Scientific Machine Learning: a basic Intro, Open problems and Challenges

Constantinos Siettos

Department of Mathematics and Applications “Renato Caccioppoli”

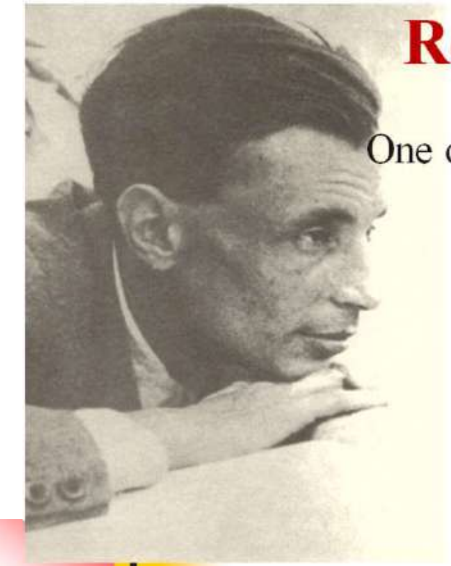
UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SSM

www.siettos.net Research Group

NUMADICS

Numerical Analysis, Machine Learning and
Data Mining for Complex and Multiscale Systems



Renato Caccioppoli

Naples, 20 Jan 1904 – Naples, 8 May 1959

One of the most important and interesting Mathematicians of the 20th century

Grandson of Michail Bakunin

- In 1927 published an important work on integration on k-dimension Varieties (related to Manifolds) in R^n

Establishing the principles of a theory of measure of plane and curved surfaces, and more generally of two or more dimensional varieties embedded in a linear space.

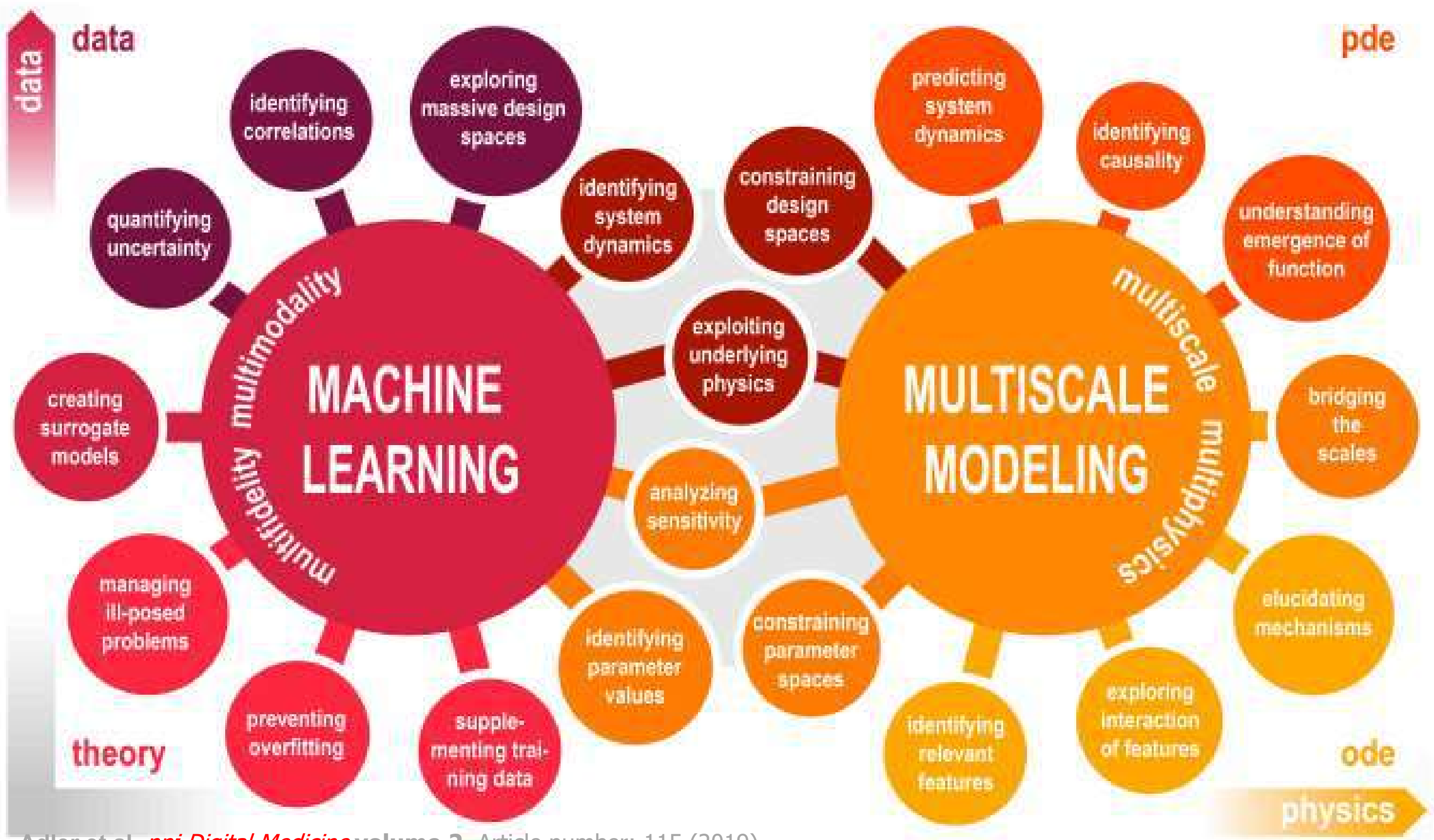
- After 1930 Caccioppoli devoted himself to the study of PDEs and ODEs and he provided existence theorems for both linear and non-linear problems
- In 1931, extended Brouwer's fixed point theorem, and applied his results to existence problems of both PDEs and ODEs.
- In May 1938 Hitler was visiting Naples with Mussolini: Caccioppoli, an antifasisct, convinced an orchestra to go our from a restaurant to play "La Marseillaise" in the street, and made a speech against the Italian and German dictators.
- He was arrested, but managed -- with the help of his aunt Maria Bakunin who was a Professor of Chemistry at the University of Naples -- to be declared mad and he was eventually sent to an asylum. In the asylum he worked on the problem of existence of closed convex surfaces of a given Riemanian metric.
- His political opposition to fascism led him to organise a strike in Naples in 1943.
- *Film: Morte di un Matematico Napoletono*

• *Film: Morte di un Matematico Napoletono*

Scientific Machine Learning

Scientific Machine Learning brings together the complementary perspectives of computational science and computer science **to craft a new generation of machine learning methods for complex applications across science and engineering**. In these applications, dynamics are **complex and multiscale**, data are sparse and expensive to acquire, decisions have high consequence, and uncertainty quantification is essential.

Santa Fe Institute



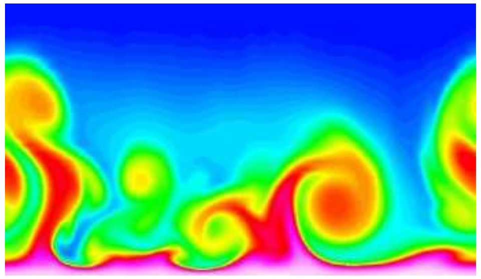
data
data
pde
identifying correlations
exploring massive design spaces
quantifying uncertainty
identifying system dynamics
constraining design spaces
predicting system dynamics
identifying causality
understanding emergence of function
multimodality
MACHINE LEARNING
multifidelity
exploiting underlying physics
creating surrogate models
analyzing sensitivity
multiscale
MULTISCALE MODELING
multiphysics
bridging the scales
managing ill-posed problems
elucidating mechanisms
identifying parameter values
constraining parameter spaces
exploring interaction of features
identifying relevant features
preventing overfitting
supple-menting trai-ning data
theory
ode
physics

Adler et al, *npj Digital Medicine* **volume 2**, Article number: 115 (2019)

Scientific Machine Learning

1 **Extract from Data Useful Information and Meaningful Patterns**

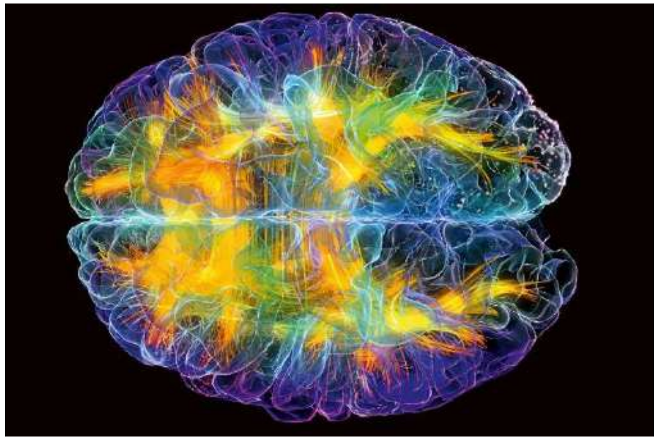
2 **Create Models to Interpret how the Data Behave and Predict their Dynamics**



Computational Simulations

For many Complex Systems,

The Physics (models or even variables) to describe the **Emergent dynamics** are not always known/ are imprecise

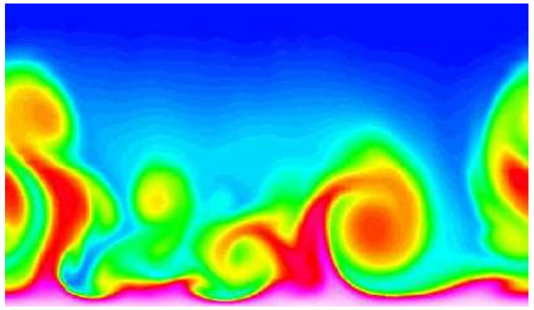


Main Objectives for Complex Systems

1 Discover Variables from Data/ Agent-Based Simulations

2 Systematically Bridge Micro and Macro Scales : THE INVERSE PROBLEM

- Construct Surrogate Models at different Scales with Machine Learning
- Numerical Analysis
  - Forward-Problem: Solving DAEs-PDEs
  - Bifurcation Analysis



Numerical Analysis/ Control

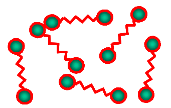
The Flourishing of SciML

Big Data, GPUs, Big! number of available microscopic/ agent-based models simulating the time evolution of Complex Systems (Biological Systems, Material Science, Complex Fluids, Epidemics, Neurons)

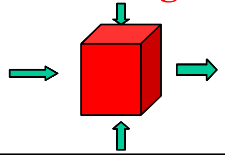
A full 90 percent of all the data in the world has been generated over the last two years.

Every day, 3.3 quintillion bytes of data (Million terabytes) created every day

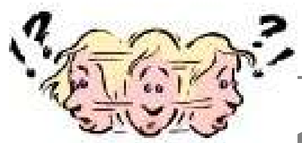
Micro-Scale



Macro Emergent) Scale



Large-Scale Microscopic Models



Mean-field (moments)

Agent-Based
Brownian D
Monte Carlo
Molecular Dynamics

Different time and space scales
Macro scales much much bigger than the bigger Microscopic scale

ODE's-PDE's

- Material Science
- Epidemiology
- Bio
- Neurons
- Markets

The analysis is usually sought at this level (system-level analysis)

… but such good models not always exist in closed form

Complex vs. Complicated or Chaos

Collective patterns emerging from many interacting components,

The emergent dynamics is more than the sum of the properties of the individual units

…decomposing the system and analyzing subunits does not necessarily give us an idea of the behavior as a whole

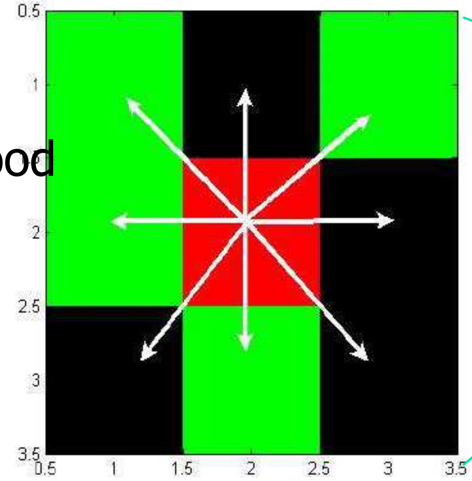
…the behavior of complex systems is therefore unpredictable

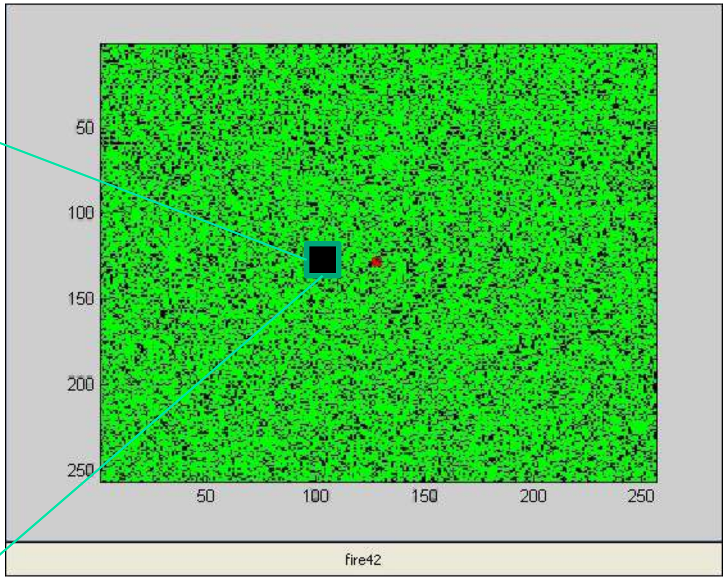
…subunits are designed and connected so that they accomplish a pre-determined (predictable or even Chaotic) behaviour

The notion of Complexity: a simplistic! model of Fire Spread

A cell can take each time one of the three states:

- 1:Black, Burned Cell
- 2:Green, Cell with Fuel/Wood
- 3:Red: Fire





The evolution rules are the following:

- Fire on a site will spread to nearest neighbors cells with Fuel at the next time step with probability **p.**

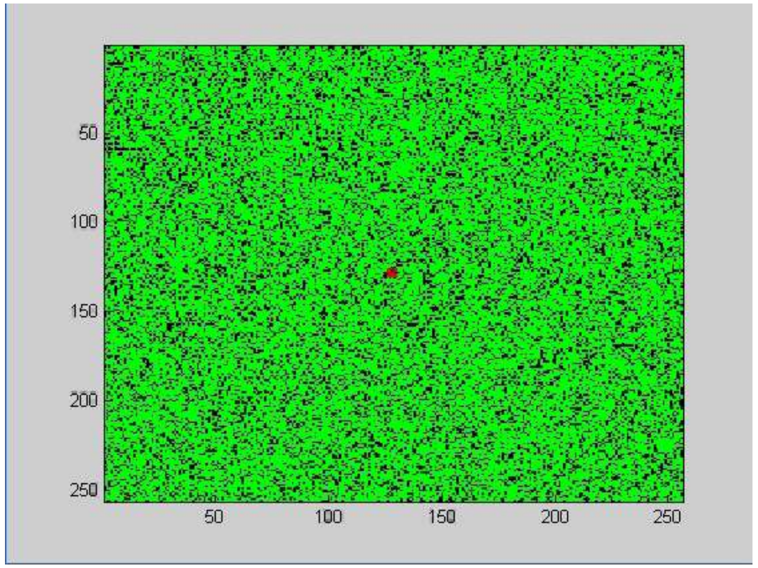
at time t **at cell (i, j)** $\xrightarrow{\text{With probability } p}$ **at time** $t+1$ **At neighbor cells**

- All cells with Fire will be burned at the next time step.

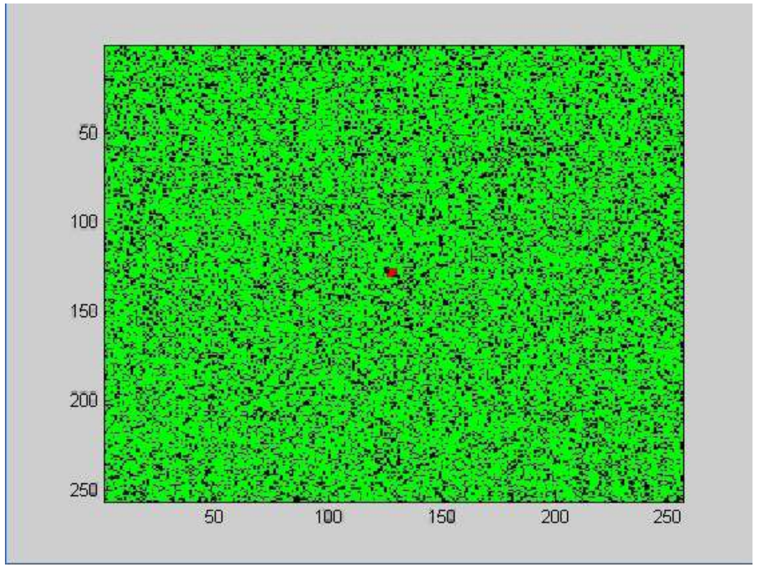
Cell with Fire At time t $\longrightarrow$ **Burned At time** $t+1$

The notion of Complexity: simple behavioral rules generate complex behavior.

P=0.44



P=0.46



The Forward and Inverse Problem in Complex Systems Modelling

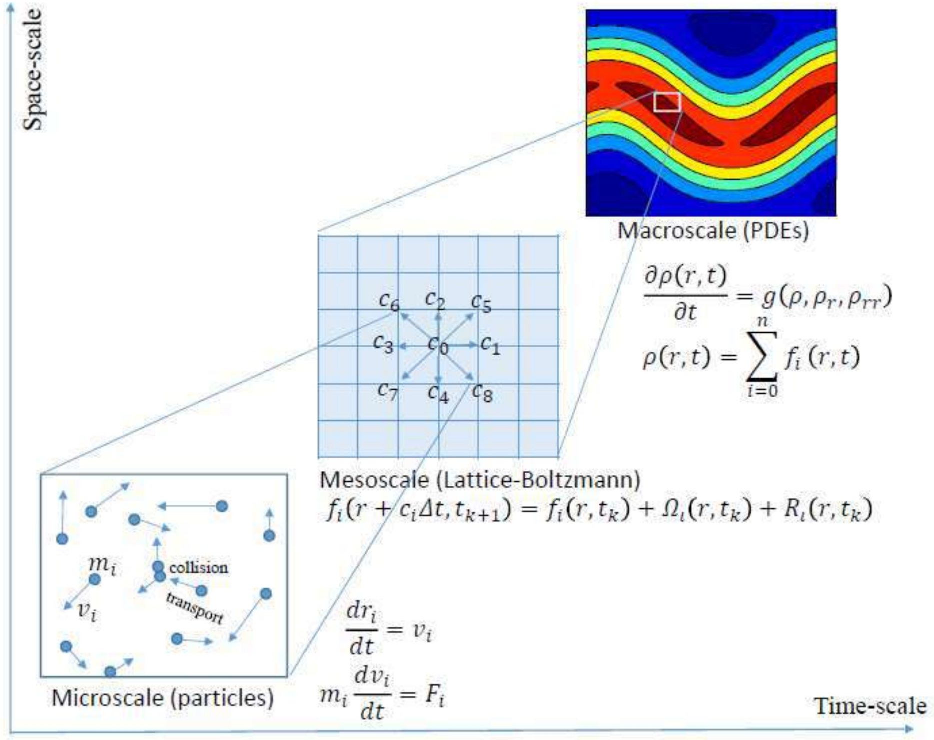
IT's ALL ABOUT DISCOVERING AND SOLVING DIFFERENTIAL EQUATIONS IN A CLOSED FORM

- Forward Problem: Numerical Solution of Large-Scale Differential Equations

-

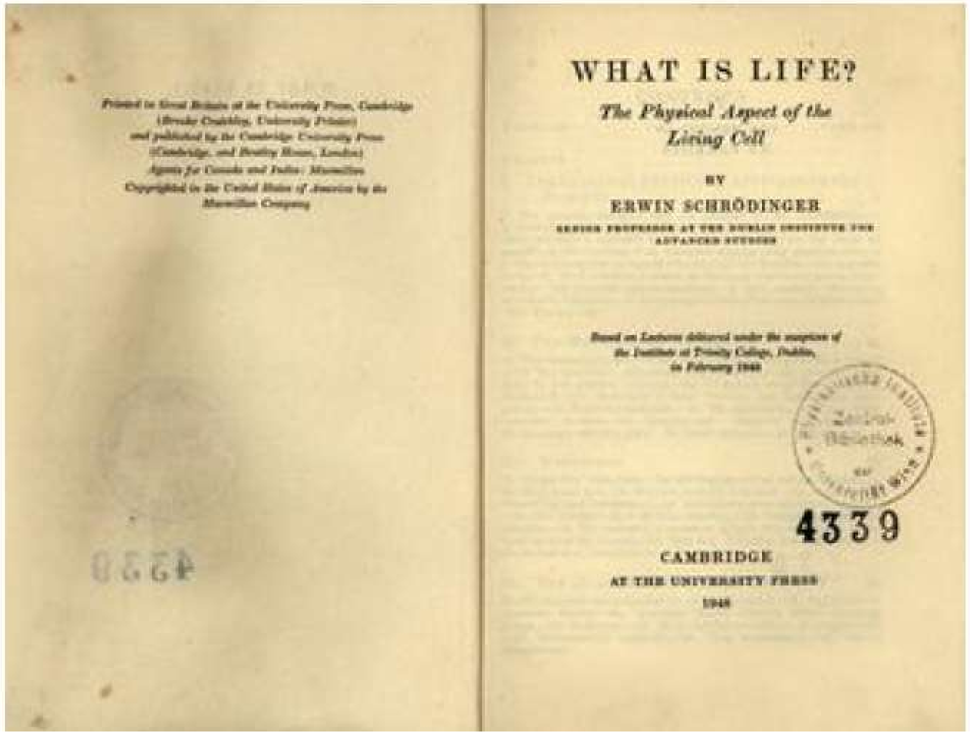
Inverse Problem: Modelling and forecasting the emergent dynamics of multiphysics and multiscale systems from DATA

- Both remain open problems!


Space-scale
Macroscale (PDEs)
$\frac{\partial \rho(r,t)}{\partial t} = g(\rho, \rho_r, \rho_{rr})$
$\rho(r,t) = \sum_{i=0}^{n} f_i(r,t)$
c_6 c_2 c_5 c_3 c_0 c_1 c_7 c_4 c_8
Mesoscale (Lattice-Boltzmann)
$f_i(r + c_i \Delta t, t_{k+1}) = f_i(r, t_k) + \Omega_i(r, t_k) + R_i(r, t_k)$
m_i
v_i
collision
transport
Microscale (particles)
$\frac{dr_i}{dt} = v_i$
$m_i \frac{dv_i}{dt} = F_i$
Time-scale

Schrödinger E. 1944 What Is Life? The Physical Aspect of the Living Cell.Cambridge University Press, Cambridge.

$$\hbar\frac{\partial}{\partial t}\Psi(\mathrm{r},t) = \hat{H}\Psi(\mathrm{r},t) \tag{1}$$



Printed in Great Britain at the University Press, Cambridge (Brooke Crutchley, University Printer) and published by the Cambridge University Press (Cambridge, and Bentley House, London) Agents for Canada and India: Macmillan Copyrighted in the United States of America by the Macmillan Company

WHAT IS LIFE?

The Physical Aspect of the Living Cell

BY

ERWIN SCHRÖDINGER

SENIOR PROFESSOR AT THE DUBLIN INSTITUTE FOR ADVANCED STUDIES

Based on Lectures delivered under the auspices of the Institute at Trinity College, Dublin, in February 1943

CAMBRIDGE

AT THE UNIVERSITY PRESS

1948

Comparison of Brains and Traditional Computers

200 billion neurons, 32 trillion synapses	64 billion bytes RAM but trillionsbytes
Element size: 10^{-6} m	Element size: 10^{-9} m
Energy use: 25W	Energy watt: ~100W (CPU)
Processing speed: 100 Hz	Processing speed: ~30^{9} Hz
Parallel, Distributed	Serial, Centralized
Fault Tolerant	Generally not Fault Tolerant
Learns: Yes	Learns: Some
Intelligent/Conscious: SomeTimes	Intelligent/Conscious: No

- 1943, McCulloch-Pitts neuron,
- 1949, Donald Hebb, The Organization of Behavior
- 1957, The Perceptron, Frank Rosenblatt
- 1959, Bernard Widrow and Marcian Hoff "ADALINE" and "MADALINE.
- 1970, Seppo Linnainmaa, Back Propagation, then Rumelhart et al.
- 1970-1985 Winter time
- 90s' ANNs Universal Approximation Theorems
- 2010- Deep Learning Era
- 2020- Generative AI

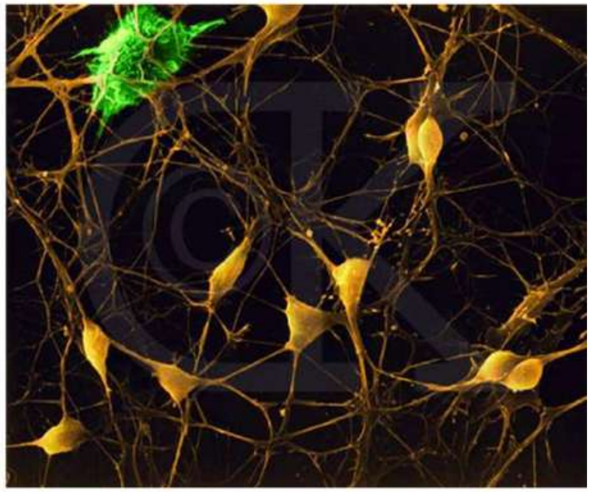
Neurons in the Brain

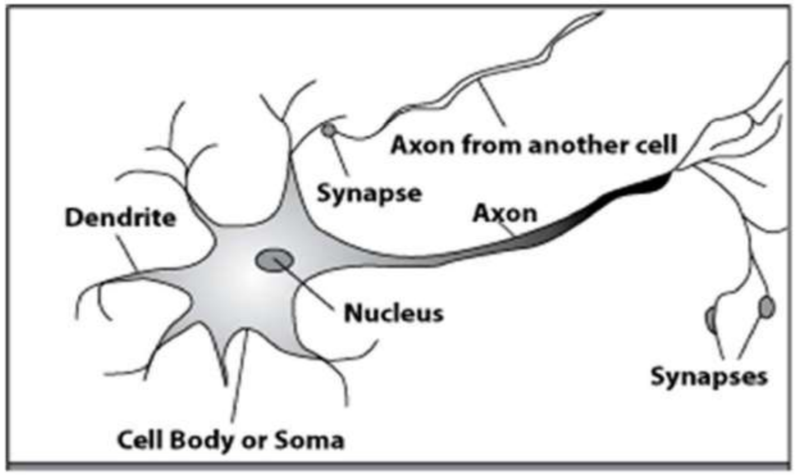
Although heterogeneous, at a low level the brain is composed of neurons

A neuron receives input from other neurons (generally thousands) from its synapses

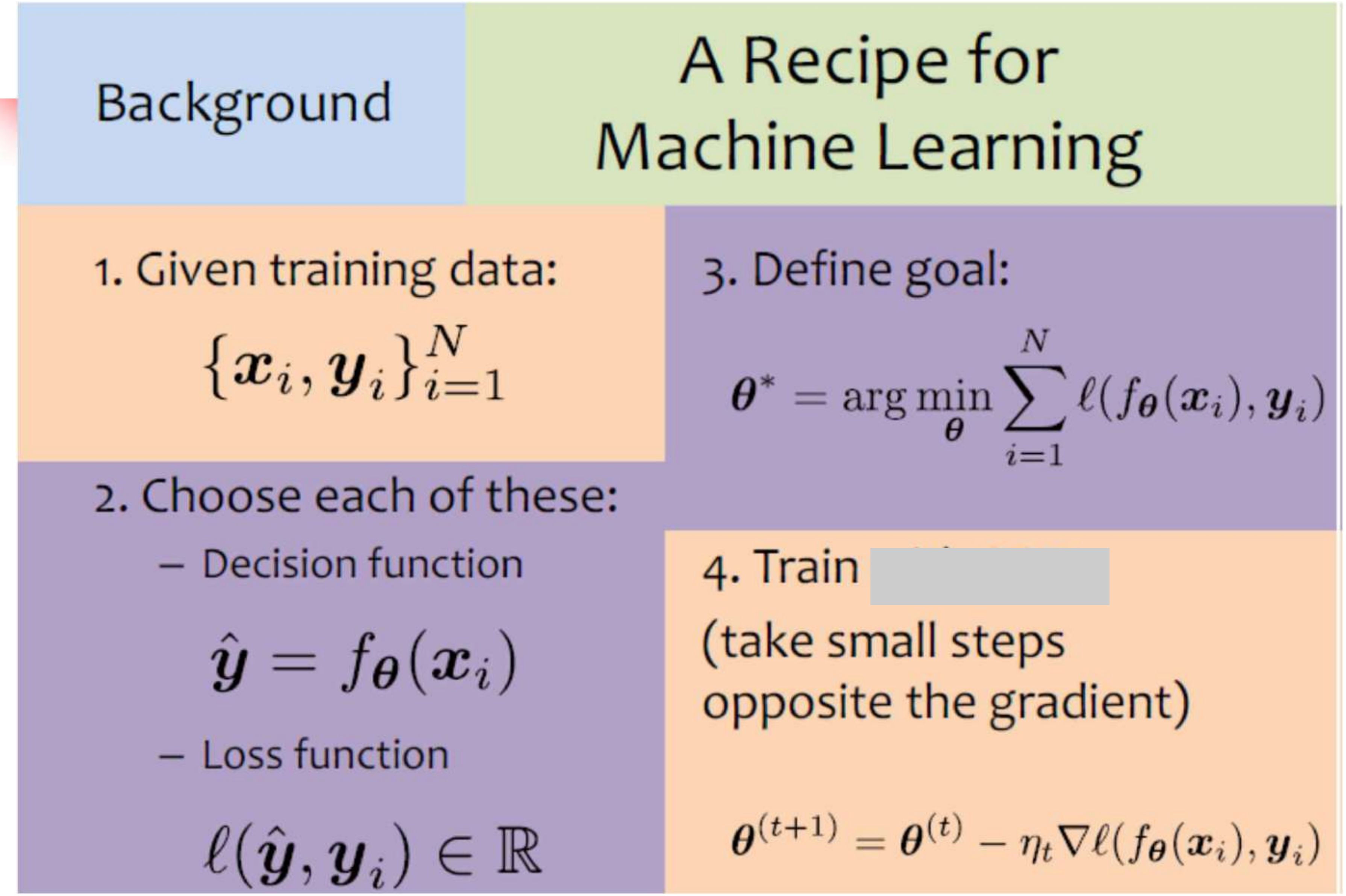
Inputs are approximately summed

When the input exceeds a threshold the neuron sends an electrical spike that travels from the body, down the axon, to the next neuron(s)





A neuron is connected to other neurons through about *10,000 synapses*



Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train [redacted]

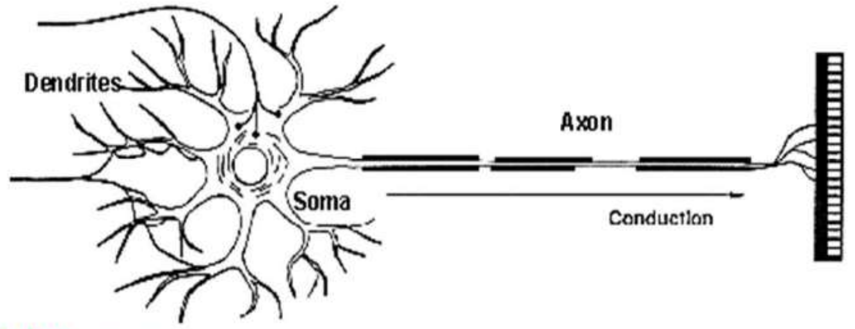
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

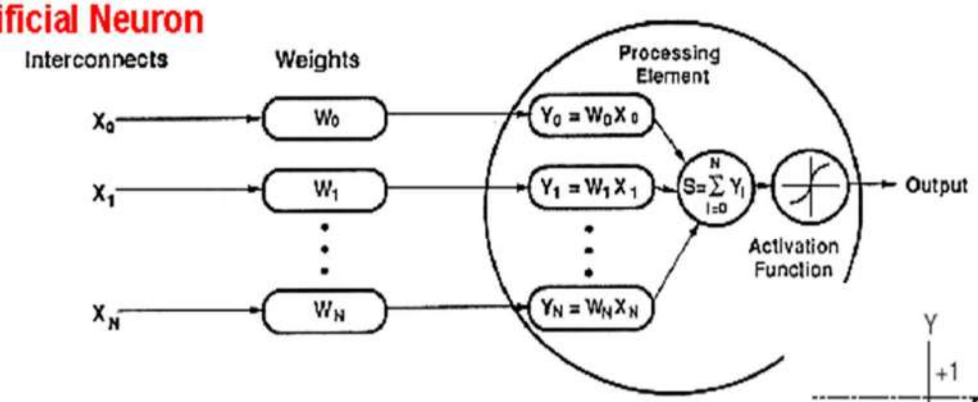
1943, McCulloch-Pitts neuron

Artificial Neuron : an imitation of a human neuron

Biological Neuron



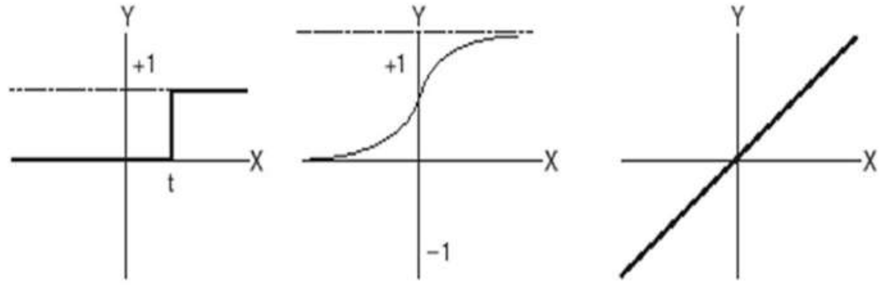
Artificial Neuron



Artificial neurons are based on biological neurons.

Each neuron in the network receives one or more inputs.

An **activation function** is applied to the inputs, which determines the output of the neuron – the activatior level.



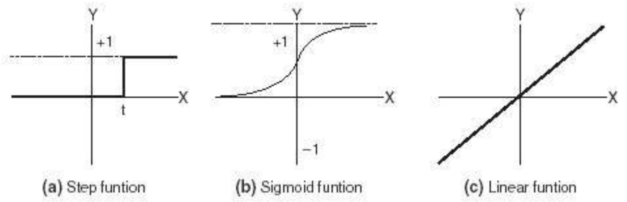
(a) Step funtion (b) Sigmoid funtion (c) Linear funtion

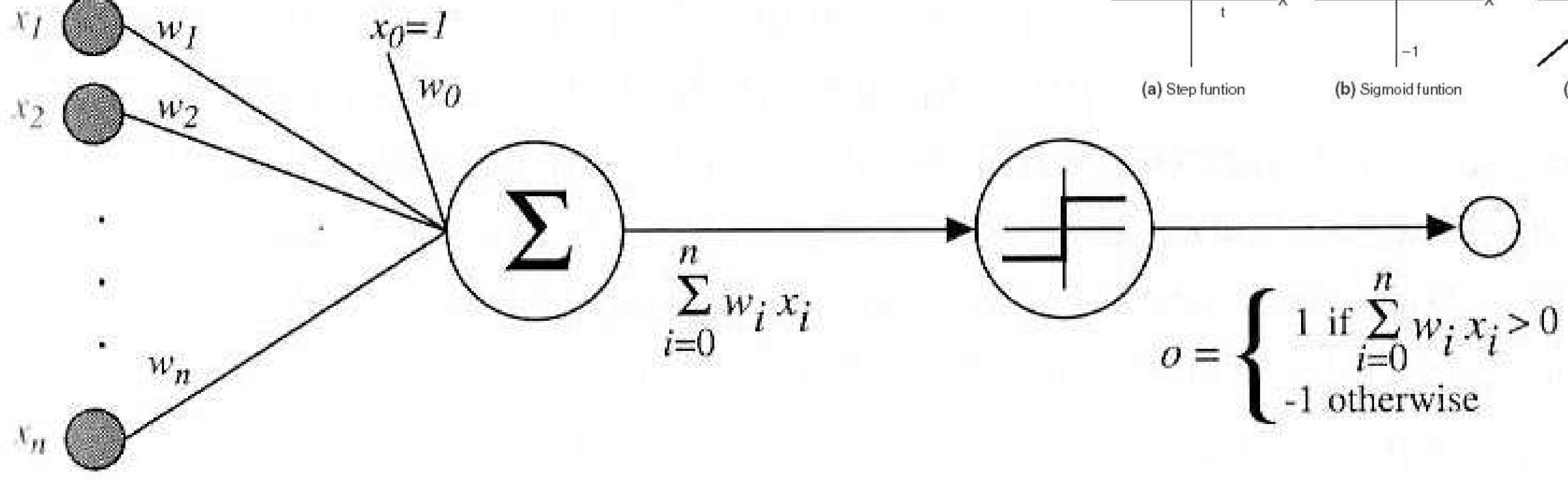
Perceptrons

1957, Frank Rosenblatt

A perceptron is a single neuron that classifies a set of inputs into one of two categories (usually 1 or -1).

The perceptron usually uses a step function, which returns 1 if the weighted sum of inputs exceeds a threshold, and 0 otherwise.





$$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$$

Learning Rule in ADALINE
is the LMS ("least mean squares")
1959, Bernard Widrow and Marcian
Hoff **Adaptive Linear Neuron**

Linear Perceptron

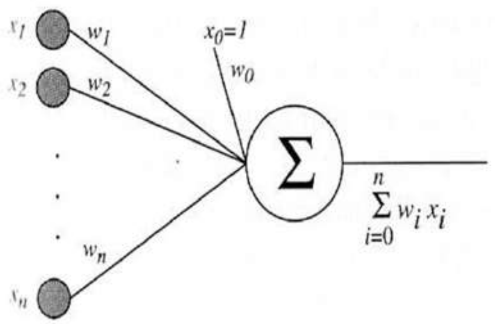
They are multivariate linear models:

$$\text{Out}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$$

And "training" consists of minimizing the sum-of-squared residuals by gradient descent.

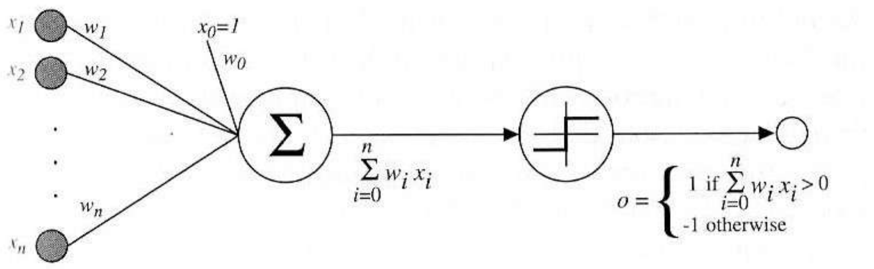
$$\mathrm{E} = \sum_k (\text{Out}(\mathrm{x}_k) - y_k)^2$$

$$= \sum_k \left(\mathrm{w}^T\mathrm{x}_k - y_k\right)^2$$



Gradient Descent in "n" Dimensions

Given $\mathrm{f}(\mathbf{w}) : \Re^{n} \rightarrow \Re$



x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

$$\nabla \mathrm{f}(\mathrm{w}) = \begin{pmatrix} \frac{\partial}{\partial w_1} \mathrm{f}(\mathrm{w}) \\ \vdots \\ \frac{\partial}{\partial w_n} \mathrm{f}(\mathrm{w}) \end{pmatrix}$$ points in direction of steepest ascent.

Maximum rate of change

GRADIENT DESCENT RULE: $\mathrm{w} \leftarrow \mathrm{w} - \eta \nabla \mathrm{f}(\mathrm{w})$

Equivalently

$$w_j \leftarrow w_j - \eta \frac{\partial}{\partial w_j} \mathrm{f}(\mathrm{w})$$ ....where w_j is the jth variable

Linear Perceptron Training Rule

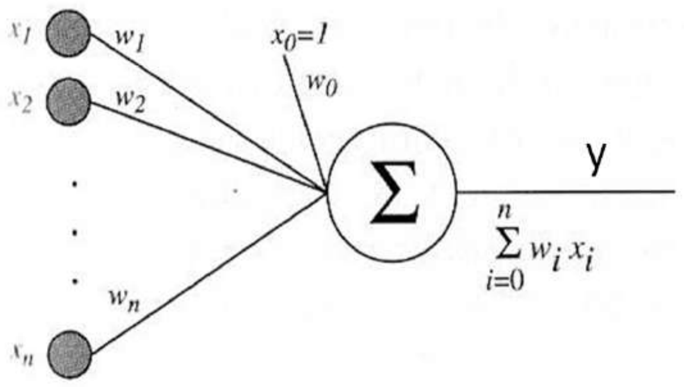
$$E = \sum_{k=1}^{R} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

R: Input-Output patterns k = 1, …, R.

Gradient descent tells us how we should update **w** to minimize *E:*

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

So what's $\frac{\partial E}{\partial w_j}$?



Linear Perceptron Training Rule

$$E = \sum_{k=1}^{R} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

Gradient descent tells us we should update **w** to minimize *E:*

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

So what's $\frac{\partial E}{\partial w_j}$?

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{R} \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

$$= \sum_{k=1}^{R} 2(y_k - \mathbf{w}^T \mathbf{x}_k) \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \mathbf{w}^T \mathbf{x}_k$$

...where...

$$\delta_k = y_k - \mathbf{w}^T \mathbf{x}_k$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \sum_{i=1}^{n} w_i x_{ki}$$

$$= -2 \sum_{k=1}^{R} \delta_k x_{kj}$$

Linear Perceptron Training Rule

$$E = \sum_{k=1}^{R} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

Gradient descent tells us how we should update w to minimize *E:*

$$\left.\begin{aligned} & w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j} \\ & \text{...where...} \\ & \frac{\partial E}{\partial w_j} = -2 \sum_{k=1}^{R} \delta_k x_{kj} \end{aligned}\right\}$$

$$\boxed{w_j \leftarrow w_j + 2\eta \sum_{k=1}^{R} \delta_k x_{kj}}$$

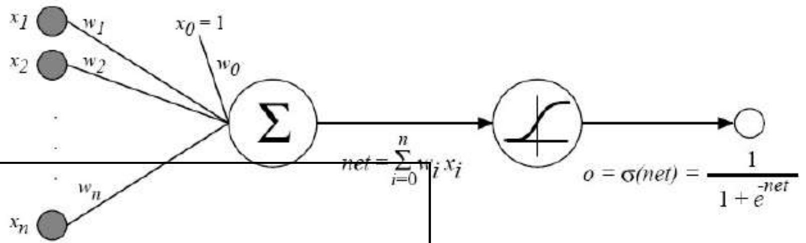
$$\delta_k = y_k - \mathbf{w}^T \mathbf{x}_k$$

The linear perceptron algorithm

```
1) Randomly initialize weights $w_1\ w_2\ \ldots\ w_m$

2) Get your training dataset
    3) for $i = 1$ to R
          $\delta_i := y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i$
    4) for $j = 1$ to m
          $w_j \leftarrow w_j + \eta \sum_{i=1}^{R} \delta_i x_{ij}$
5) if $\sum \delta_i^2$ stop. Else go to 3.
```

Gradient descent with sigmoid on a perceptron



x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$

First, notice $g'(x) = g(x)(1-g(x))$

Because: $g(x) = \frac{1}{1+e^{-x}}$ so $g'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$

$$= \frac{1-1+e^{-x}}{(1+e^{-x})^2} = \frac{-1}{(1+e^{-x})^2} + \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = g(x)(1-g(x))$$

$$\text{Out(x)} = g\left(\sum_k w_k x_k\right)$$

$$E = \sum_i \left(y_i - g\left(\sum_k w_k x_{ik}\right)\right)^2$$

$$\frac{\partial E}{\partial w_j} = \sum_i 2\left(y_i - g\left(\sum_k w_k x_{ik}\right)\right)\left(-\frac{\partial}{\partial w_j} g\left(\sum_k w_k x_{ik}\right)\right)$$

$$= \sum_i -2\left(y_i - g\left(\sum_k w_k x_{ik}\right)\right) g'\left(\sum_k w_k x_{ik}\right) \frac{\partial}{\partial w_j} \sum_k w_k x_{ik}$$

$$= \sum_i -2\delta_i g(\text{net}_i)(1-g(\text{net}_i)) x_{ij}$$

where $\delta_i = y_i - \text{Out}(x_i)$ $\quad \text{net}_i = \sum_k w_k x_k$

The sigmoid perceptron update rule:

$$w_j \leftarrow w_j + \eta \sum_{i=1}^{R} \delta_i g_i (1 - g_i) x_{ij}$$

where $\quad g_i = g\left(\sum_{j=1}^{m} w_j x_{ij}\right)$

$$\delta_i = y_i - g_i$$

Multilayer Networks: **Deep Learning**

Single-layer artificial neural networks, have limitations in terms of the types of functions they can approximate.

$$\mathrm{Out}(\mathrm{x}) = g\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) = g\left(\sum_j w_j x_j\right)$$

Use a wider representation !

$$\mathrm{Out}(\mathrm{x}) = g\left(\sum_j W_j g\left(\sum_k w_{jk} x_{jk}\right)\right)$$

This is a nonlinear function
Of a linear combination
Of non linear functions
Of linear combinations of inputs

1970, Seppo Linnainmaa, Back Propagation, then Rumelhart et al.

Training the ANN: Backpropagation algorithm

$$\mathrm{Out}(\mathrm{x}) = g\left(\sum_j W_j g\left(\sum_k w_{jk} x_k\right)\right)$$

Find a set of weights $\{W_j\}, \{w_{jk}\}$

to minimize

$$\sum_i \left(y_i - \mathrm{Out}(\mathrm{x}_i)\right)^2$$

by automatic differentiation (chain rule)

That's it!

That's the backpropagation algorithm.

Backpropagation Algorithm ***(with GD)***

Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.

Initialize all network weights to small random numbers

```
Until termination condition is met, Do
    **For each <x,t> in training examples**, Do
        *Propagate the input forward through the network:*
        1. **Present Input x** to the network and compute the output $O_u$ of every u in the network
        ***Propagate the errors backward*** *through the network:*
        2. For each network output $k=1,2,\ldots,n_{out}$ calculate its error term $\delta_k$
            $\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$
        3. For each hidden unit $h=1,2,\ldots n_{hidden}$, calculate its error term $\delta_h$
            $\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{kh}\delta_k$
        4. Update each network weight $w_{ji}$
            $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$
```

Backpropagation Learning

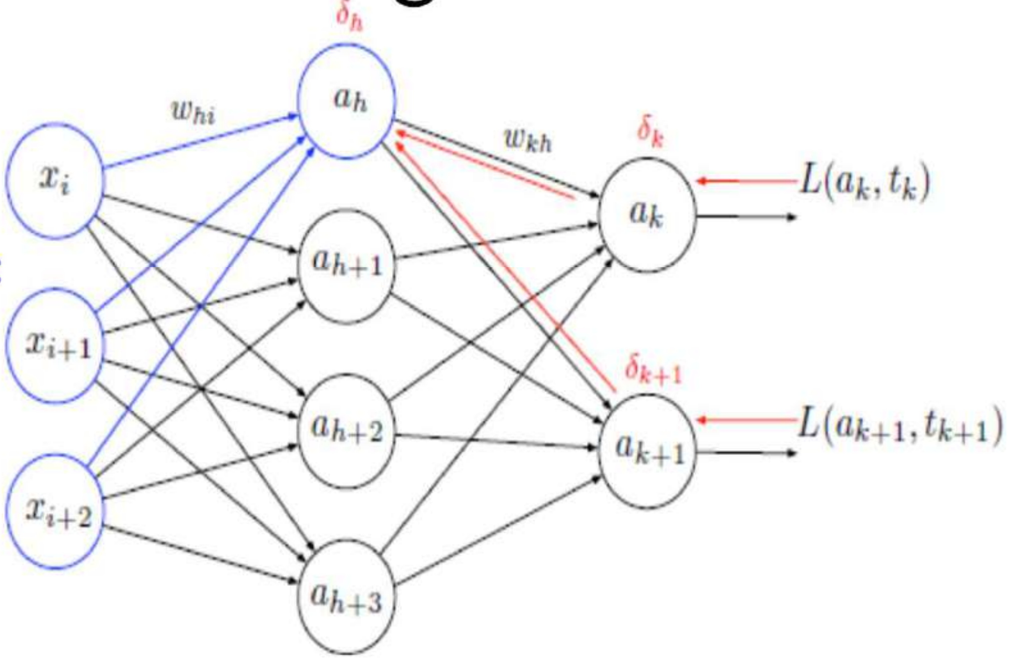
For each <*x*,*t*> in training examples

Minimize

$$L = \frac{1}{2}\sum_{k\in O}(t_k - a_k)^2$$

Gradient Descent Rule:

$$\Delta w_{ji} = -\alpha \frac{\partial L}{\partial w_{ji}}$$



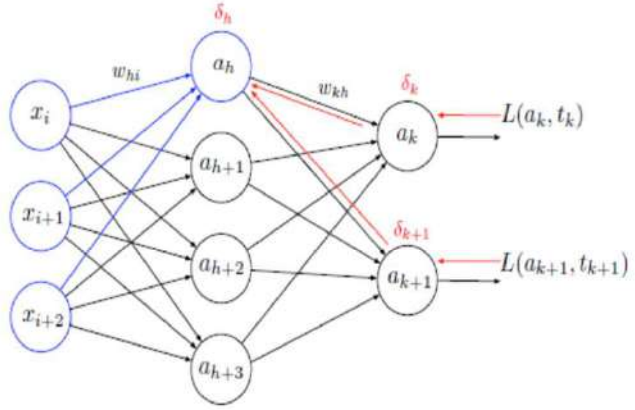
We use the following notation:

- x_{ji} – the ith input to unit j
- w_{ji} – the weight associated with ith input to unit j
- z_j – the weighted sum of input for unit j, i.e. $z_j = \sum_i w_{ji}x_{ji}$
- a_j – the output computed by unit j, i.e. $a_j = g(z_j)$ where g is an activation function (sigmoid here)

Backpropagation Learn

We use the following notation:

$$L = \frac{1}{2}\sum_{k \in O}(t_k - a_k)^2$$



- x_{ji} – the ith input to unit j
- w_{ji} – the weight associated with ith input to unit j
- z_j – the weighted sum of input for unit j, i.e. $z_j = \sum_i w_{ji} x_{ji}$
- a_j – the output computed by unit j, i.e. $a_j = g(z_j)$ where g is an activation function (sigmoid here)

Notice first that weight w_{ji} can influence the network's output only through z_j.

$$\Delta w_{ji} = -\alpha \frac{\partial L}{\partial w_{ji}}$$

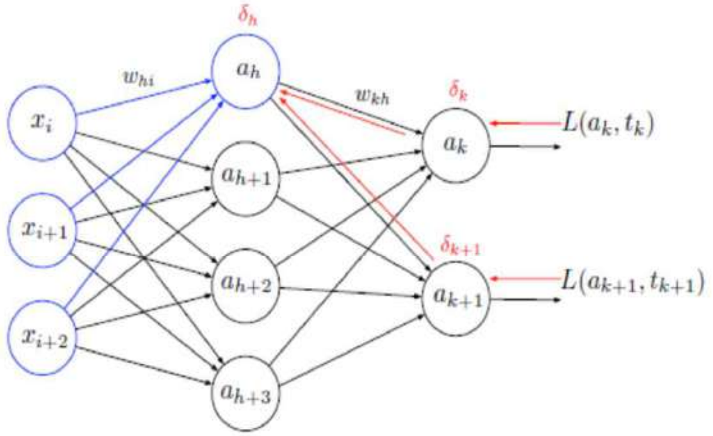
$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}} = \frac{\partial L}{\partial z_j} x_{ji}$$

$$\frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial a_j}\frac{\partial a_j}{\partial z_j} = \frac{\partial L}{\partial a_j} a_j(1 - a_j)$$

$$\begin{aligned}
\frac{\partial L}{\partial a_j} &= \frac{\partial}{\partial a_j}\frac{1}{2}\sum_{k \in O}(t_k - a_k)^2 \\
&= \frac{\partial}{\partial a_j}\frac{1}{2}(t_j - a_j)^2 \\
&= \frac{1}{2}2(t_j - a_j)\frac{\partial}{\partial a_j}(t_j - a_j) \\
&= -(t_j - a_j)
\end{aligned}$$

$$\frac{\partial L}{\partial z_j} = -(t_j - a_j)a_j(1 - a_j) = \delta_j$$

Backpropagation



We use the following notation:

- x_{ji} – the ith input to unit j
- w_{ji} – the weight associated with ith input to unit j
- z_j – the weighted sum of input for unit j, i.e. $z_j = \sum_i w_{ji} x_{ji}$
- a_j – the output computed by unit j, i.e. $a_j = g(z_j)$ where g is an activation function (sigmoid here)

Notice first that weight w_{ji} can influence the network's output only through z_j .

Case 1: j is an output unit

Example : αk

$$\Delta w_{ji} = -\alpha \frac{\partial L}{\partial w_{ji}}$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}} = \frac{\partial L}{\partial z_j} x_{ji}$$

$$\frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial a_j}\frac{\partial a_j}{\partial z_j} = \frac{\partial L}{\partial a_j} a_j(1-a_j)$$

$$\begin{aligned}\frac{\partial L}{\partial a_j} &= \frac{\partial}{\partial a_j}\frac{1}{2}\sum_{k\in O}(t_k - a_k)^2 \\ &= \frac{\partial}{\partial a_j}\frac{1}{2}(t_j - a_j)^2 \\ &= \frac{1}{2}2(t_j - a_j)\frac{\partial}{\partial a_j}(t_j - a_j) \\ &= -(t_j - a_j)\end{aligned}$$

$$\frac{\partial L}{\partial z_j} = -(t_j - a_j)a_j(1-a_j) = \delta_j$$

$$\begin{aligned}\frac{\partial L}{\partial w_{kh}} &= \frac{\partial L}{\partial z_k}\frac{\partial z_k}{\partial w_{kh}} \\ &= \frac{\partial L}{\partial z_k} a_h \\ &= -(t_k - a_k)a_k(1-a_k)a_h \\ &= \delta_k a_h\end{aligned}$$

Finally: $\Delta w_{kh} = -\alpha \frac{\partial L}{\partial w_{kh}} = -\alpha \delta_k a_h$

Backpropagation Learning

We use the following notation:

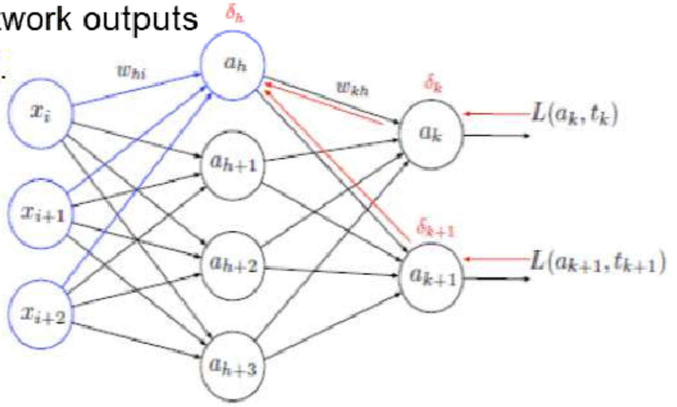
- x_{ji} – the ith input to unit j
- w_{ji} – the weight associated with ith input to unit j
- z_j – the weighted sum of input for unit j, i.e. $z_j = \sum_i w_{ji} x_{ji}$
- a_j – the output computed by unit j, i.e. $a_j = g(z_j)$ where g is an activation function (sigmoid here)

Notice first that weight **Wji** can influence the network's output only through Zj .

Case 2: j is a hidden unit

When **unit j is an internal unit** we must also consider every unit immediately downstream of unit j, i.e. all units whose direct input include the output of unit j **(variations in hidden unit activations affect outputs)**

This is because a change in Wji (and therefore in Zj) influences the network outputs only through these units. Let *ds(j)* denote units downstream of unit j.



$$\begin{aligned}\frac{\partial L}{\partial z_j} &= \sum_{k \in ds(j)} \frac{\partial L}{\partial z_k} \frac{\partial z_k}{\partial z_j} \\ &= \sum_{k \in ds(j)} \delta_k \frac{\partial z_k}{\partial a_j} \frac{\partial a_j}{\partial z_j} \\ &= \sum_{k \in ds(j)} \delta_k w_{kj} a_j (1 - a_j)\end{aligned}$$

$$\frac{\partial L}{\partial z_j} = -(t_j - a_j) a_j (1 - a_j) = \delta_j$$

use δ_j to denote $\frac{\partial L}{\partial z_j}$

$$\delta_j = a_j (1 - a_j) \sum_{k \in ds(j)} \delta_k w_{kj}$$

Backpropagation Learning

We use the following notation:

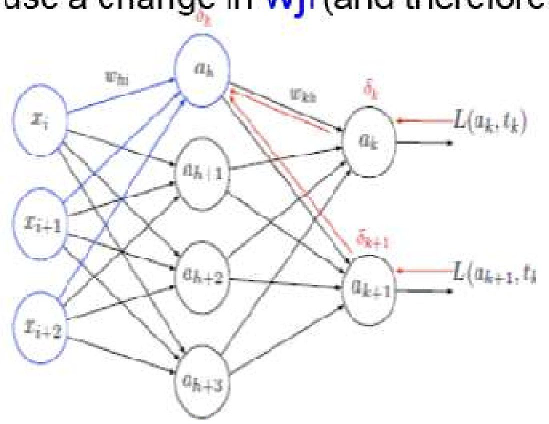
- x_{ji} – the ith input to unit j
- w_{ji} – the weight associated with ith input to unit j
- z_j – the weighted sum of input for unit j, i.e. $z_j = \sum_i w_{ji} x_{ji}$
- a_j – the output computed by unit j, i.e. $a_j = g(z_j)$ where g is an activation function (sigmoid here)

When **unit j is an internal unit** we must also consider every unit immediately downstream of unit j, i.e. all units whose direct input include the output of unit j.

This is because a change in wji (and therefore in zj) influences the network outputs only through these units.

Example: a_h

$$\begin{aligned}\frac{\partial L}{\partial z_j} &= \sum_{k \in ds(j)} \frac{\partial L}{\partial z_k}\frac{\partial z_k}{\partial z_j}\\ &= \sum_{k \in ds(j)} \delta_k \frac{\partial z_k}{\partial a_j}\frac{\partial a_j}{\partial z_j}\\ &= \sum_{k \in ds(j)} \delta_k w_{kj} a_j(1-a_j)\end{aligned}$$



$$\begin{aligned}\frac{\partial L}{\partial w_{hi}} &= \frac{\partial L}{\partial z_h}\frac{\partial z_h}{\partial w_{hi}}\\ &= \sum_{k \in ds(h)} \frac{\partial L}{\partial z_k}\frac{\partial z_k}{\partial z_j} x_i\\ &= a_h(1-a_h) \sum_{k \in ds(h)} \delta_k w_{kh} x_i\\ &= \delta_h x_i\end{aligned}$$

use δ_j to denote $\frac{\partial L}{\partial z_j}$

$$\Delta w_{hi} = -\alpha \frac{\partial L}{\partial w_{hi}} = -\alpha \delta_h x_i$$

Improving Gradient Descent

Gauss-Newton's method

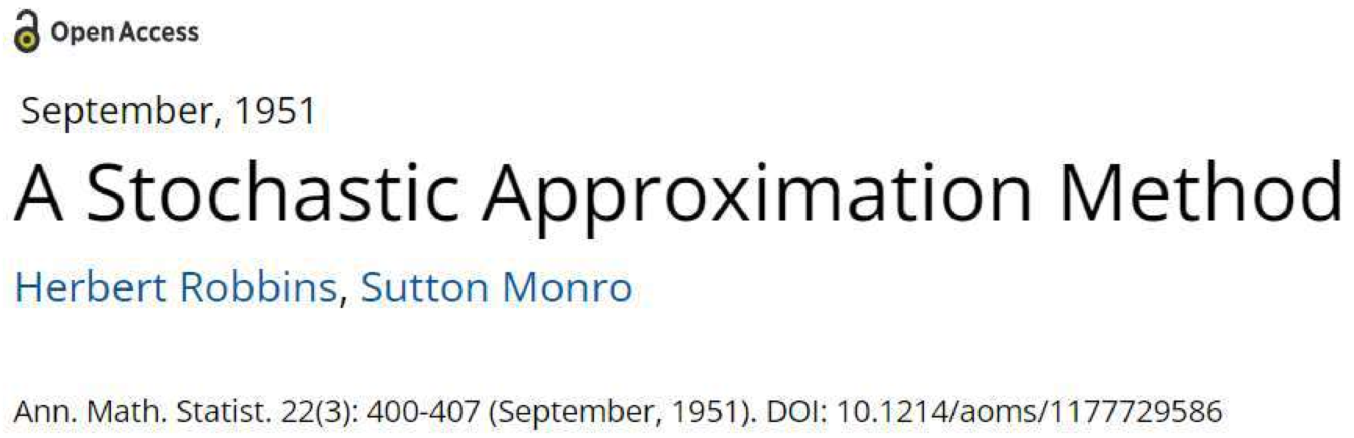
$$E(\mathbf{w}+\mathbf{h}) = E(\mathbf{w}) + \mathbf{h}^T \frac{\partial E}{\partial \mathbf{w}} + \frac{1}{2}\mathbf{h}^T \frac{\partial^2 E}{\partial \mathbf{w}^2}\mathbf{h} + O(|\mathbf{h}|^3)$$

If we neglect the *$O(h^3)$* terms, this is a ***quadratic form***

$$\mathbf{w} \leftarrow \mathbf{w} - \left[\frac{\partial^2 E}{\partial \mathbf{w}^2}\right]^{-1} \frac{\partial E}{\partial \mathbf{w}}$$

This should send us directly to the global minimum if the function is truly quadratic.

Stochastic GD



Open Access

September, 1951

A Stochastic Approximation Method

Herbert Robbins, Sutton Monro

Ann. Math. Statist. 22(3): 400-407 (September, 1951). DOI: 10.1214/aoms/1177729586

```
Choose an initial vector of parameters  w and learning rate

Repeat until an approximate minimum is obtained:

Randomly shuffle samples in the training set

Apply GD

End Until
```

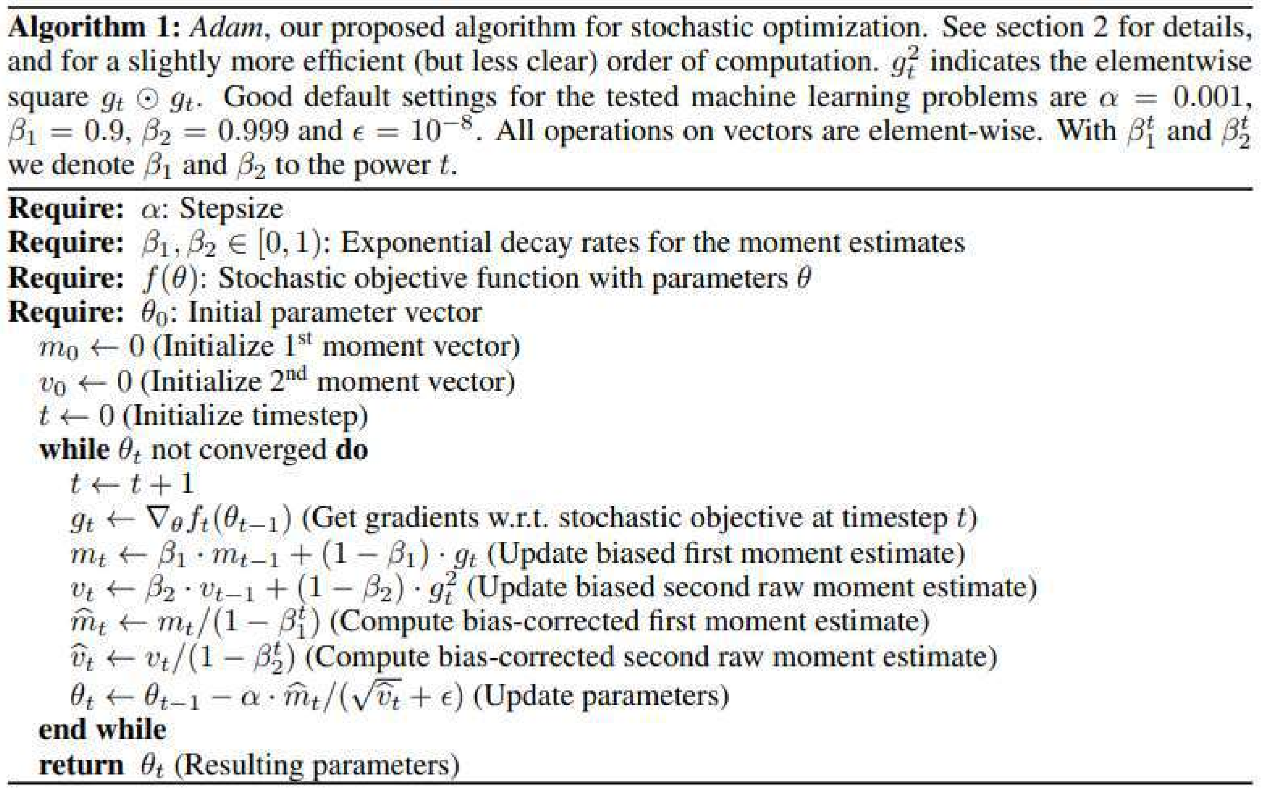
Published as a conference paper at ICLR 2015

Adam: A Method for Stochastic Optimization

Diederik P. Kingma[*]
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba[*]
University of Toronto
jimmy@psi.utoronto.ca

stochastic optimization requiring first-order gradients computing individual adaptive learning rates from estimates of first and second moments of the gradients; the name Adam is derived from adaptive moment estimation.

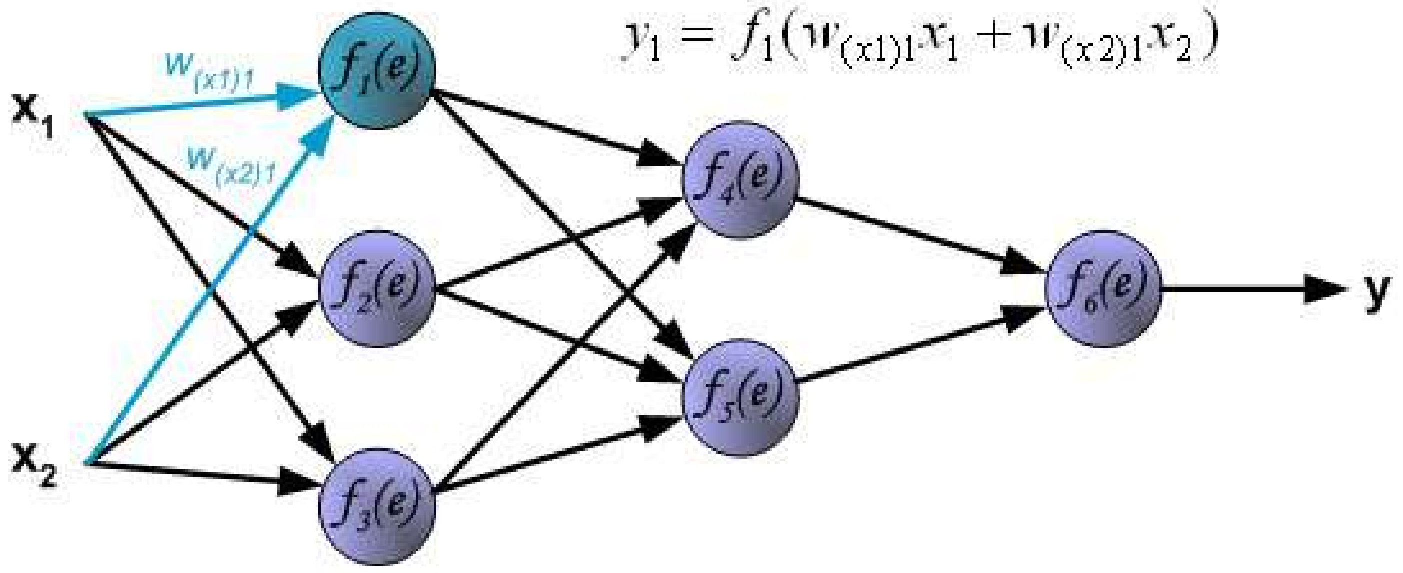


Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

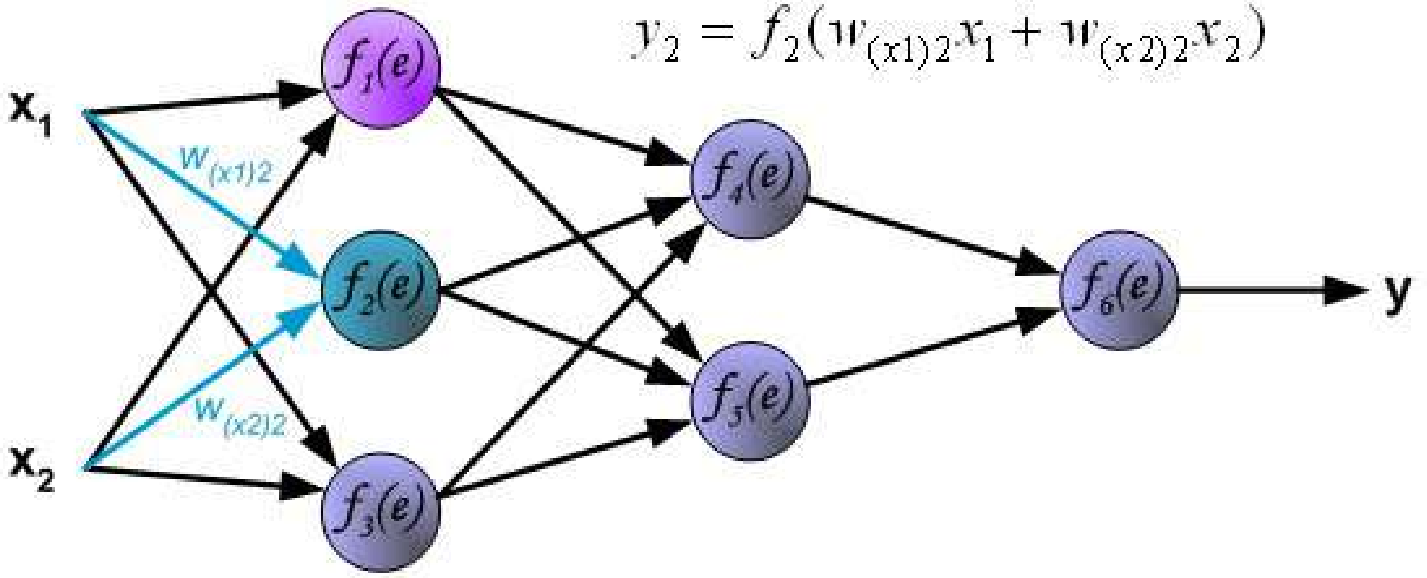
```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Learning Algorithm: Backpropagation

Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.

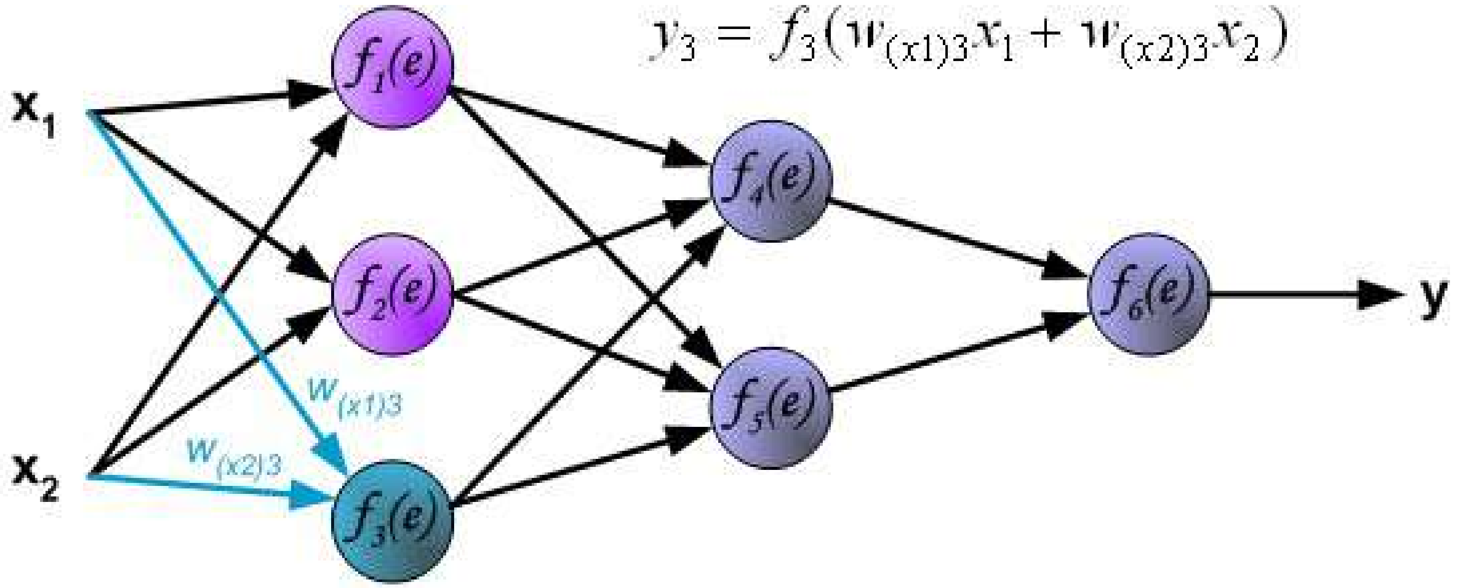


Backpropagation



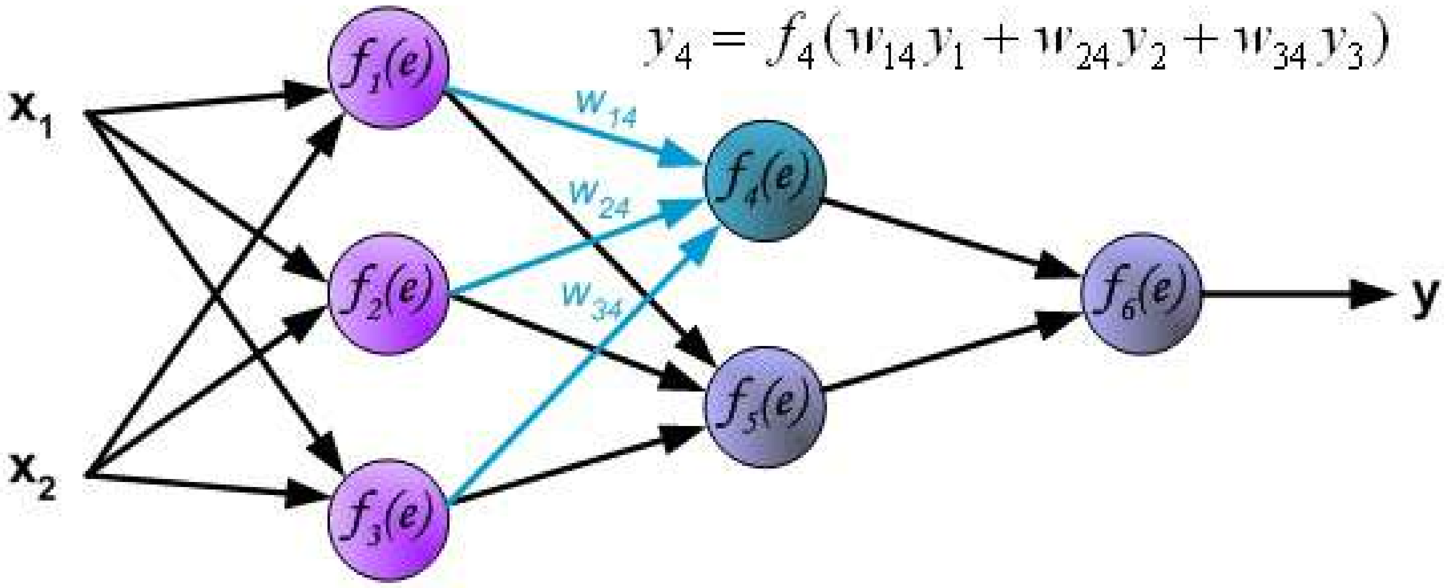
Backpropagation

$$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$$



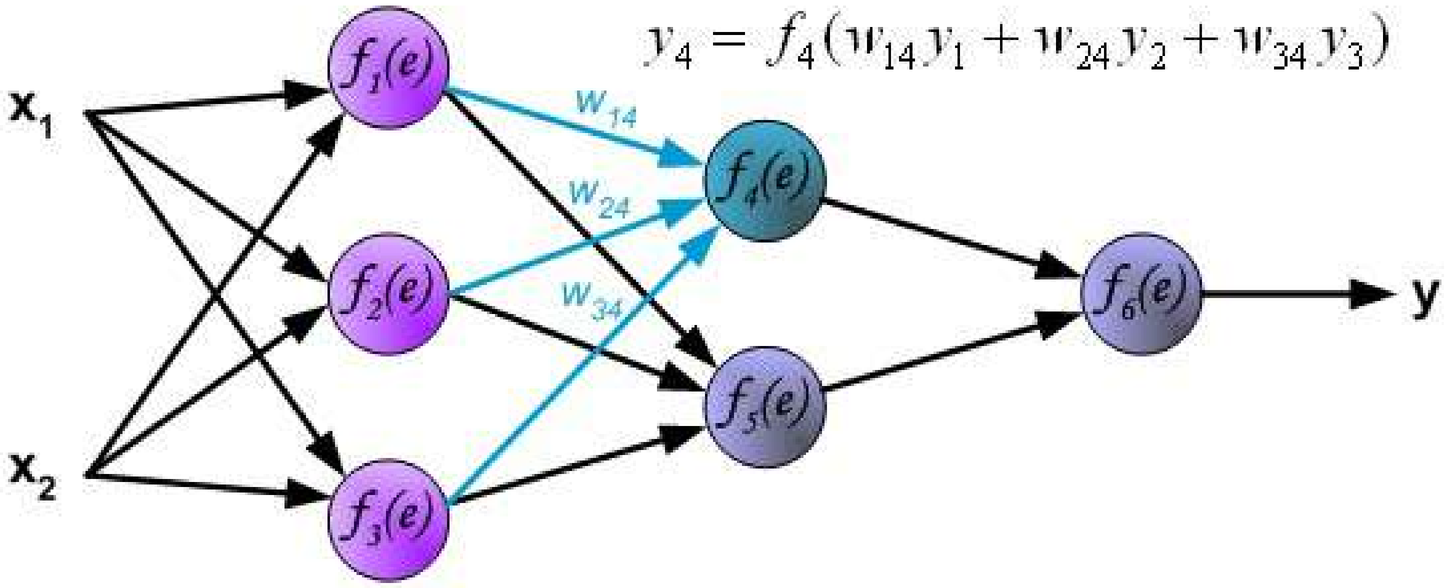
Backpropagation

Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron *m* and input of neuron *n* in the next layer.

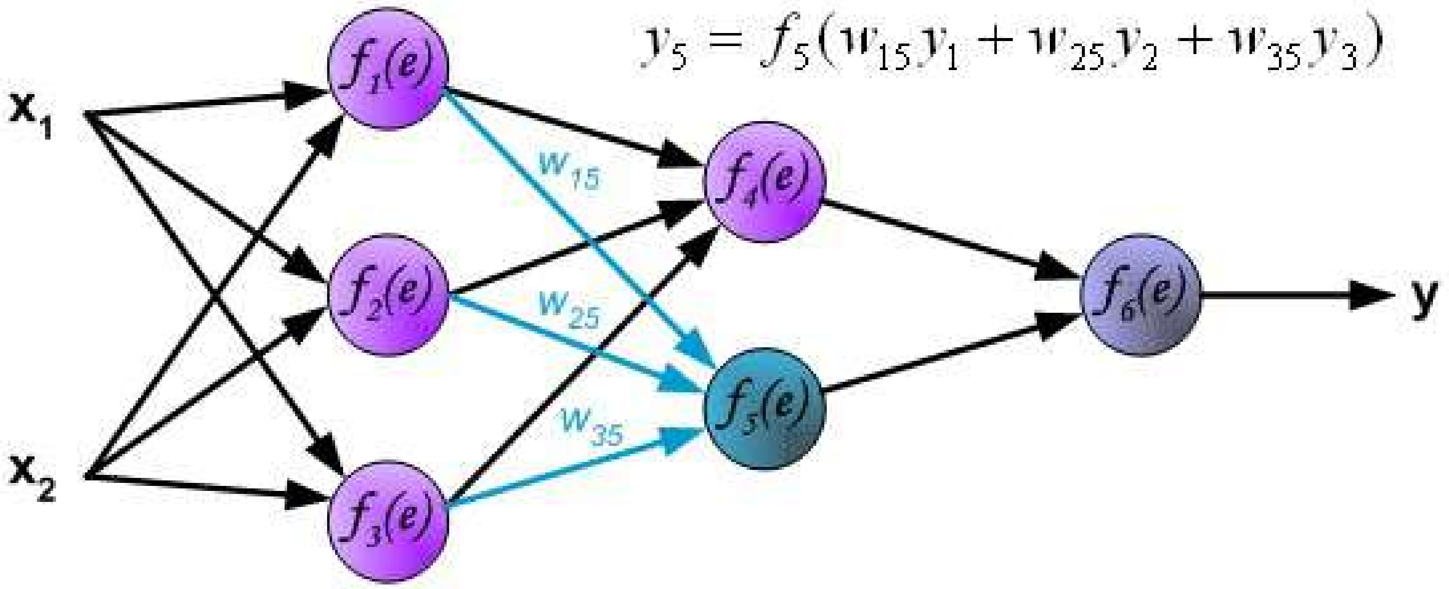


Backpropagation

$$y_4 = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3)$$

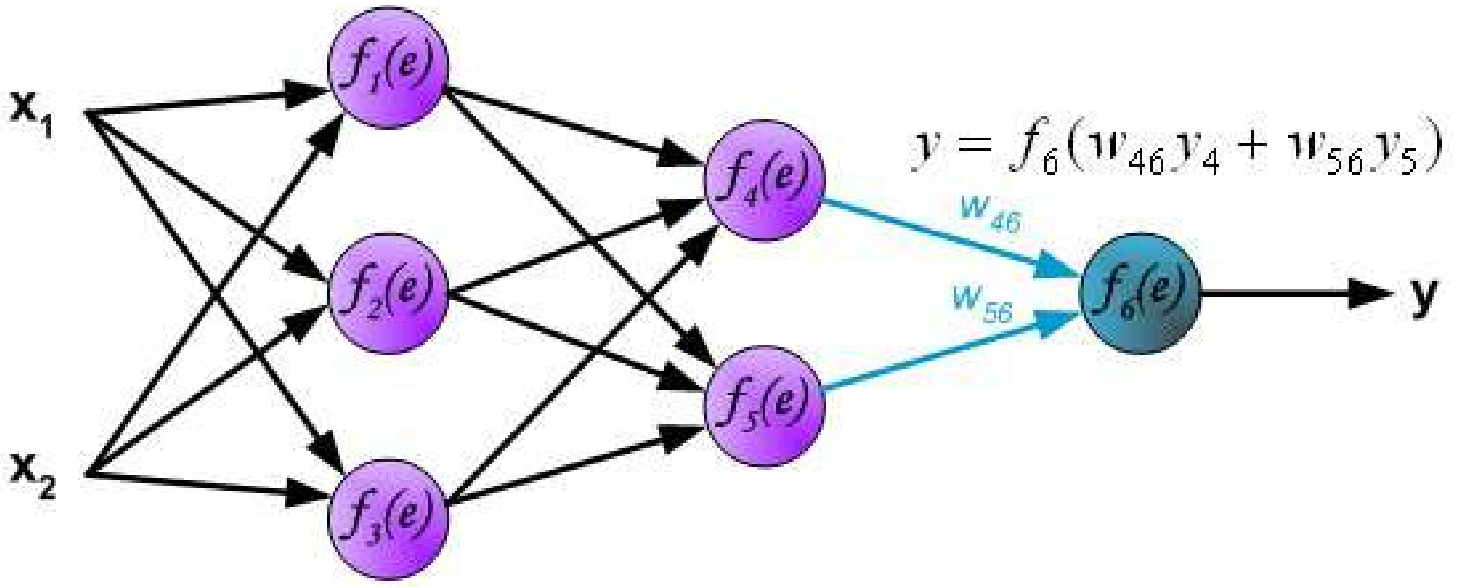


Backpropagation



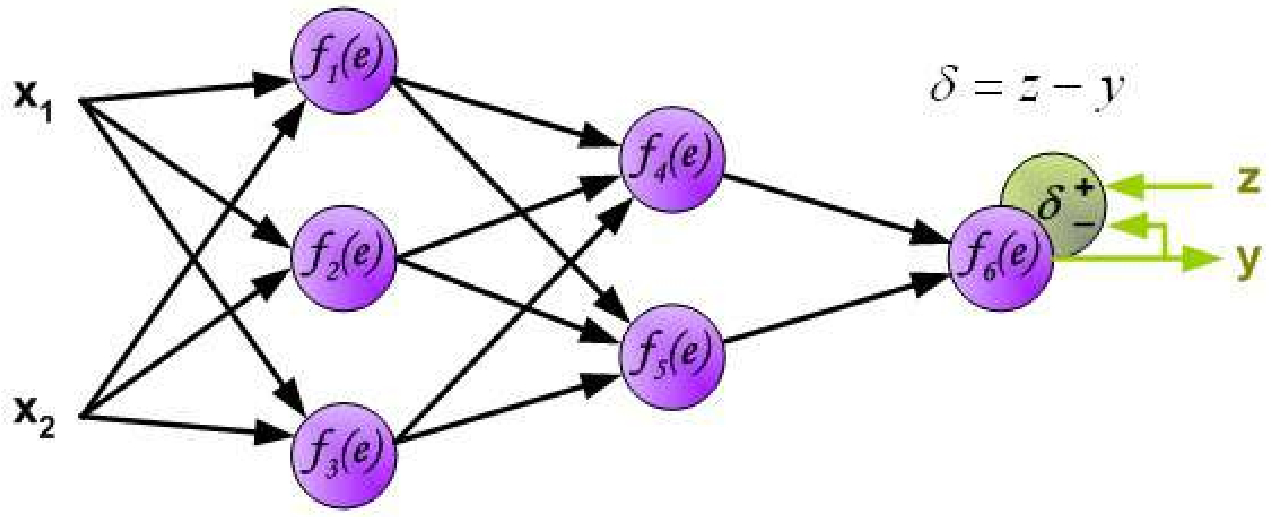
Backpropagation

Propagation of signals through the output layer.



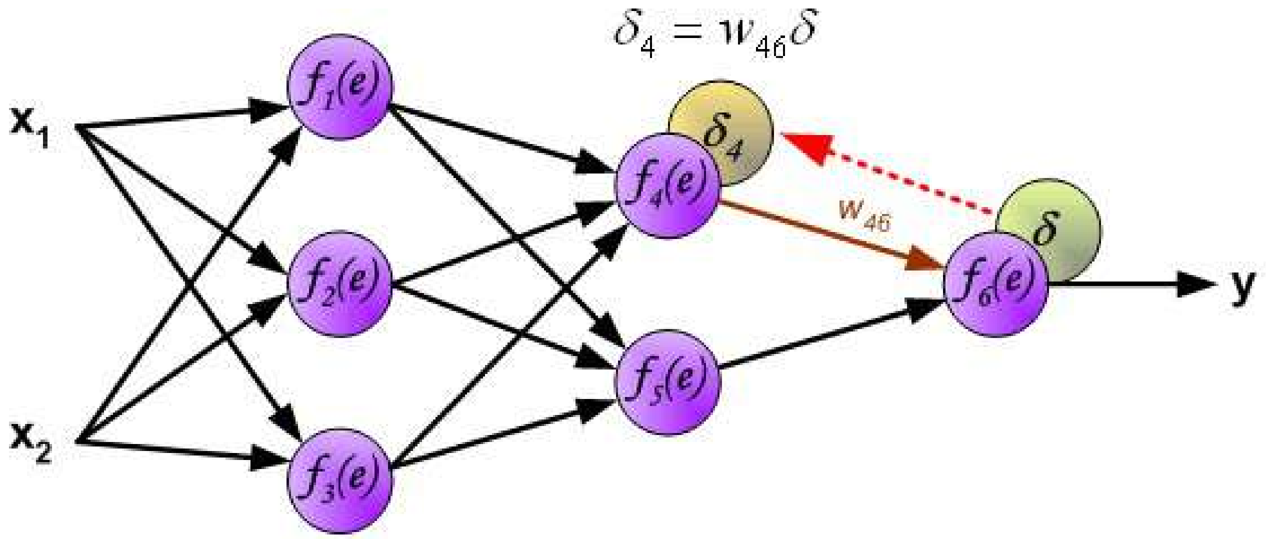
Backpropagation

In the next step the output signal of the network *y* is compared with the desired output value (the target), in training data set. The difference is called error signal *δ* of output layer neuron



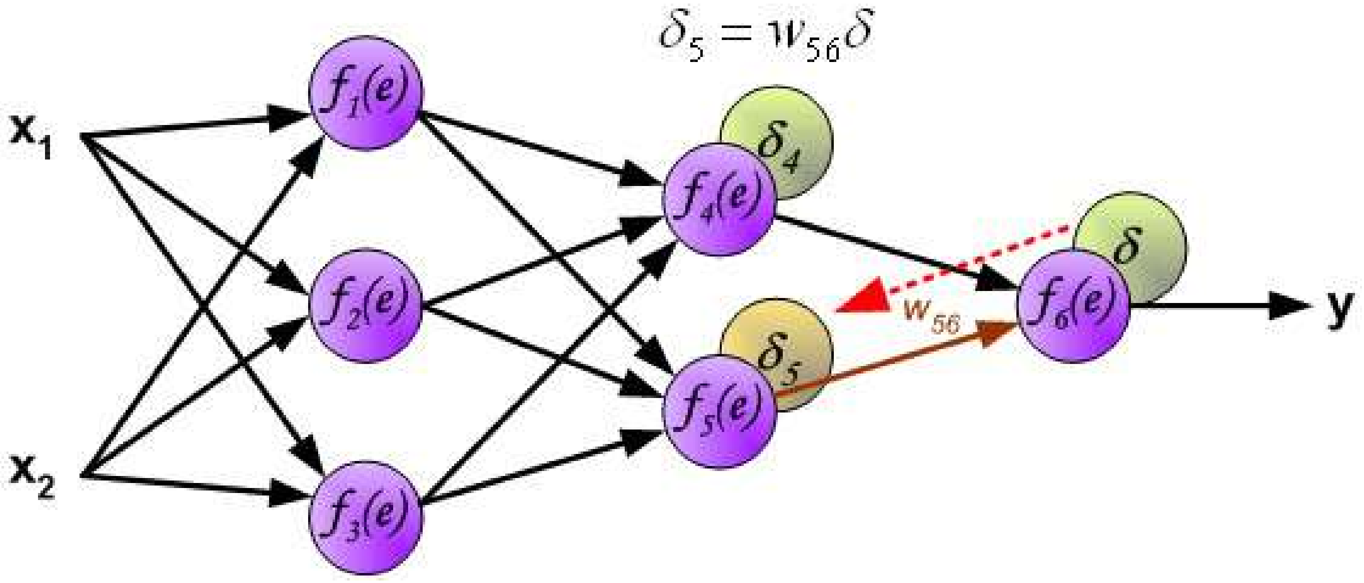
Backpropagation

The idea is to propagate the error signal *δ* (computed in the single teaching step) back to all neurons, which output signals were input to the reference neuron.



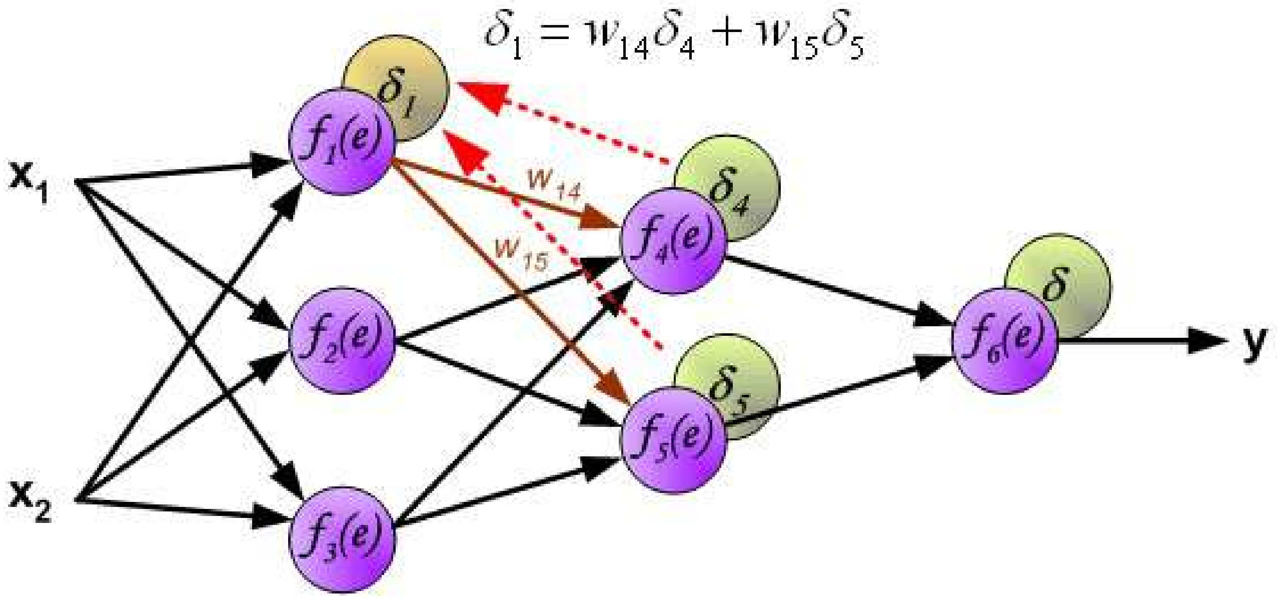
Backpropagation

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input to the reference neuron.



Backpropagation

The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:

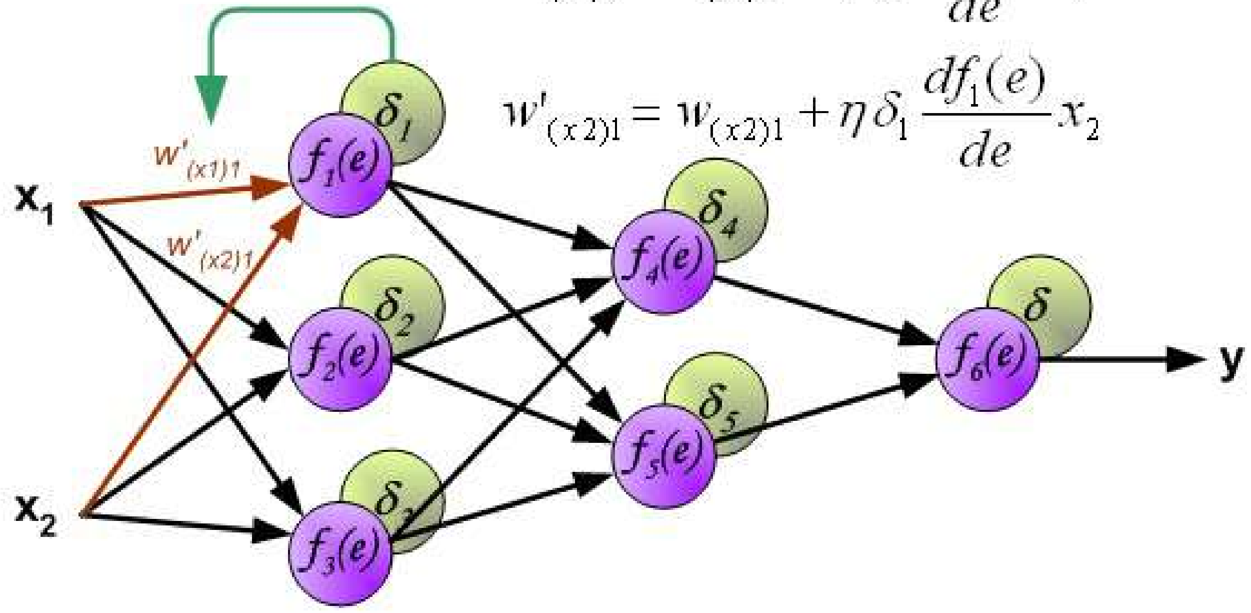


Backpropagation

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$

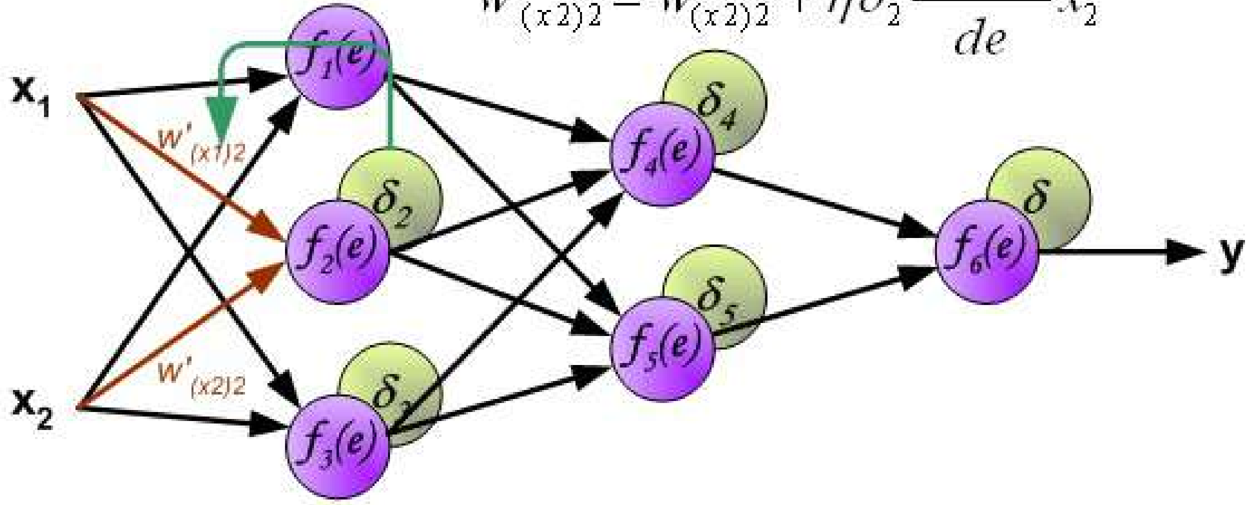


Backpropagation

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

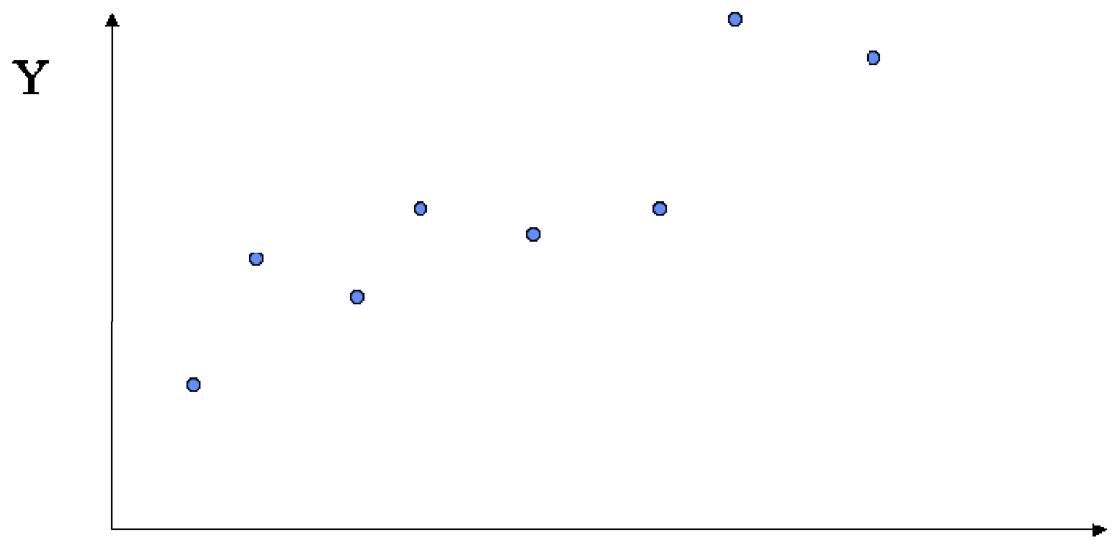
$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



Generalization

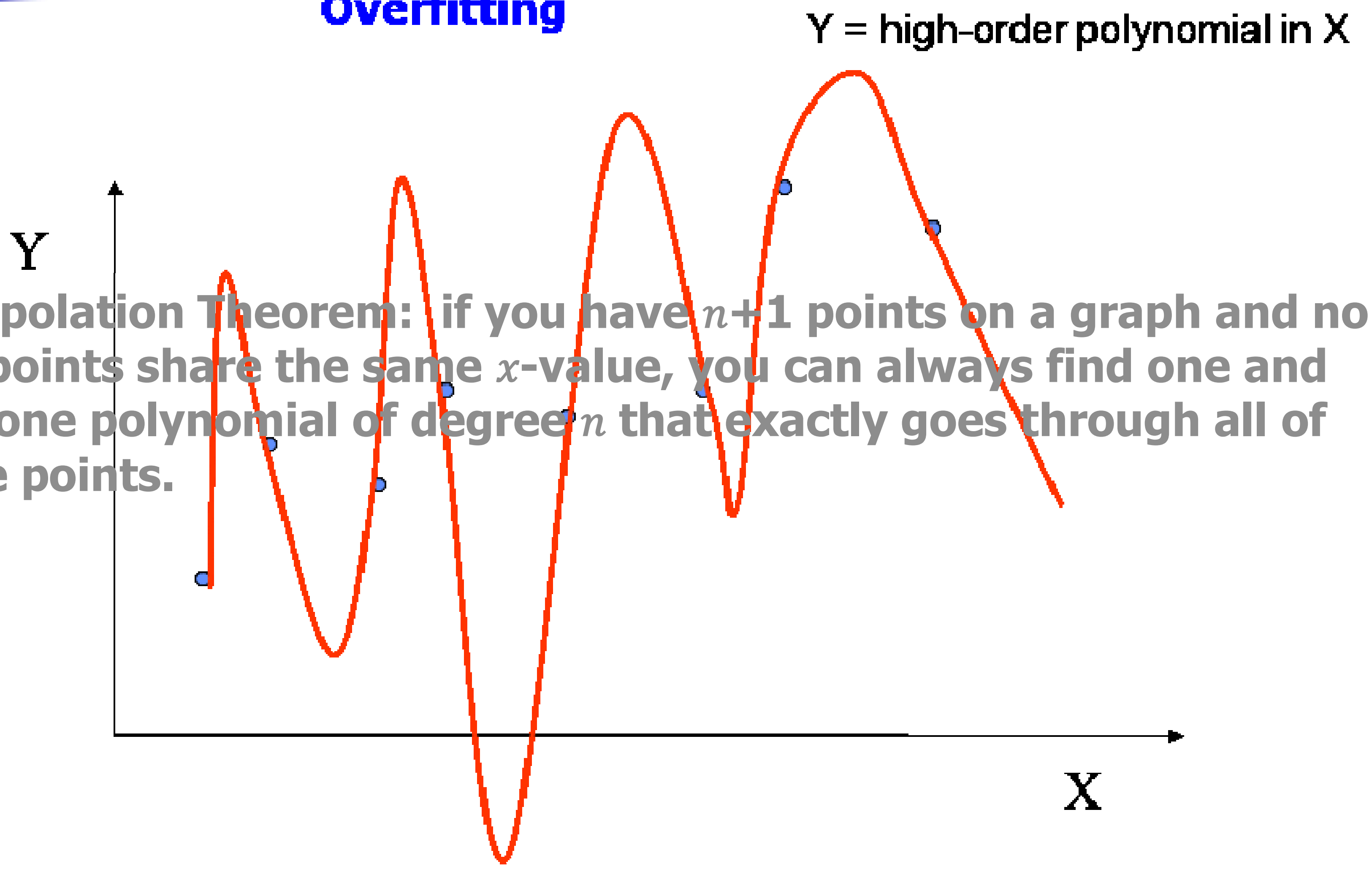
Generalization can be defined as a mathematical Robust *interpolation* or *regression* over a set of training points:

Overfitting and Underfitting



Generalization

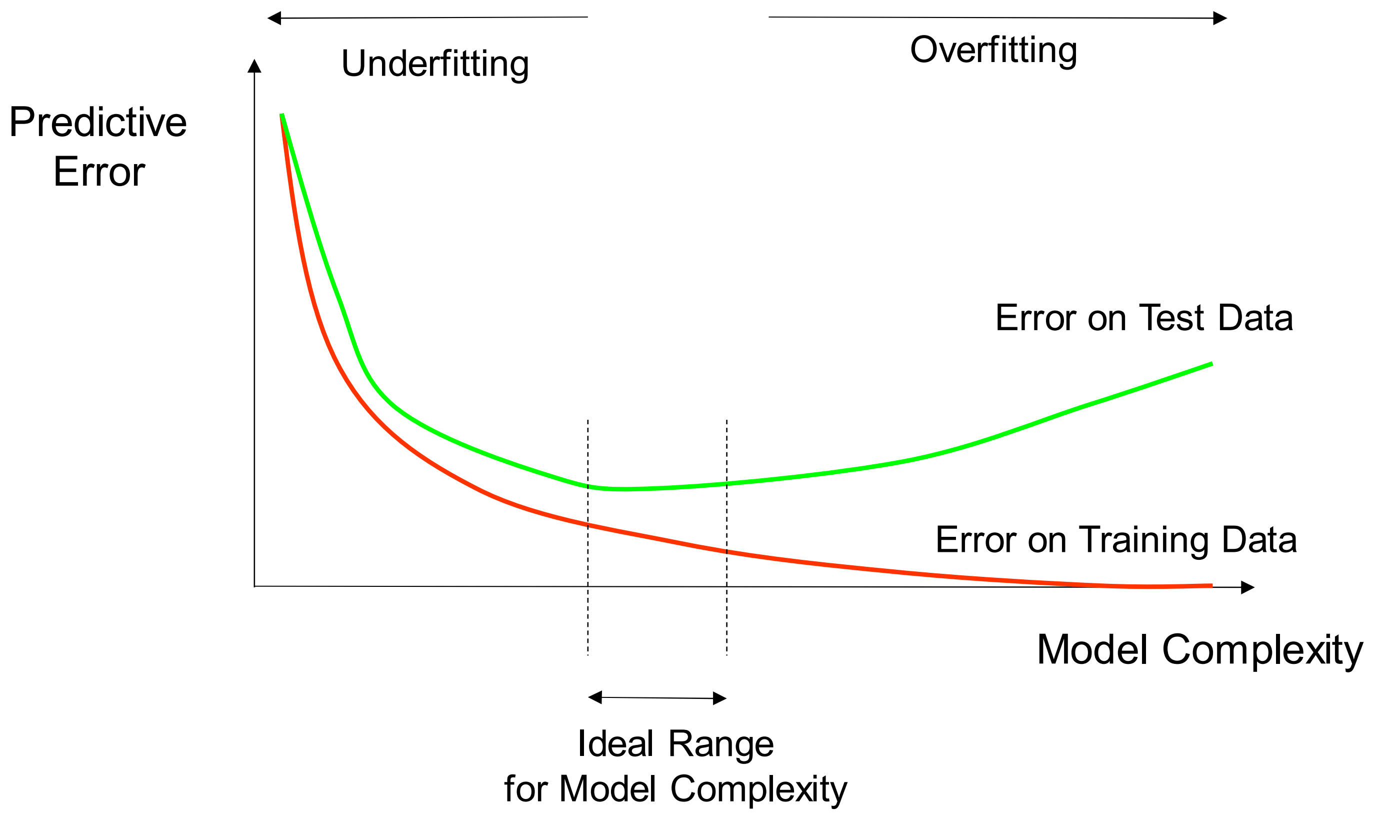
Overfitting



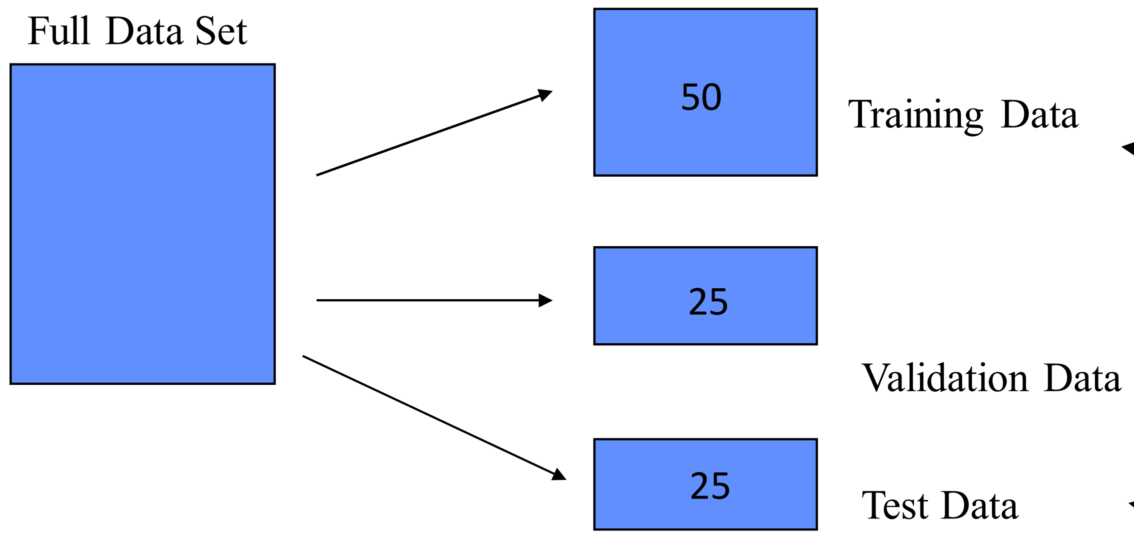
Interpolation Theorem: if you have n+1 points on a graph and no two points share the same x-value, you can always find one and only one polynomial of degree n that exactly goes through all of these points.

Generalization

How Overfitting affects Prediction



Training and Validation Data



Idea: train each model on the "training data"

and then test each model's accuracy on the validation data

And then test the best model on the Test data. This is the final accuracy

The k-fold Cross-Validation Method

- Why just choose one particular 90/10 “split” of the data?
  - In principle we could do this multiple times
- “k-fold Cross-Validation” (e.g., k=10)
  - We partition randomly our full dataset into k <u>disjoint subsets</u> (each roughly of size n/k, n = total number of training data points)
    - for i = 1:10 (here k = 10)
      - train on 90% of data in each subset (We use all combinations if the dataset is limited.)
      - Acc(i) = accuracy on other 10%
    - end
    - Cross-Validation-Accuracy = $1/k \; \Sigma_i \; Acc(i)$
  - choose the method with the highest cross-validation accuracy
  - common values for k are 5 and 10
  - Can also do “leave-one-out” where k = n

  Split a dataset into a training set and a testing set, using all but one observation as part of the training set. Repeat this process n times (where n is the total number of observations in the dataset), leaving out a different observation from the training set each time.

Generalization: ANN

N = # hidden nodes m = # training cases

W = # weights error tolerance ε

Network will generalize with 95% confidence if:

1. Error on training set < $\varepsilon / 2$

2. $m > O(\frac{W}{\varepsilon} \log_2 \frac{N}{\varepsilon}) \approx m > \frac{W}{\varepsilon}$

Universal Approximation Theorems for ANNs (late 80s-early '90s)

Single Layer NNs

- Cybenco, 1988, Approximation by superpositions of a sigmoidal function (L^1)

$$N(x; \alpha, \beta, w) = \sum_{i=1}^{n} w_i \sigma(\alpha_i x + \beta_i).$$

Theorem

For any function $f \in L^1([a,b])$ and for all $\varepsilon > 0$ there exist a choice of α, β, w such that

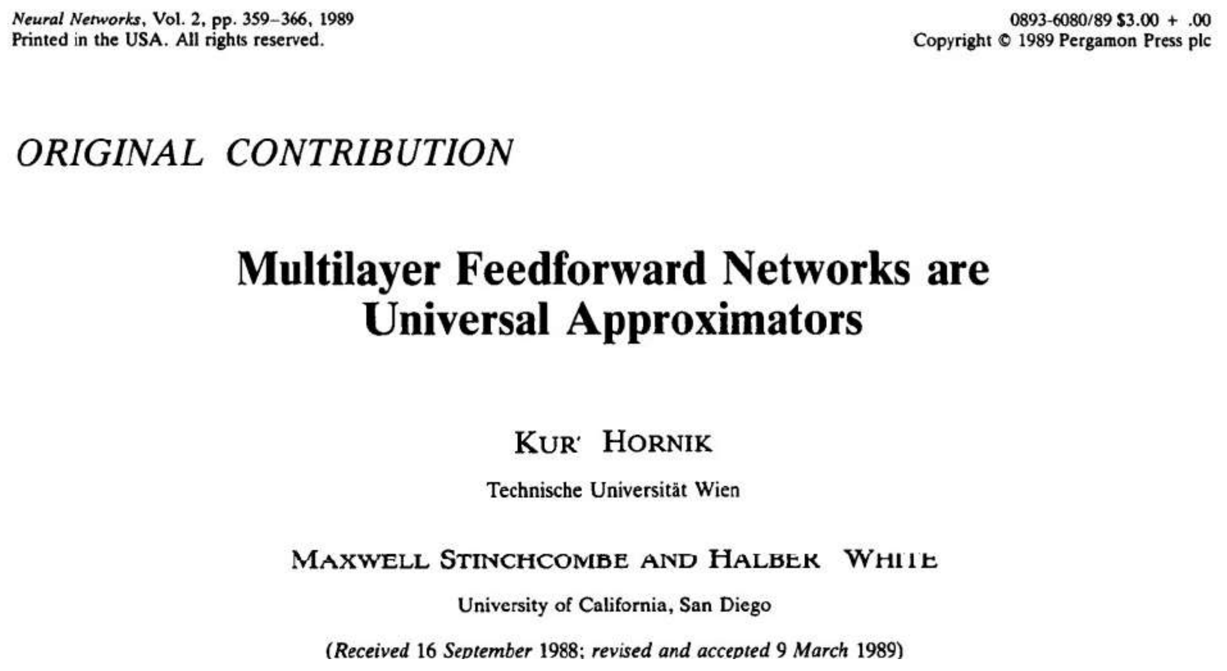
$$||N - f||_{L^1} < \varepsilon.$$

where $\| \cdot \|_{L^1}$ is the usual L^1 norm, i.e. $||\psi||_{L^1} = \int_a^b |\psi(x)|dx$.

Universal Approximation Theorems for ANNs (late 80s-early ‘90s)

Multilayer Layer NNs: Deep Learning

can approximate any Borel measurable function arbitrarily closely.



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

0893-6080/89 $3.00 + .00
Copyright © 1989 Pergamon Press plc

ORIGINAL CONTRIBUTION

Multilayer Feedforward Networks are Universal Approximators

KUR' HORNIK
Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBEK WHITE
University of California, San Diego

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

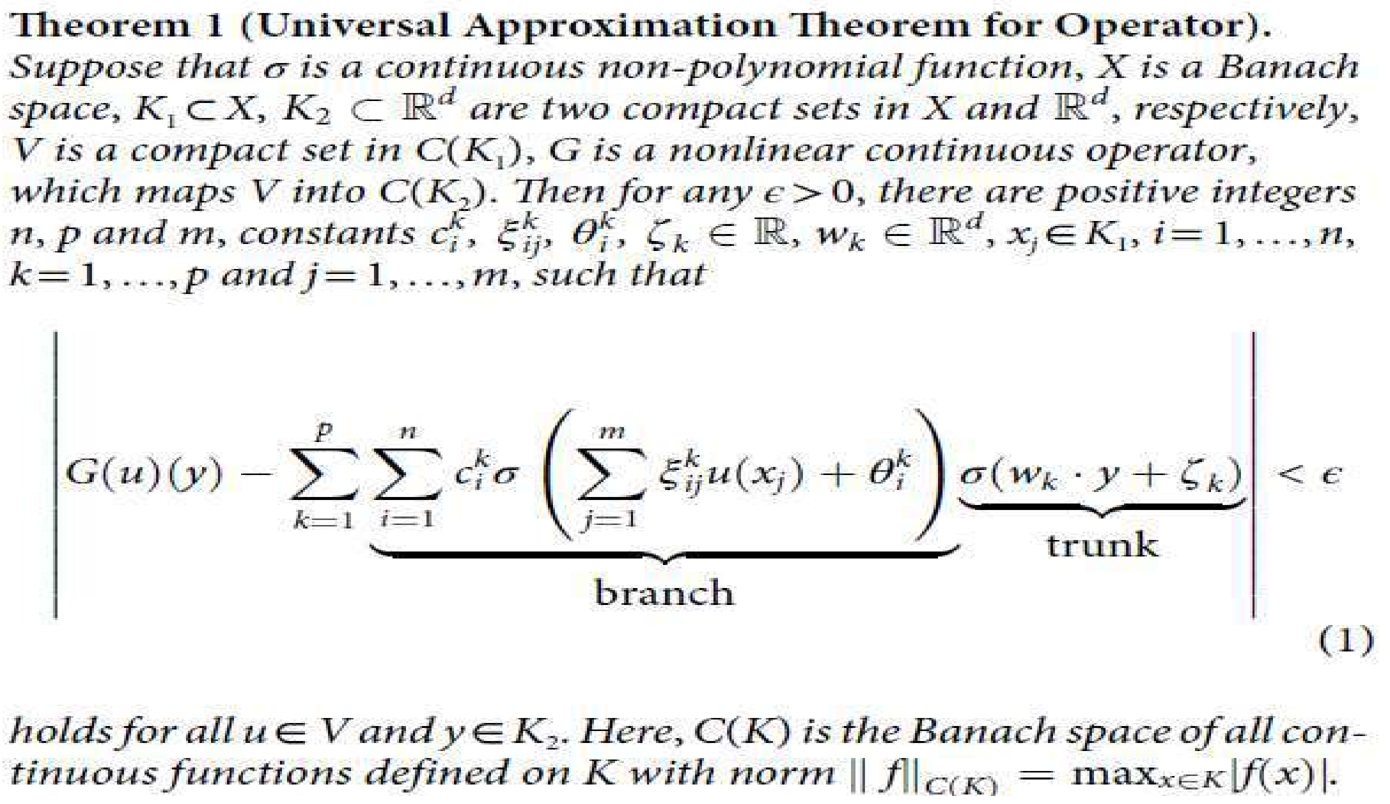
Borel Functions: Defined in terms of measurability with respect to the Borel sigma-algebra.

L^p **Functions:** Defined in terms of integrability with respect to a measure.

Universal Approximation Theorems for ANNs (‘90s)

Non-linear Operators

Chen, T. & Chen, H. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Trans. Neural Networks* **6**, 911–917 (1995).



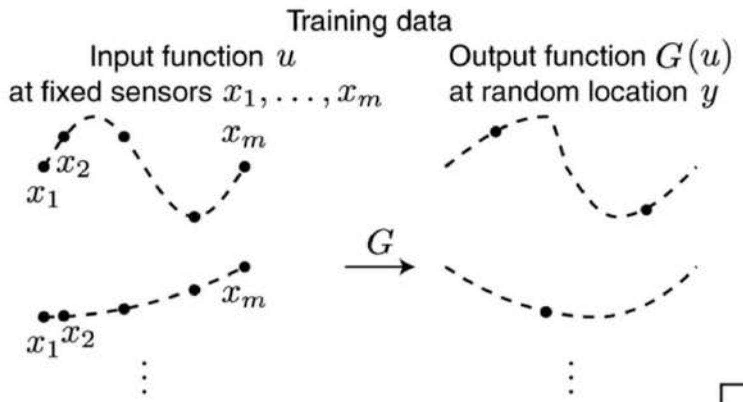
Theorem 1 (Universal Approximation Theorem for Operator). *Suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p and m, constants c_i^k, ξ_{ij}^k, θ_i^k, $\zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$ and $j = 1, \ldots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{\text{branch}} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{\text{trunk}} \right| < \epsilon \tag{1}$$

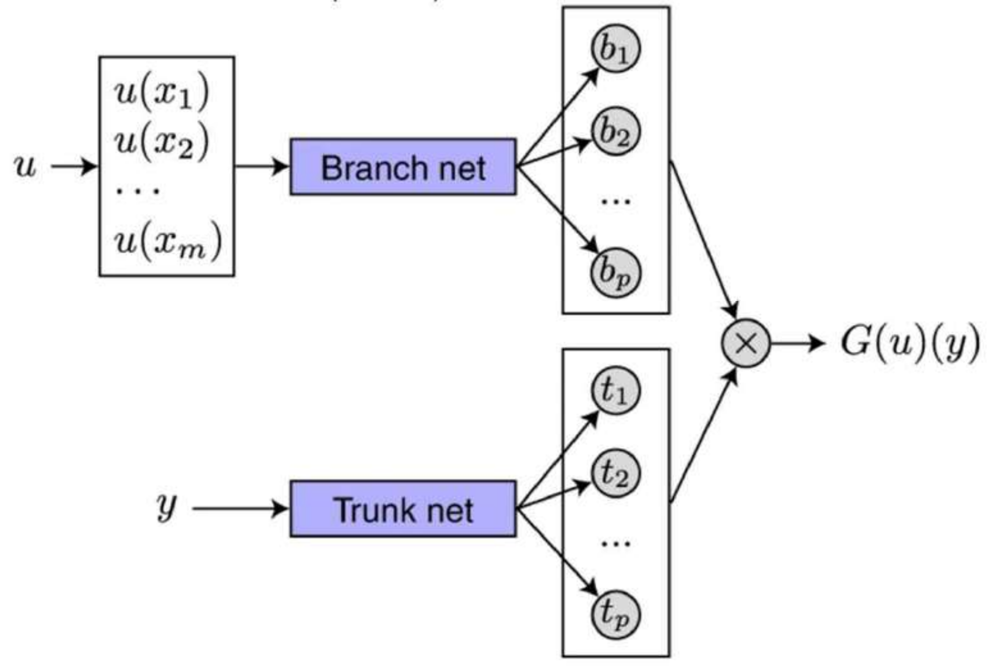
holds for all $u \in V$ and $y \in K_2$. Here, $C(K)$ is the Banach space of all continuous functions defined on K with norm $\| f \|_{C(K)} = \max_{x \in K} |f(x)|$.

Deep Learning 2020s

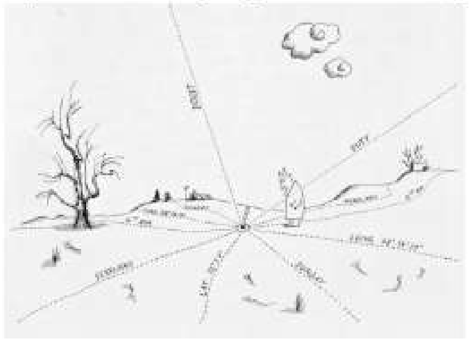
The DEEP-O-NET



Lu, L., Jin, P., Pang, G. *et al.* Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nat Mach Intell* **3**, 218–229 (2021).



Open challenging Tasks

- (A) Can ML beat traditional numerical analysis methods for the solution of stiff ODEs and PDEs?, **FORWARD PROBLEM**
- (B) Deal with the so-called "curse of dimensionality" when trying to efficiently learn ML models with good generalization properties, and 
- (C) Discover from data the appropriate macroscopic quantities/physics for the emergent dynamics,
This is Mr Smith from Big Data Mining.
He says he's found an insight.
INVERSE PROBLEM
- (D) Bridge Machine Learning with Physics-based Modelling, Discover Physical Laws from Data