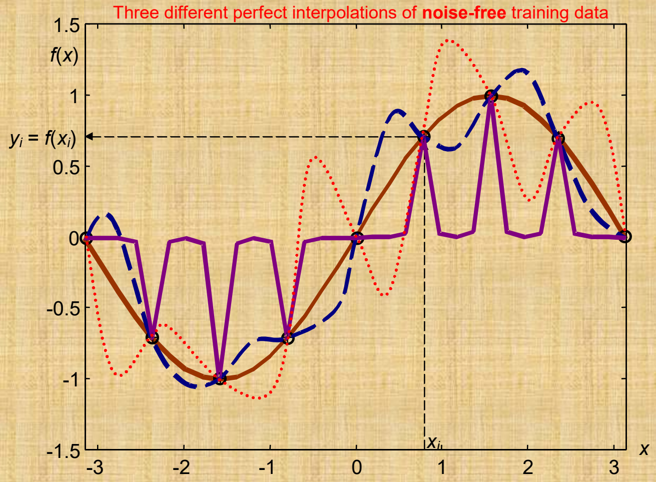
However, such a 'greatness' pays **heavy price at unseen data**, i.e., in a **generalization phase**, or in use!!!



For this, during the training unseen, input x the model gives $y = f(x) = 0$
This is (deliberately chosen) extremely bad modeling, **but real!**
The same or similar phenomena will be present in the high dimensional cases, too!

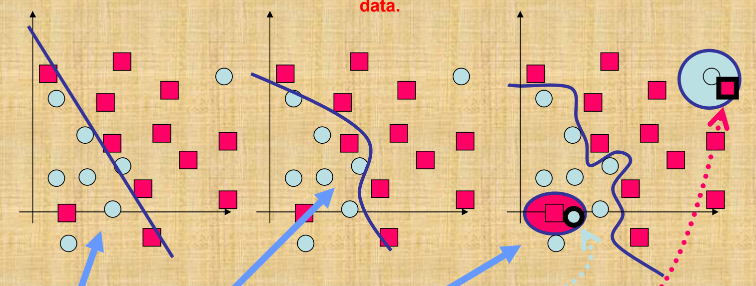
One more example showing **the perfect training results**, but **very bad generalization ones.**

Note that all three models have the training error equal zero! Bias = 0! Perfect interpolants!



And, still one more example, but now from the PATTERN RECOGNITION (CLASSIFICATION) task, showing various models and their performances.

Note that the last model (learning machine) learns perfectly, i.e., separates all the training data.



On the left, the separation boundary is linear, and it misses not only the outliers, but some 'easy' points. The solution on the right does not miss anything. By having high capacity, it learns each data belongings 'by heart', but it is unlikely that it will perform well on the new data, say this one

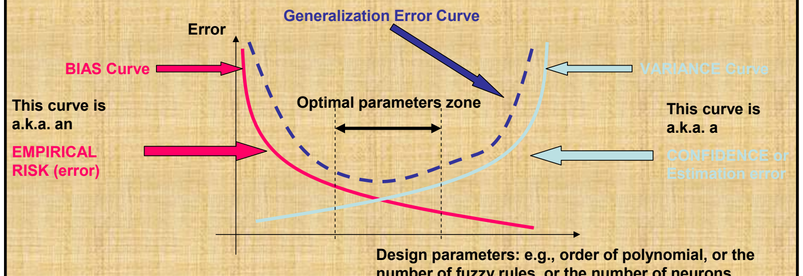
Or, this one

Central solution is of an intermediate capacity, separating most of the points, without putting too much trust into any particular training data point!!!

Obviously, we need much <u>more</u> than being good (or even excellent) on the training data set!

This 'more' means, we want that our models perform well on all future, previously unseen data, generated by the same data generator (i.e., plant, system, process, probability distribution).

The whole statistical learning fights (optimizes) the following two curves!



Although the graph looks very simple, finding the optimal modeling parameters is an EXTREMALY DIFFICULT task!!!

and, this is due to the following facts:

- **we never (or, rarely only) know the underlying probability distribution, meaning the data generation, function,**
- **we never know the space of (<u>target</u>) functions, or to which class of functions our f. belongs**
- **we always have scarce (insufficient, not enough) data,**
- **our data are always high- (or/and extremely high-) dimensional,**
- **there is always the noise, or data are corrupted**

Bias & **Variance**

In modeling an **unknown dependency (regression or discrimination function)**, without knowledge of its mathematical form (**target space**), our **models (functions from hypothesis space**) produce **approximating functions**, which may be incapable of representing the **target function** behavior.

A difference between the model output and unknown target function (data) is called **the bias.**

When there are not sufficient data, (or even if there appears to be sufficient representative data, **noise** contamination can still contribute that) **the sample of data** that is **available for training** *may not be representative of **average** data generated by the target function*.

Consequently, there may be a difference between a network output **for a particular data set**, and network function output **for the average of all data sets** produced by the target function.

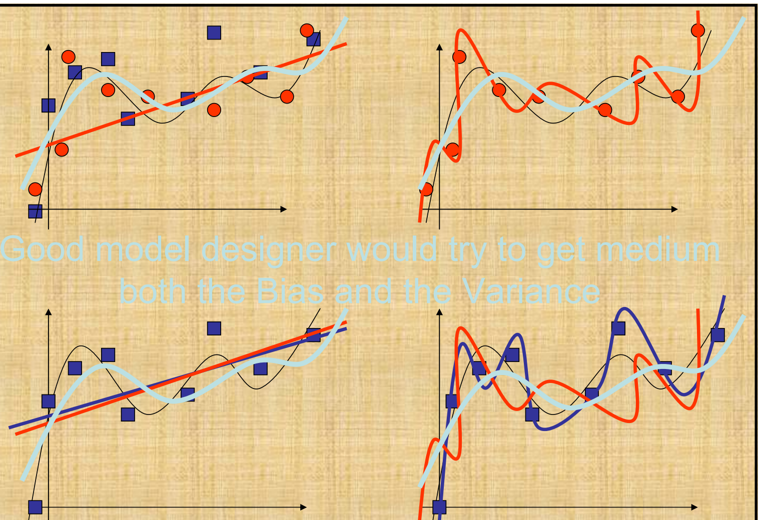
The *square* of this difference is called **the variance**

Remember, our basic task is to **learn, i.e. to estimate, the underlying function and to filter out all possible noises**.

Too **simple model** (NN with small number of neurons or polynomial of low order) results in a **Big BIAS and Small VARIANCE**,

while a very **complex model** (NN with plenty of neurons or a polynomial of high order) produces **Small BIAS and Big VARIANCE**

We explain the above, by presenting a meaning of BIAS and VARIANCE geometrically (graphically) !



Good model designer would try to get medium both the Bias and the Variance

High Bias - Low Variance

Low Bias - High Variance

The **Experimental** Tool for resolving a Bias-Variance Dilemma is a **Cross Validation**

Slides on Cross Validation are Coming Next

And now, finally, we arrived at

NEURAL NETWORK

However, **not yet!**

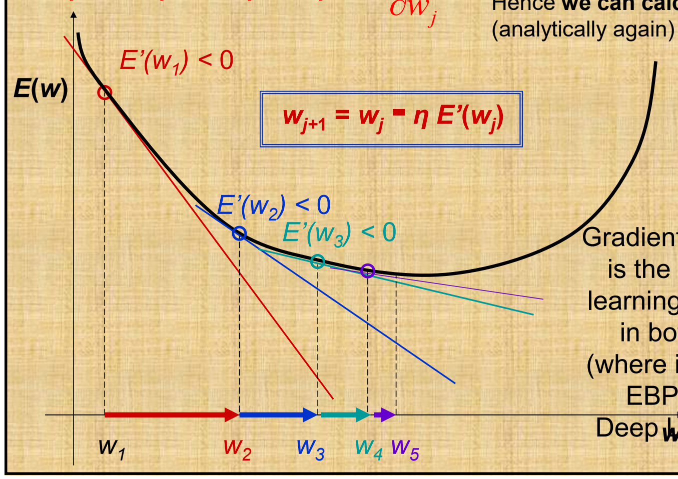
NNs learning is a GRADIENT based one, and in order to understand their training we must show the basics of a

Gradient Method in Optimization

A basic idea of an iterative gradient method

$$w_{j+1} = w_j + \Delta w_j = w_j - \eta \frac{\partial E}{\partial w_j}$$

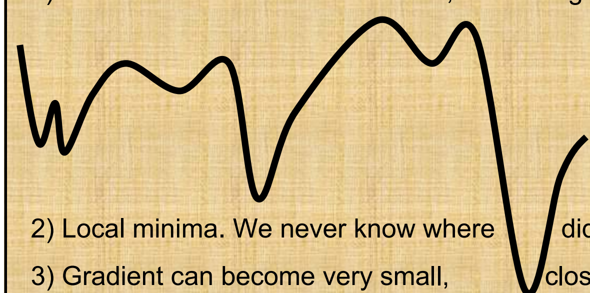
Note! **We know (analytically) the function *E*(*w*).** Hence **we can calculate** (analytically again) ***E'*(*w*).**



Gradient descent is the BASIC learning method in both NN (where it's a.k.a. EBP) and Deep Learning

Issues with an iterative gradient method

1) We don't know where to start i.e., initial weights are ?????



2) Local minima. We never know where did we end.

3) Gradient can become very small, close to 0.

4) What should be a learning rate η? Too small one, learning takes forever while a too big one can lead to a random jumping over the nonlinear surface or lead to a divergence

Solution – Crossvalidate and use "tricks"!

Good news – gradient method works very well in a real life ☺

Multilayer Perceptron NN

Error Back Propagation Learning Algorithm

Until now you have learnt about:

- LMS Learning Rule
- Linear Neuron Learning Rule by PINV

- Now, we'll present the learning in the Multilayer Neural Networks having sigmoidal functions as the activation i.e., transfer functions
- (this is Chapter 4 in your textbook)

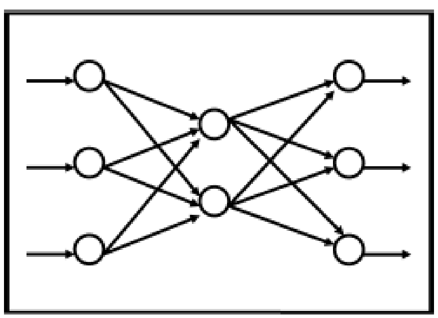
Architecture of an ANN

Architecture depends on: kind of links between neurons
➔ different trainingsalgorithm (learning rules) for the weights

Feedforward networks: Links only from one layer to the next (no feedback connections).

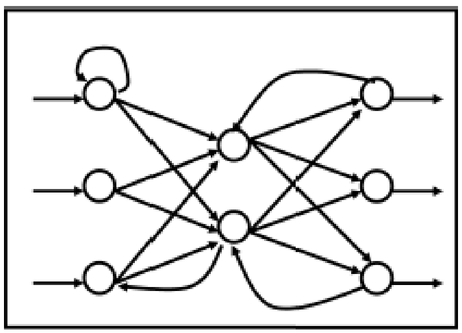
Feedback networks: Output of a neuron is directly or indirectly by other neurons linked back to its input

Lateral networks: Links between neurons of a layer

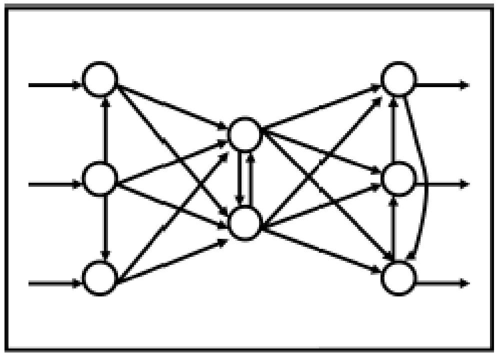


Feedforward network

We'll introduce and analyze only these ones

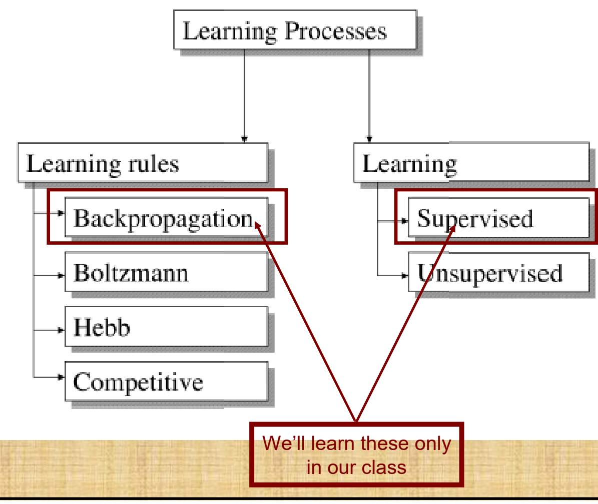


Feedback network

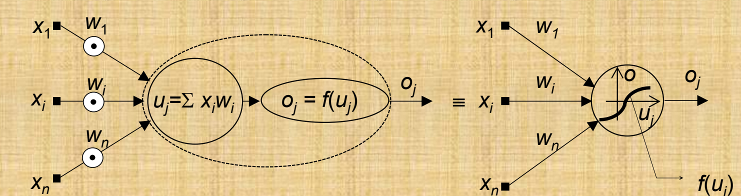


Lateral network

Learning processes and -rules



Structure of the Single Neuron *j*



u_j is the input to the neuron calculated as the scalar i.e., dot product

$$u_j = \sum_{i=1}^{n} x_i w_i = \mathbf{x}'\mathbf{w} = \mathbf{w}'\mathbf{x}$$

All & Everything about NNs is ABOUT INDEXING!!!

Notation used is as follows:

j:	the *j*-th HL neuron
i:	index of the input
n:	number of the inputs
x_i:	input *i*
w_{ji}:	weight to the HL neuron *j* from the input *i*
u_j:	sum of the weighted inputs to the neuron

$$u_j = \sum_{i=1}^{n} x_i w_{ji} = \mathbf{x}'\mathbf{w}_j = \mathbf{w}_j'\mathbf{x}$$

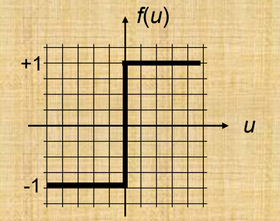
$f(u_j)$:	transfer or activation function of the neuron
$y_j=f\,(u_j)$	output from the HL neuron *j*

Transfer i.e. Activation Functions of Artificial Neurons

PERCEPTRON's AF is

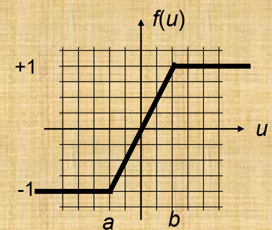
Threshold Function

$$f(u) = \begin{cases} 1 & u > 0 \\ -1 & u < 0 \end{cases}$$



Constrained Linear Function

$$f(u) = \begin{cases} 1 & u > b \\ u & a < u < b \\ -1 & u < a \end{cases}$$



Classic & Novel (a.k.a. Deep Learning) Activations Functions & Their Derivatives

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

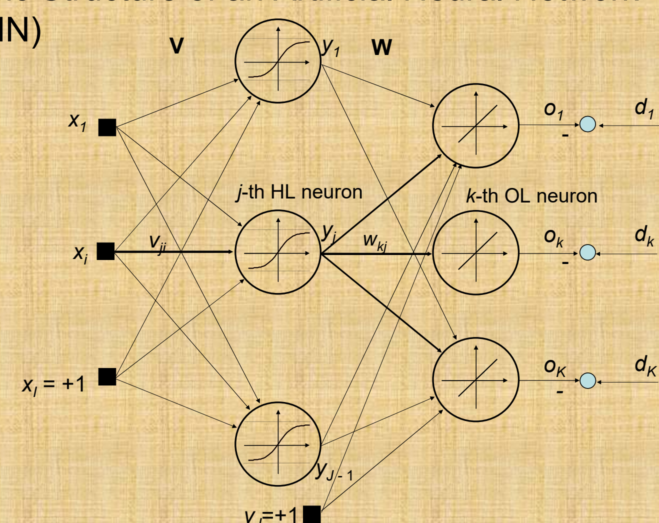
$f(x)$, $o(u)$ — x, u

Notice!

$\boldsymbol{x}$ in this slide is $\boldsymbol{u}$ in all the other ones. Also note that $f(x) = o$ i.e., $o(u)$

Deep Learning Act. Functions

Basic Structure of an Artificial Neural Network (ANN)



ERROR BACKPROPAGATION

For the calculation of the neuron weights various algorithms are used, depending upon the network architecture.

Mostly used and well known is the **Error Backpropagation (EBP) Algorithm**

For a given input pattern **x**, a desired output d_j and an ANN with the weights w_i an output o_j is calculated.

The error e is given by the difference $d_j - o_j$ and it should be minimized by weight changes.

The **output error is backpropagated** to the preceding

WARNING

almost all is about Indices i.e. Indexes!

- In what is following THE GOOD OLD CHAIN DERIVATIVES RULE is applied and one must be very cautious about the indices.
- Differentiating i.e. distinguishing between the layers and neurons is crucial.
- This is done by indexes!!!
- Remember index of the layer
  - Input Layer is ***i***
  - Hidden Layer is ***j***
  - Output Layer is ***k***

Notation is used as follows:

x_i: input i to the hidden layer

$y_j = f(u_j)$ output from the j-th hidden layer neuron

d_k: desired output for the neuron k in the output layer

o_k: output from the k-th neuron in the output layer

$f(u)$: transfer or activation function of the neuron

u: sum of the weighted inputs to the neuron

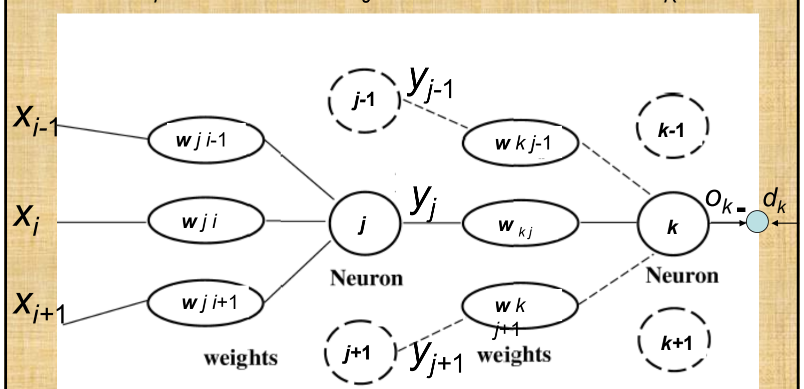
$$u_j = \sum_{i=1}^{I} x_i v_{ji} \qquad u_k = \sum_{j=1}^{J} y_j w_{kj}$$

For a logistic i.e. sigmoidal AF

$$y_j = f(u_j) = \frac{1}{1+e^{-u_j}} \qquad o_k = f(u_k) = \frac{1}{1+e^{-u_k}}$$

Example for a Backpropagation Network

I *J* *K*



Input **x** *Hidden Layer Weights* *Hidden Layer* *Output Layer Weights* *Output Layer*

The cost function for optimization is proportional to the sum of squared errors (SSE) of the neuron j in the output layer and it is:

$$E_k = \frac{1}{2}(d_k - o_k)^2 \qquad (2.1)$$

The total error E of an output layer is:

$$E = \sum_k E_k = \frac{1}{2}\sum_k (d_k - o_k)^2 \qquad (2.2)$$

The task is to **minimize the error *E*** (which is **SSE**) by **changing the weights** by **using <u>the GRADIENT DESCENT method</u>** for **obtaining the required weight change Δw_{kj}.**

$$\boxed{w_{kj} = w_{kj} + \Delta w_{kj} = w_{kj} - \eta \frac{\partial E_k}{\partial w_{kj}},} \qquad 0 < \eta < \eta_{crit} \qquad (2.3)$$

η is a step size, known as the learning rate. For quadratic cost function, if $\eta > \eta_{crit}$ the gradient descent does not converge. For general NL cost function η_{crit} is problem dependent!

$$\Delta w_{kj} = -\eta \frac{\partial E_k}{\partial w_{kj}}, \qquad (2.4)$$

with (2.1) and:

Note, the derivation that follows is for a logistic i.e. sigmoidal AF

$$o_k = f(u_k) = \frac{1}{1+e^{-u_k}} \qquad (2.5)$$

$$u_k = \sum_{k=1}^{J} y_j w_{kj} \qquad (2.6)$$

it follows:

$$\Delta w_{kj} = -\eta \frac{\partial E_k}{\partial w_{kj}} = -\eta \frac{\partial E_k}{\partial o_k} \frac{\partial o_k}{\partial w_{kj}} = -\eta \frac{\partial E_k}{\partial o_k} \frac{\partial o_k}{\partial u_k} \frac{\partial u_k}{\partial w_{kj}}. \qquad (2.7)$$

Note that

$$\frac{\partial E_k}{\partial o_k} = \frac{-2}{2}(d_k - o_k), \qquad (2.8)$$

as well as that $\partial o_k / \partial u_k$ can be expressed as

$$\frac{\partial o_k}{\partial u_k} = \frac{\partial f(u_k)}{\partial u_k} = f'(u_k) = \frac{1}{\left(1+e^{-u_k}\right)^2} e^{-u_k} = \frac{1}{\left(1+e^{-u_k}\right)} \frac{e^{-u_k}}{\left(1+e^{-u_k}\right)}$$

$$= \frac{1}{\left(1+e^{-u_k}\right)}\left(1 - \frac{1}{\left(1+e^{-u_k}\right)}\right) = o_k(1-o_k). \qquad (2.9)$$

Note, this is an expression ONLY for a **logistic i.e. sigmoidal** AF

Also, by the fact that

$$\frac{\partial u_k}{\partial w_{kj}} = \frac{\partial \sum w_{kj} y_j}{\partial w_{kj}} = y_j, \qquad (2.10)$$

we have got **$\Delta w_{kj} = \eta(d_k - o_k) o_k(1 - o_k) y_j$,** (2.11)

and the new weights are: **$\Delta w_{kj} = w_{kj} + \Delta w_{kj}$** . (2.12)

If **tangent hyperbolic** was used, (2.9) is **$(1 - o_k^2)$** and

$\Delta w_{kj} = \eta(d_k - o_k)(1 - o_k^2) y_k$

Here we are introducing the DELTA δ_{ok} variable

$$\delta_{ok} = -\frac{\partial E}{\partial u_k} = -\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial u_k} = \underbrace{(d_k - o_k)}_{e_k} f'(u_k)$$

$$\delta_{ok} = (d_k - o_k) f'_{ok}(u_k) = e_k f'_{ok}(u_k) = e_k o_k (1-o_k) \quad \text{for logistic AF}$$

$$\delta_{ok} = (d_k - o_k) f'_{ok}(u_k) = e_k f'_{ok}(u_k) = e_k (1-o_k^2) \quad \text{for tgh AF}$$

Backpropagation means:

The error of the output layer is backpropagated to both the output layer weights and the hidden layer ones.

Same as in (2.3) the hidden layer weights' changes are calculated as:

$$\Delta v_{ji} = -\eta_h \frac{\partial E_j}{\partial v_{ji}}, \qquad 0 < \eta_h < \eta_{h\_crit} \qquad (2.14)$$

η and η_h are usually chosen to be same, but they also can be different. Having the following expressions for o_j and u_j,

$$o_j = f(u_j) = \frac{1}{1+e^{-u_j}}, \qquad (2.15)$$

$$u_j = \sum_{i=1}^{I} x_i v_{ji}, \qquad (2.16)$$

one can proceed by calculating required derivatives (gradients) as follows on the next page,

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial v_{ji}}, \qquad (2.17)$$

$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial v_{ji}} = \eta \delta_{yj} x_i, \qquad (2.18)$$

$$\delta_{yj} = -\frac{\partial E}{\partial u_j} \qquad (2.19)$$

$$\delta_{yj} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial u_j} \qquad \frac{\partial y_j}{\partial u_j} = f_j'(u_j)$$

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}\{\frac{1}{2}\sum_{k=1}^{K}[d_k - f(u_k(\mathbf{y}))]\}^2$$

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K}(d_k - o_k)\frac{\partial}{\partial y_j}\{f[u_k(\mathbf{y})]\}$$

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K}\underbrace{(d_k - o_k)f'(u_k)}_{\delta_{ok}\ from(4.9)}\frac{\partial u_k}{\partial y_j}$$

By using

$$u_k = w_{k1}y_1 + w_{k2}y_2 + \ldots + w_{kj}y_j + \ldots + w_{kJ}y_J$$

the derivative term above equals w_{kj}, and

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K}\delta_{ok}w_{kj}$$

Combining last expression we obtain THE EXPRESION FOR AN ERROR SIGNAL of the HL NEURONS

$$\delta_{yj} = f_j'(u_j)\sum_{k=1}^{K}\delta_{ok}w_{kj}$$

Finally the weight's adjustment for both an output layer weight w_{kj} and a hidden layer weight v_{ji} are

$$\Delta w_{kj} = \eta f^{'}(u_k)(d_k - o_k)y_j = \eta \delta_{ok} y_j$$

$$w_{kj} = w_{kj} + \Delta w_{kj} = w_{kj} + \eta f^{'}(u_k)(d_k - o_k)y_j = w_{kj} + \eta \delta_{ok} y_j$$

$$\Delta v_{ji} = \eta \left(f_j{'}(u_j) \sum_{k=1}^{K} \delta_{ok} w_{kj} \right) x_i,$$

$$v_{ji} = v_{ji} + \Delta v_{ji} = v_{ji} + \eta f_j{'}(u_j) x_i \sum_{k=1}^{K} \delta_{ok} w_{kj},$$

Hence, the general shapes for the error backpropagation learning rules for the output layer weights as well as for the hidden layer ones are given below:

$$\delta_k = f'(u_k)\,(d_k - o_k) = \underline{f'(u_k)}\,\underline{e_k}$$

$$\delta_j = \underline{f'(u_j)} \underline{\sum_{k=1}^{K} f'(u_k)(d_k - o_k) w_{kj} = f'(u_j) \sum_{k=1}^{K} \delta_k w_{kj}}$$

$\Delta w_{kj} = \eta \delta_k y_j$, **output layer weights**

$\Delta v_{ji} = \eta_h \delta_j x_i$, **hidden layer weights**

The use of δ **(delta) error signals** is extremely handy and this is why the EBP is often called the **Generalized Delta Learning Rule**!

Example on the next three slides show this very

Table is from Kecman's The MIT Press book, p. 264

The index *p* denotes that we are doing an iterative i.e., on line learning and *p* denotes the index of pattern i.e., sample used.

Notice the important order:

the steps 6 & 7 do precede the steps 8 & 9!!!

Table 4.1a: Summary of the EBP algorithm - online version

Given is a set of P measured data that will be used for training: $X = \{\mathbf{x}_p, \mathbf{d}_p, p = 1, \dots, P\}$ consisting of the input pattern vectors $\mathbf{x} = [x_1, \ x_2, \ \dots, \ x_n, \ +1]^T$ and the output desired responses $\mathbf{d} = [d_1, \ d_2, \ \dots \ d_K]^T$

FEEDFORWARD PART

STEP 1: Choose the learning rate η and predefine the maximally allowed, or desired, error E_{des}.
STEP 2: Initialize weight matrices $\mathbf{V}_p(J\text{-}1, I)$ and $\mathbf{W}_p(K, J)$.
STEP 3: Perform the online training (weights are adjusted after each training pattern), $p = 1, \dots, P$
Apply the new training pair $(\mathbf{x}_p, d_p)$ in sequence, or randomly, to the hidden layer neurons.
STEP 4: Consecutively calculate the outputs from the hidden and output layer neurons.
$y_{jp} = f_h(u_{jp}),\ o_{kp} = f_o(u_{kp})$
STEP 5: Find the value of the sum of errors square cost function E_p for the data pair applied and given weight matrices $\mathbf{V}_p$ and $\mathbf{W}_p$. (in the first step of an epoch initialize $E_p = [\,]$).

$$E_p = \frac{1}{2}\sum_{k=1}^{K}(d_{kp} - o_{kp})^2 + E_p$$

Note that the value of the cost function is accumulated over all the data pairs.

BACK PROPAGATION PART

STEP 6: Calculate the output layer neurons' error signals δ_{okp} as follows

$$\delta_{okp} = \underbrace{(d_{kp} - o_{kp})}_{e_{kp}} f_{ok}{'}(u_{kp}) \quad (k = 1, \dots, K)$$

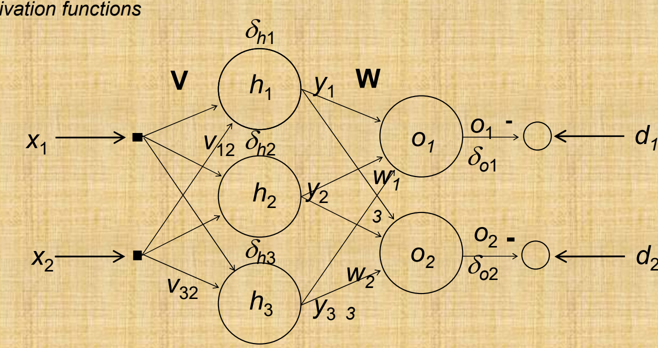
STEP 7: Calculate the hidden layer neurons' error signal δ_{yjp}

$$\delta_{yjp} = f_{hj}{'}(u_{jp}) \sum_{k=1}^{K} \delta_{okp} w_{kjp} \quad (j = 1, \dots, J\text{-}1)$$

STEP 8: Calculate the updated output layer weights $w_{kj,\,p+1}$
$w_{kj,\,p+1} = w_{kj\,p} + \eta\, \delta_{okp}\, y_{jp}$
STEP 9: Calculate the updated hidden layer weights $v_{ji,\,p+1}$
$v_{ji,\,p+1} = v_{ji\,p} + \eta\, \delta_{yjp}\, x_{ip}$
STEP 10: If $p < P$ go to *STEP 3*, otherwise go to *STEP 11*

STEP 11: The *learning epoch* (the sweep through all the training patterns) is completed, ($p = P$).
For $E_P < E_{des}$ terminate learning, otherwise go to *STEP 3* and start a new learning epoch, $p = 1$.

EXAMPLE: *For the network shown below, calculate the expressions for the weight changes using the EBP algorithm in an online learning mode. The training data consisting of the input pattern vectors* $\mathbf{x} = [x_1 \ \ x_2]^T$ *and the output desired responses* $\mathbf{d} = [d_1 \ \ d_2]^T$, *are given as* $X = \{\mathbf{x}_p, \mathbf{d}_p, p = 1, \dots, P\}$. *Note that* h_j *and* o_k *denote the HL and OL activation functions*



Important, we know **x, d, V, W** and activation functions **h** & **o** and their derivations. What we want is **iteratively**, going backward, update **V** & **W** in order to **minimize error** *E*(**V,W**)

After choosing initial set of the weights randomly, or by using some 'good' heuristics, and after presenting the very first input vector **x** = $[x_1 \;\; x_2]^T$, the output vector **o** = $[o_1 \;\; o_2]^T$ is calculated first. Knowing activation functions in neurons, their derivatives can be readily calculated and by using the given desired vector **d** $=[d_1 \;\; d_2]^T$, we can **calculate the delta signals** for the OL neurons

$$\delta_{ok} = e_k f_k', \qquad (k = 1, 2).$$

Having δ_k we can find the hidden layer neurons' deltas (or error signals) δ_j as follows

$$\delta_j = f_j' \Sigma_k \, \delta_{ok} w_{kj}, \qquad (j = 1, 2, 3, \quad k = 1, 2).$$

Only now can we calculate the weight changes for specific weights. Thus, for example

$$\Delta v_{12} = \eta\delta_1 x_2, \quad \Delta v_{32} = \eta\delta_3 x_2, \quad \Delta w_{23} = \eta\delta_{o2} y_3, \quad \Delta w_{13} = \eta\delta_{o1} y_3$$

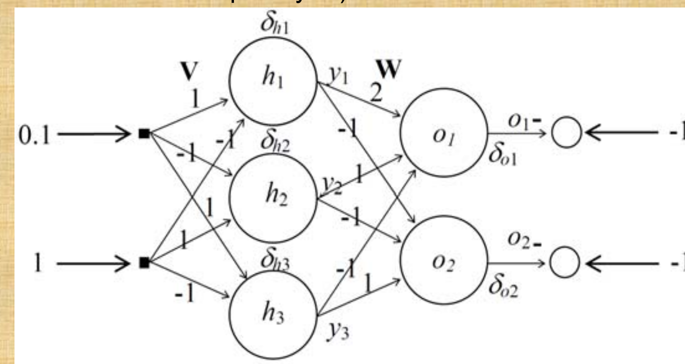
After the first data pair has been used the new weights obtained are

$$v_{12} = v_{12} + \Delta_{v12}, \; v_{32n} = v_{32} + \Delta v_{32},$$
$$w_{23} = w_{23} + \Delta w_{23}, \; w_{13n} = w_{13} + \Delta w_{13},$$

where the subscripts *n* and *o* stand for *new* and *old* respectively.

Now, EBP i.e., calculation of delta signals with real activation functions (AFs), weights and signals

Calculate all delta (δ) signals in NN below if AFs are as follows: $h_1 = u_{j1}^2$, $h_2 = u_{j2}^3$ and $h_3 = 2u_{j3}$. In output layer AFs a linear function i.e., $o_k = u_k$. (Remind, *j* are indices in a hidden layer and *k* are the ones in the output layer.)



Solution

$$\mathbf{u}_{hl} = \begin{bmatrix} -0.9000 \\ 0.9000 \\ -0.9000 \end{bmatrix} \; \mathbf{y} = \begin{bmatrix} 0.8100 \\ 0.7290 \\ -1.8000 \end{bmatrix} \; \mathbf{y}' = \begin{bmatrix} -1.8000 \\ 2.4300 \\ 2.0000 \end{bmatrix} \; \mathbf{u}_k = \begin{bmatrix} 4.1490 \\ -3.3390 \end{bmatrix}$$

$$\mathbf{e} = \begin{bmatrix} -5.1490 \\ 2.3390 \end{bmatrix} \quad \mathbf{o}' = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \delta_o = \begin{bmatrix} -5.1490 \\ 2.3390 \end{bmatrix} \quad \delta_h = \begin{bmatrix} 22.7466 \\ -18.1958 \\ 14.9760 \end{bmatrix}$$

Solve it at home tonight, please!

If you get same delta signals values you can claim your expertise in NN i.e., in EBP algorithm and you can start writing your first NN code in any language you like, prefer or need ☺

Now we'll discuss The Bag of NNs' Tricks which is woven in order to address the following practical issues, questions, dilemmas*

- How many hidden layers (HL) – 1, 2, 3, more?
- How many neurons, i.e. activation functions, in HL (bias-variance dilemma)?
- What activation function to use?
- How to initialize the weights? **W** range?
- Error function for measuring the quality of learning & for stopping the learning?
- Learning rate and Momentum term?

* Pages 266 – 302 in Kecman's The MIT Press book

If you think this was a huge bag with too many heuristics (popularly known as tricks) consider this:

NN's pocket valet of tricks

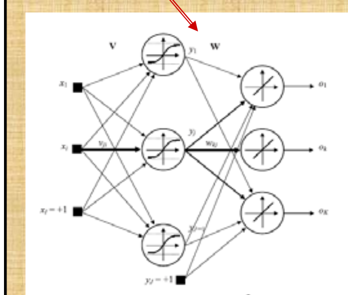
Deep Learning's 30" Spinner of tricks

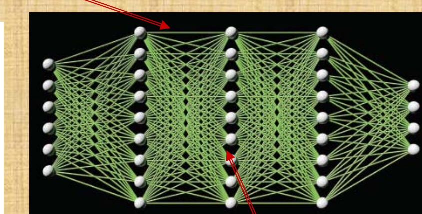
What's a Deep Learning (DL)?

- Well, it is
- Multilayer Perceptron (with more, much more, HLs)
- Trained by gradient descent (a kind of EBP)
- Remind, gradient is a differentiation of error norm ***E*(w)**
- Novel activation functions are introduced
- Powered by everything you've got on your computer laptop, desktop, workstation, server, …
- Meaning, all cores, GPU's and whatever is of any use
- and, by using many tricks (deleting some neurons (i.e., ignoring them - dropout), stopping the learning, limiting (constraining) the weights, …, etc, …)
- **DL** is showing fine results in **Image Recognition** (and, possibly, in **NLP**) applications (as of today).

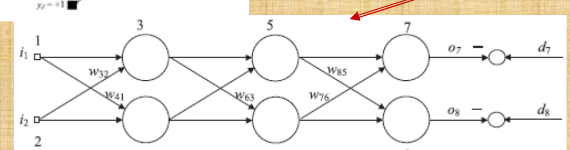
Just, two pages about
DEEP LEARNING (DL) because
DL has shown very fine results in
image recognition/processing
and there are hints that it may be
good for
natural language processing
(NLP)

Deep Learning is a Hot **Buzzword**. It's just an NN with More Hidden Layers which learn by SGD algorithm known as the EBP.





DL Networks



Two hints for all of you who want to start playing with DL algorithm:

1) start with **PyTorch** (Facebook AI people, Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan) or by

2) two free software

KERAS (François Chollet, employed by Google at a time)

i

TENSORFLOW (Google people)

I shall not dive into the sea of DL

Prof. Tom Arodz will cover DL and refer you to ideas, approaches, tricks, software, books, survey papers and internet. He'll talk about:

Automated Differentiation (AD), AD by PyTorch, Problems in Training Deep Networks and Their Resolutions

Convolutional Neural Networks (CNNs)

Residual Networks (ResNets)

☺

No Conclusions Today

Because whatever is said may be irrelevant in a matter of days!

AI, and its basic tool **ML,** are in the phase of hectic development and it is so hard to be the Prophet about what will happen tomorrow ☹,

but

it is important to jump on the AI, DL and ML wagon, start developing courses, do research in the field ASAP and

work on becoming the leader,

or, at least, a great, able user & follower!

This is **not difficult assuming the state supports you** at the beginning

☺

You won't believe it, but this is the very **last** slide !!!

Thanks!