PAUL SCHERRER INSTITUT

A General Introduction to Machine Learning (whenever possible with a twist towards accelerators)

Andreas Adelmann

2/3 May 2022

Acknowledgements

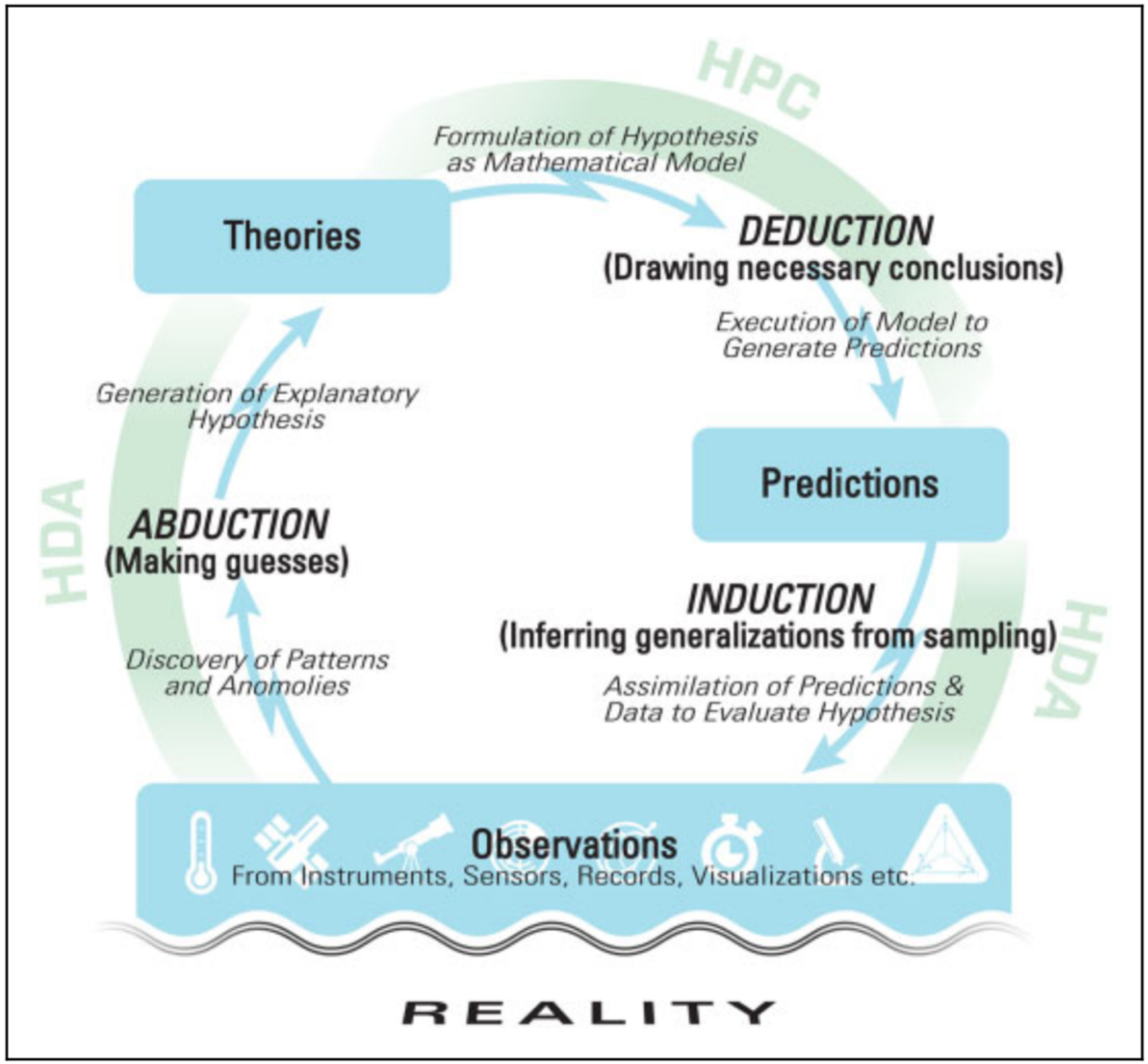
- Romana Boiger
- Kim Woo Hyun
- MIT 6.S191 lecture notes
- Titus Neupert et al. (arXiv:2102.04883)

PAUL SCHERRER INSTITUT

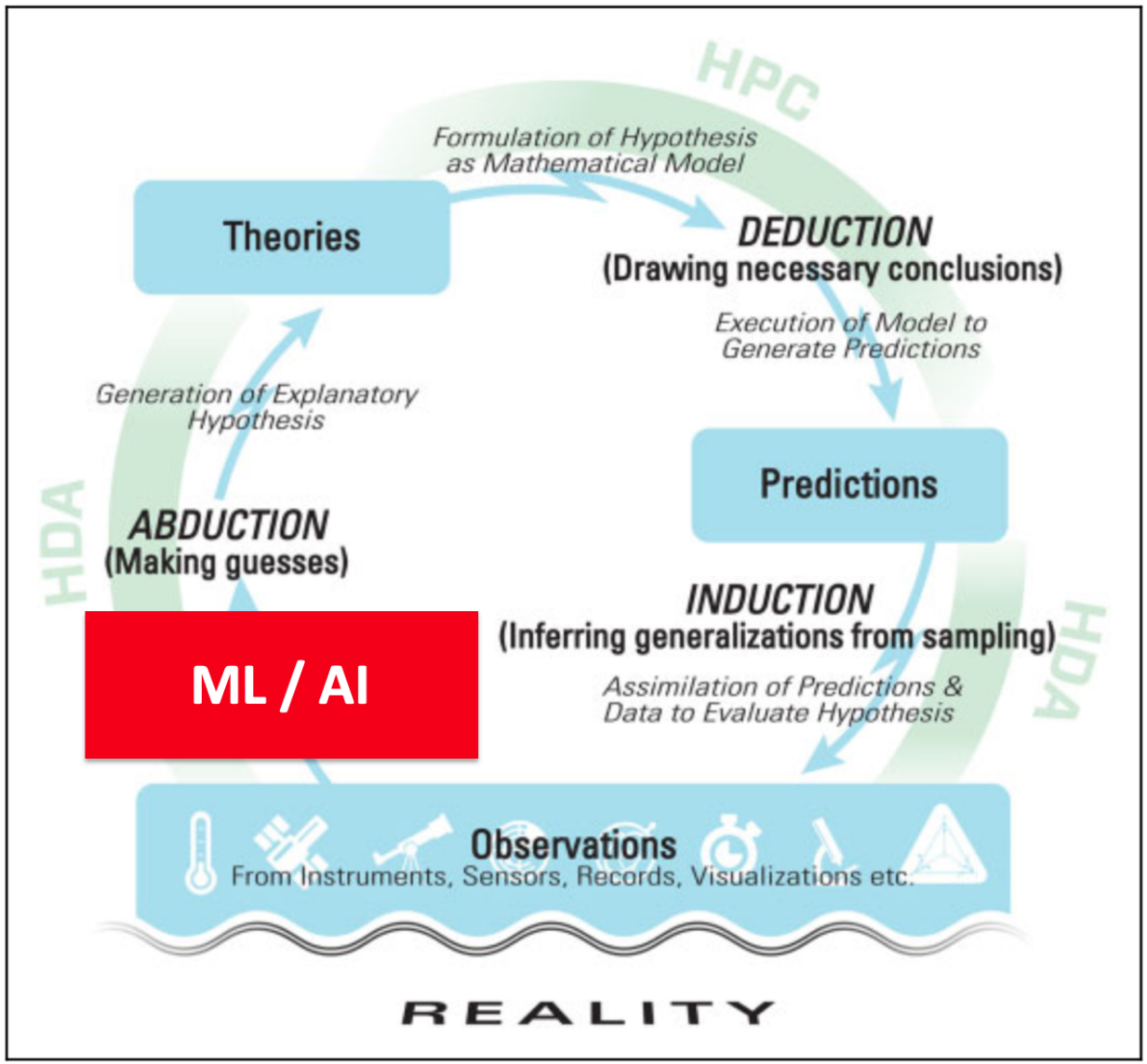
Outline

PAUL SCHERRER INSTITUT

Setup



Big data and extreme-scale computing:
Pathways to Convergence-Toward a shaping strategy for a future software and data ecosystem for scientific inquiry, M. Asch et al., DOI: 10.1177/1094342018778123



Big data and extreme-scale computing:
Pathways to Convergence-Toward a shaping strategy for a future software and data ecosystem for scientific inquiry, M. Asch et al., DOI: 10.1177/1094342018778123

Resources for Machine Learning I

Good software:

- almost any ML task one might be interested in can be done with relatively few lines of code simply by relying on external libraries.
- at the moment, most of the external libraries are written for the Python programming language.

Here are some useful Python libraries:

1. **TensorFlow.** Developed by Google, Tensorflow is one of the most popular and flexible library for machine learning with complex models, with full GPU support.
2. **PyTorch.** Developed by Facebook, Pytorch is the biggest rival library to Tensorflow, with pretty much the same functionalities.

PAUL SCHERRER INSTITUT

Resources for Machine Learning II

3. **Scikit-Learn.** Whereas TensorFlow and PyTorch are catered for deep learning practitioners, Scikit-Learn provides much of the traditional machine learning tools, including linear regression and PCA.
4. **Pandas.** Modern machine learning is largely reliant on big datasets. This library provides many helpful tools to handle these large datasets.

References

For further reading, I recommend the following books:

- **ML with neural networks**: Neural Networks and Deep Learning, M. Nielson (http://neuralnetworksanddeeplearning.com)
- **Deep Learning Theory**: Deep Learning, I. Goodfellow, Y. Bengio and A. Courville (http://www.deeplearningbook.org)

Supervised Learning without Neural Networks I

Supervised learning is the term for a machine learning task, where we are given a dataset consisting of input-output pairs

$$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$$

and our task is to "learn" a function which maps input to output $f : \boldsymbol{x} \mapsto y \in \mathbb{R}$ w.l.g.
The output data that we have is called the ground truth and sometimes also referred to as "labels" of the input.
Within the scope of supervised learning, there are two main types of tasks:

Supervised Learning without Neural Networks II

1. *Classification:* the output y is a discrete variable corresponding to a classification category. An example of such a task would be to distinguish stars with a planetary system (exoplanets) from those without given time series of images of such objects.
2. *Regression:* the output y is a continuous number or vector. For example predicting emittances in a FODO channel based previous high fidelity simulation and/or measurements.

PAUL SCHERRER INSTITUT

Linear regression I

Fiting a linear model to a dataset. Consider a dataset consisting of input-output pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$, where the inputs are n-component vectors $\boldsymbol{x}^T = (x_1, x_2, \ldots, x_n)$ and the output y is a real-valued number. The linear model then takes the form

$$f(\boldsymbol{x}|\boldsymbol{\beta}) = \beta_0 + \sum_{j=1}^{n} \beta_j x_j = \tilde{\boldsymbol{x}}^T \boldsymbol{\beta}$$

where $\tilde{\boldsymbol{x}}^T = (1, x_1, x_2, \ldots, x_n)$ and $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_n)^T$ are $(n+1)$ dimensional row vectors.
The aim then is to find parameters $\hat{\boldsymbol{\beta}}$ such that $f(\boldsymbol{x}|\hat{\boldsymbol{\beta}})$ is a good estimator for the output value y.

Linear regression II

Given a real-valued loss function $L(\boldsymbol{\beta})$ (cost function), the good set of parameters $\hat{\boldsymbol{\beta}}$ is then the minimizer of this loss function

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, L(\boldsymbol{\beta}). \tag{1}$$

There are many choices for this loss function, as we will see later.

Surrogate Model & Sensitivity Analysis I

AA, SIAM/ASA J. UQ 2019 Vol. 7, No. 2, pp. 383-416

All square integrable, second-order random processes with finite variance output, $u(\xi) \in L_2(\Omega, \mathcal{F}, \mathcal{P})$, can be written as [**?**]

$$u = \sum_{k=0}^{\infty} u_k \psi_k(\xi)$$

- u: Random Variable (RV) represents 1D PCE
- u_k PC coefficients (deterministic)
- ψ_k: 1D Hermite polynomial of order k
- ξ: Gaussian RV

Surrogate Model & Sensitivity Analysis II

AA, SIAM/ASA J. UQ 2019 Vol. 7, No. 2, pp. 383-416

Expansion in terms of functions of random variables multiplied with deterministic PC coefficients

- Set of deterministic PC coefficients fully describes RV
- Separates randomness from deterministic dimensions

Algorithm: generate for each design or controllable, a PC surrogate model

1. generate $N = (p+1)^d$ quadrature point-weight pairs $(\boldsymbol{\xi}^n, w_n)$

Surrogate Model & Sensitivity Analysis III

AA, SIAM/ASA J. UQ 2019 Vol. 7, No. 2, pp. 383-416

2. for each of quadrature point $\boldsymbol{\xi}^n$ compute corresponding model input $\boldsymbol{\lambda}^n$ by

$$\boldsymbol{\lambda}^n = \lambda_j^n = \sum_{k=0}^{K-1} \lambda_{jk} \Psi_k(\boldsymbol{\xi}^n) \qquad j = 1, \ldots, d. \tag{2}$$

3. create the training points with high fidelity simulations (OPAL)

$$u_i^n = \mathcal{M}(\boldsymbol{\lambda}^n, y_i) \qquad i = 1, \ldots, l. \tag{3}$$

Surrogate Model & Sensitivity Analysis IV

AA, SIAM/ASA J. UQ 2019 Vol. 7, No. 2, pp. 383-416

4. calculate the expectation via orthogonal projection and using quadrature

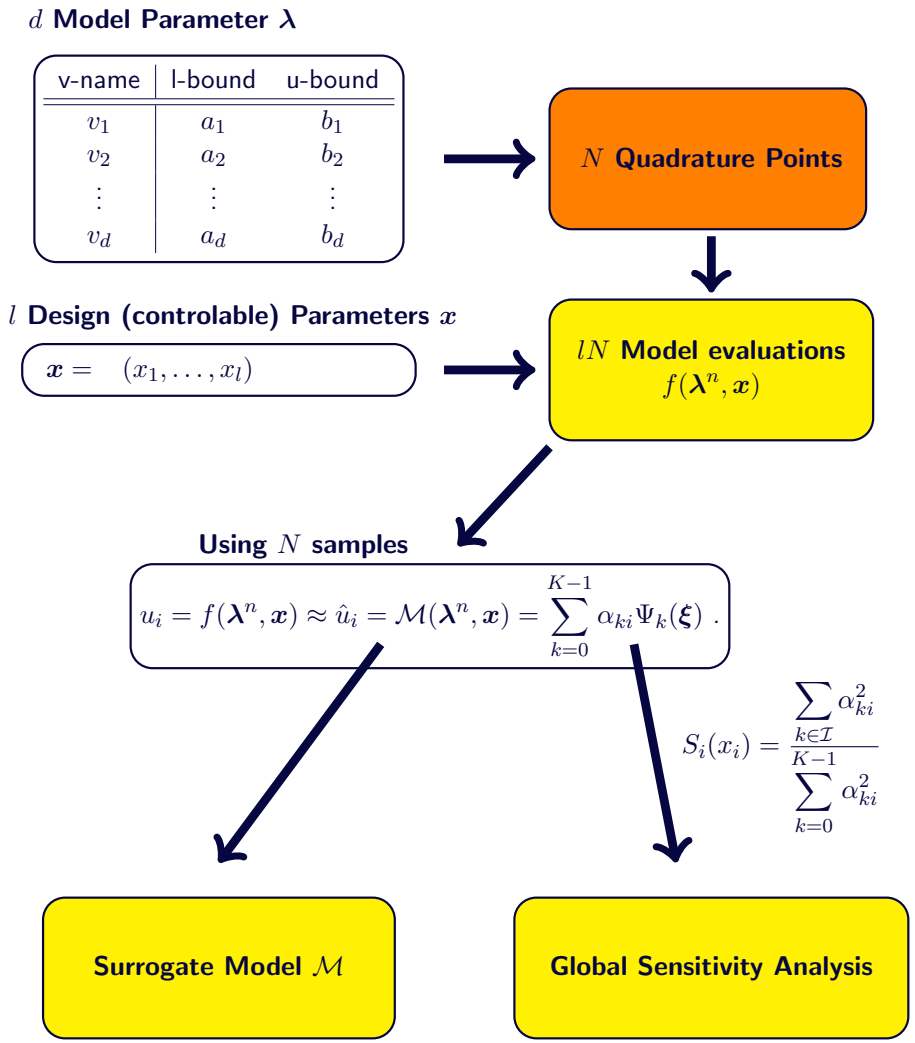
$$\alpha_{ki} = \frac{\langle u\Psi_k \rangle}{\langle \Psi_k^2 \rangle} = \frac{1}{\langle \Psi_k^2 \rangle} \sum_{n=1}^{N} u_i^n \Psi_k(\boldsymbol{\xi}^n) w_n, \quad k = 0, \ldots, K-1. \tag{4}$$

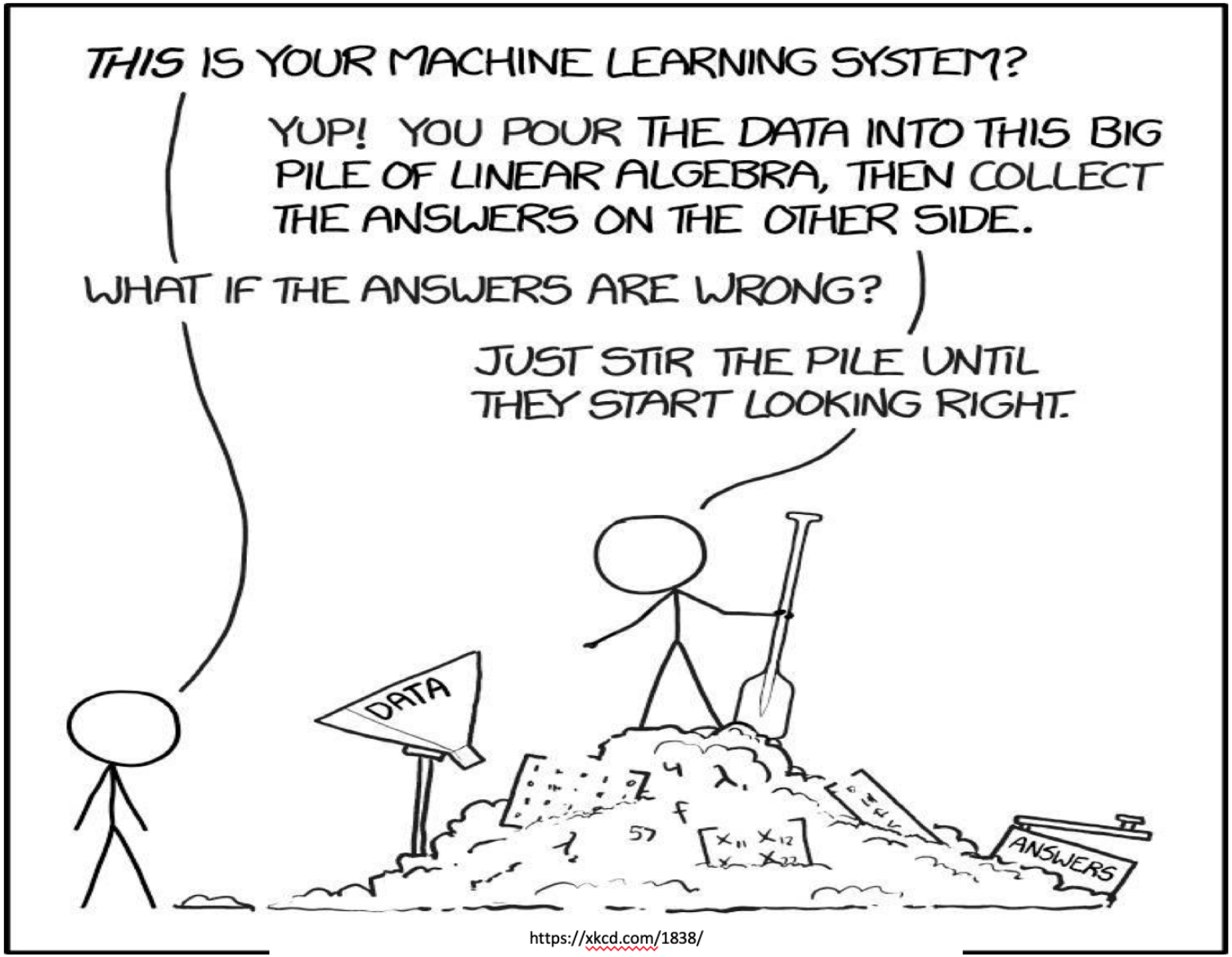
5. Given the computed α_{ki} values for each i and k, one assembles the PCE

$$\hat{u}_i = \sum_{k=0}^{K-1} \alpha_{ki} \Psi_k(\boldsymbol{\xi}), \quad k = 0, \ldots, K-1. \tag{5}$$

Surrogate Model & Sensitivity Analysis V

AA, SIAM/ASA J. UQ 2019 Vol. 7, No. 2, pp. 383-416





https://xkcd.com/1838/

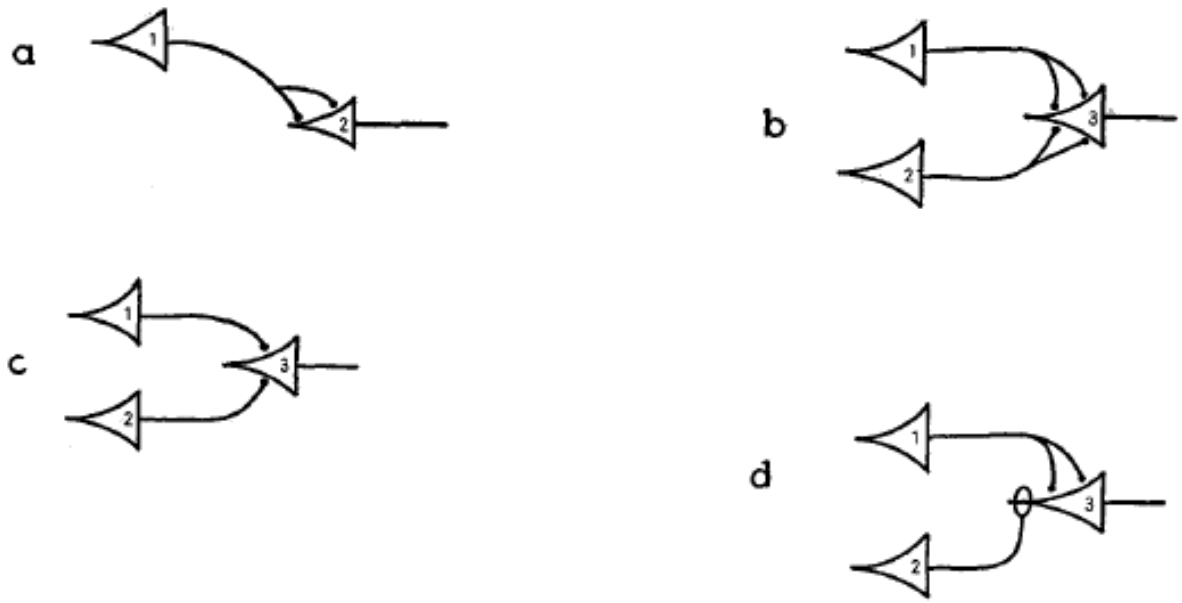
PAUL SCHERRER INSTITUT

Different Neural Networks Topologies

First Generation

Artificial Neural Network (ANN)

At 1943 **McCulloch, Warren S.**, and **Walter Pitts** suggested

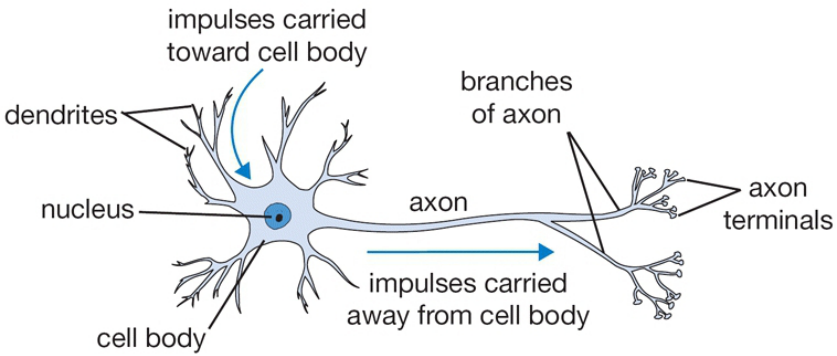


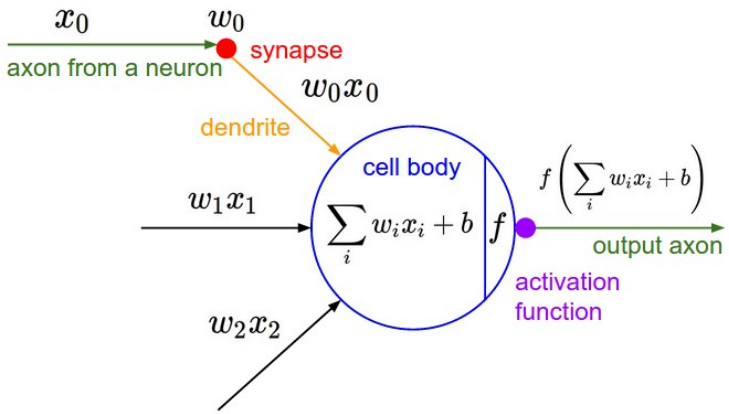
- Mimic the human neural structure by connecting switches

First Generation

Perceptron

In 1958 **Frank Rosenblatt** suggested Linear Classifier.



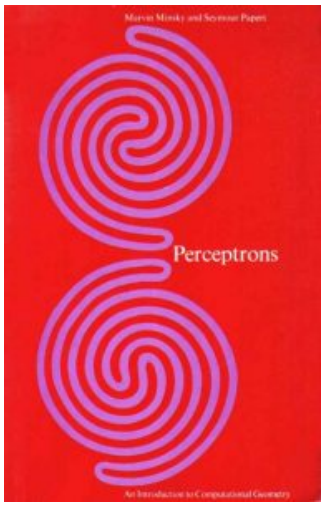


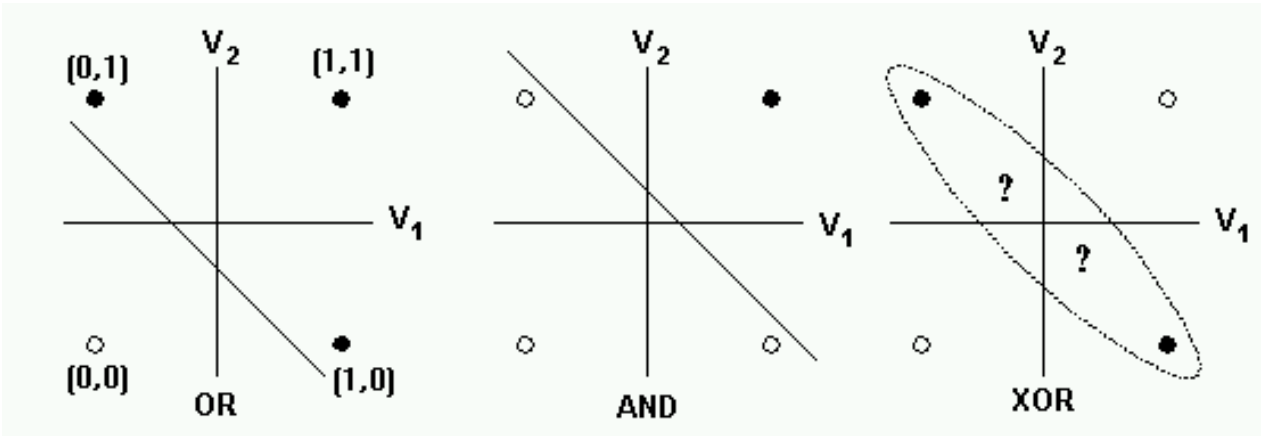
- Expected computer can do things human can do
- Basic structure is not changed until now
- Using sigmoid with **Activation function** (Make output $\in$ [0,1])

First Generation

Problem

In 1969 **Marvin Minsky, Seymour Papert** proved limitations of perceptron.

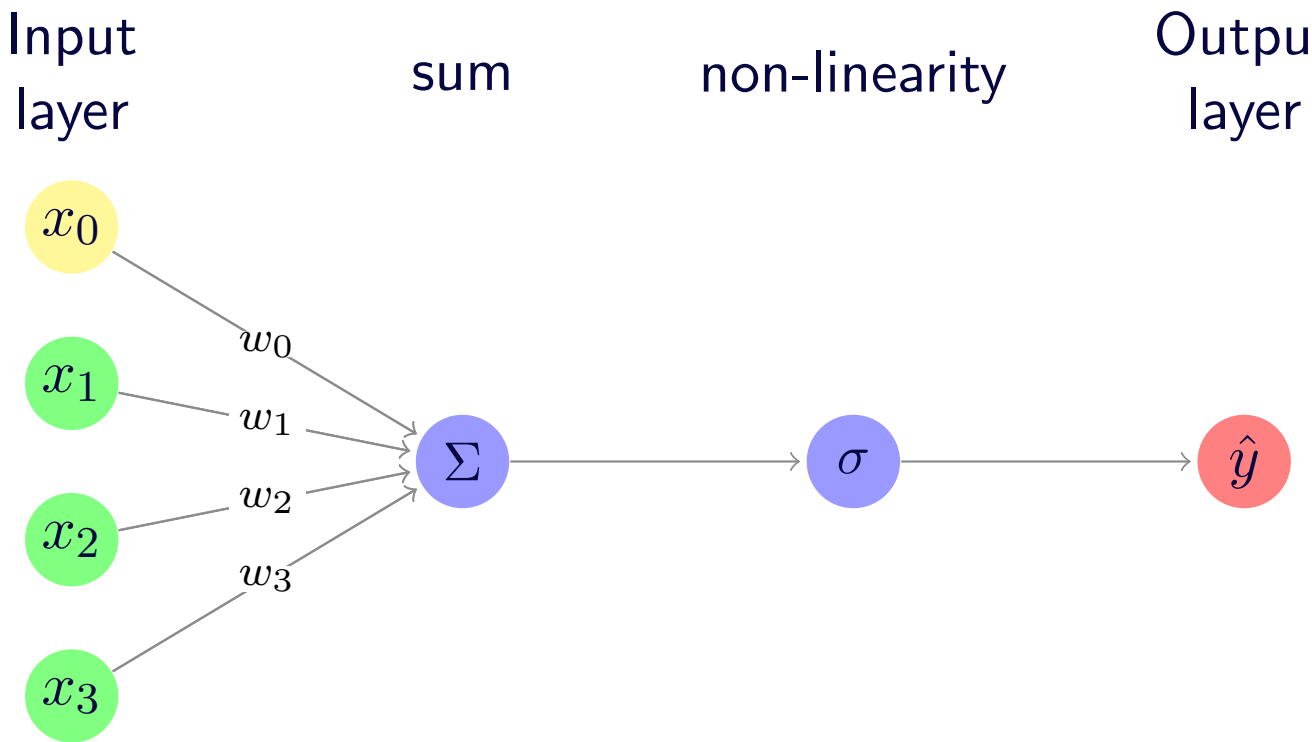




Remark

It can't solve XOR problem even.

Perceptron a closer Look I



We now set $x_0 = 1$ and identify w_0 with the bias and $m = 3$:

$$\hat{y} = \sigma\left(w_0 + \sum_{i=1}^{m} x_i w_i\right) = \sigma\left(w_0 + \mathbf{X}^T\mathbf{W}\right)$$

Perceptron a closer Look II

Remarks

- *a perceptron (one fully connected unit) without activation function will just be a linear regressor*
- *if you put a sigmoid activation you get a classifier*
- *actually, with neural networks, classification is a special case of regression where we "regress" the probability of belonging to a given class.*

PAUL SCHERRER INSTITUT

Perceptron a closer Look III

- universal approximation theorems are results that establish the density of an algorithmically generated class of functions within a given function space of interest.
- typically, these results concern the approximation capabilities of the feedforward architecture on the space of continuous functions between two Euclidean spaces, and the approximation is with respect to the compact convergence topology
- there are results between non-Euclidean spaces and other commonly used architectures such as the convolutional neural network architecture, radial basis-functions

PAUL SCHERRER INSTITUT

Perceptron a closer Look IV

Theorem (Universal approximation theorem (simplified))

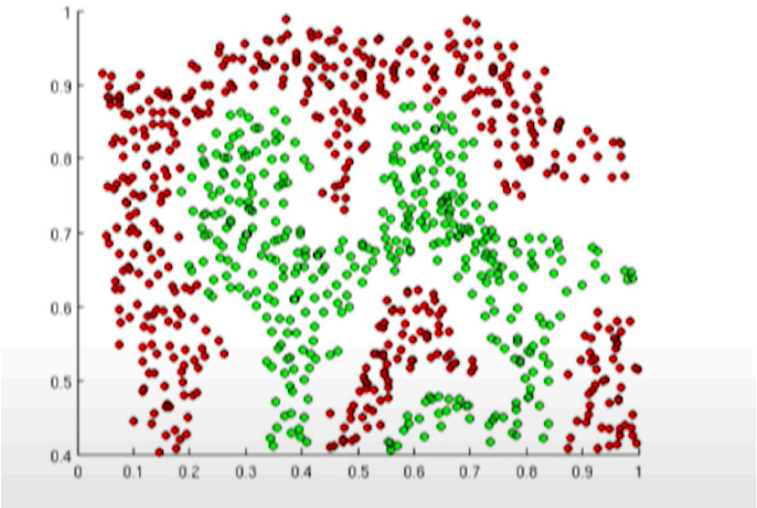
Let $C(X, Y)$ denote the set of continuous functions from X to Y. Let $\sigma \in C(\mathbb{R}, \mathbb{R})$, one of our activation functions. Then for every $n \in \mathbb{N}$, $m \in \mathbb{N}$, compact $K \subseteq \mathbb{R}^n$, $f \in C(K, \mathbb{R}^m), \varepsilon > 0$, there exist $k \in \mathbb{N}, A \in \mathbb{R}^{k \times n}, b \in \mathbb{R}^k$ such that

$$\sup_{x \in K} \|f(x) - g(x)\| < \varepsilon$$

where $g(x) = (\sigma \circ (A \cdot x + b))$.

The Importance & Purpose of Activation Functions

Suppose we want to distinguish (classify) red from green points.

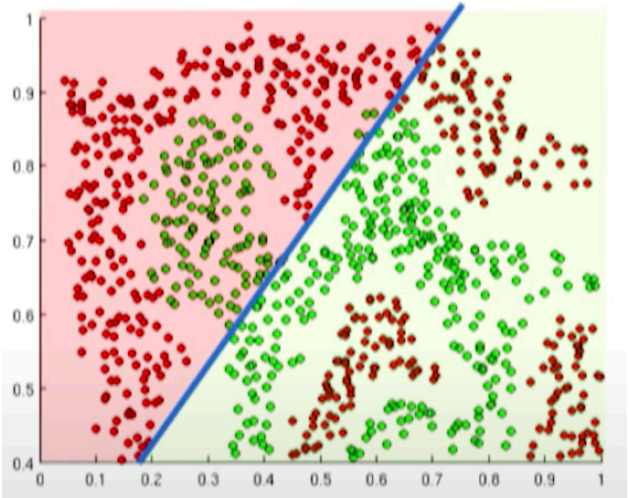


Remark

- *a linear function will not be able to do the job*
- *need non-linearities*

The Importance & Purpose of Activation Functions

Suppose we want to distinguish (classify) red from green points.

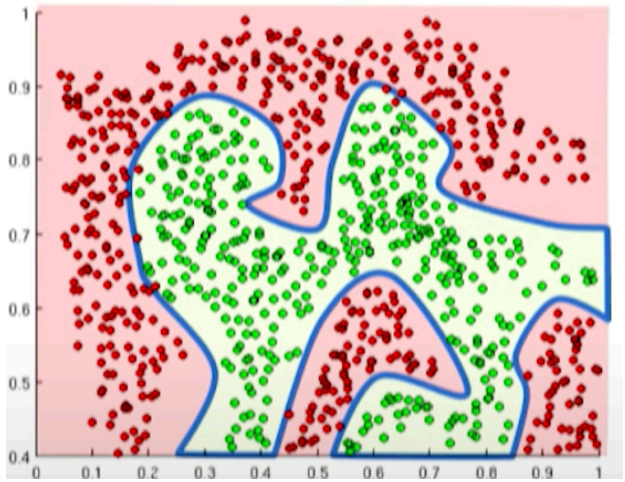


Remark

- *a linear function will not be able to do the job*
- *need non-linearities*

The Importance & Purpose of Activation Functions

Suppose we want to distinguish (classify) red from green points.



Remark

- *a linear function will not be able to do the job*
- *need non-linearities*

Activation Functions I

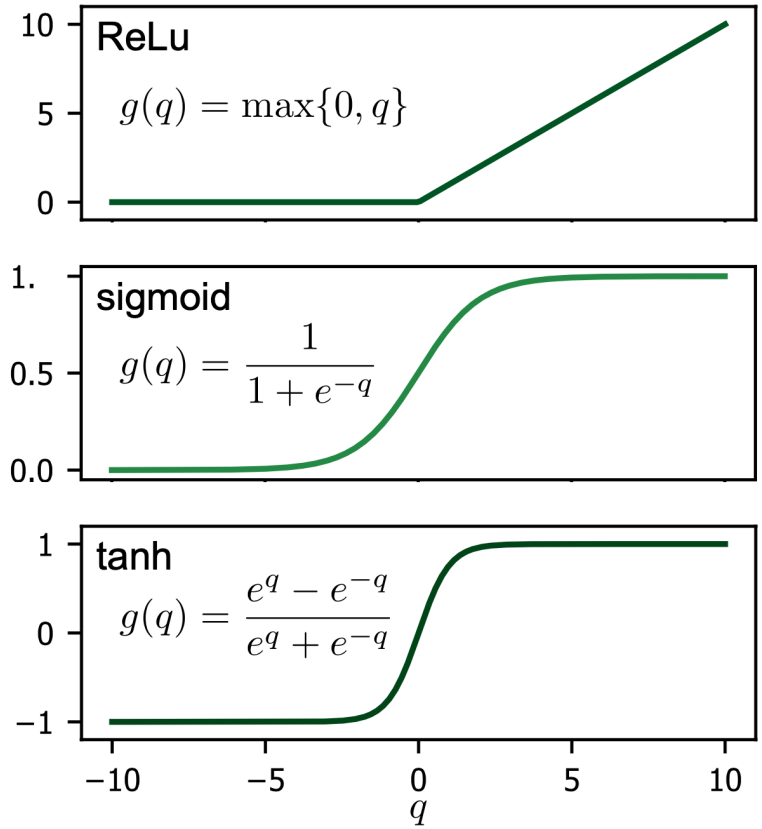
Table 3: Non-linearities tested.

Name	Formula	Year
none	$y = x$	-
sigmoid	$y = \frac{1}{1+e^{-x}}$	1986
tanh	$y = \frac{e^{2x}-1}{e^{2x}+1}$	1986
ReLU	$y = \max(x, 0)$	2010
(centered) SoftPlus	$y = \ln(e^x + 1) - \ln 2$	2011
LReLU	$y = \max(x, \alpha x)$, $\alpha \approx 0.01$	2011
maxout	$y = \max(W_1 x + b_1, W_2 x + b_2)$	2013
APL	$y = \max(x,0) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s)$	2014
VLReLU	$y = \max(x, \alpha x)$, $\alpha \in 0.1, 0.5$	2014
RReLU	$y = \max(x, \alpha x)$, $\alpha = \text{random}(0.1, 0.5)$	2015
PReLU	$y = \max(x, \alpha x)$, α is learnable	2015
ELU	$y = x$, if $x \geq 0$, else $\alpha(e^x - 1)$	2015

Notice at gap between tanh and ReLU.

Activation Functions II

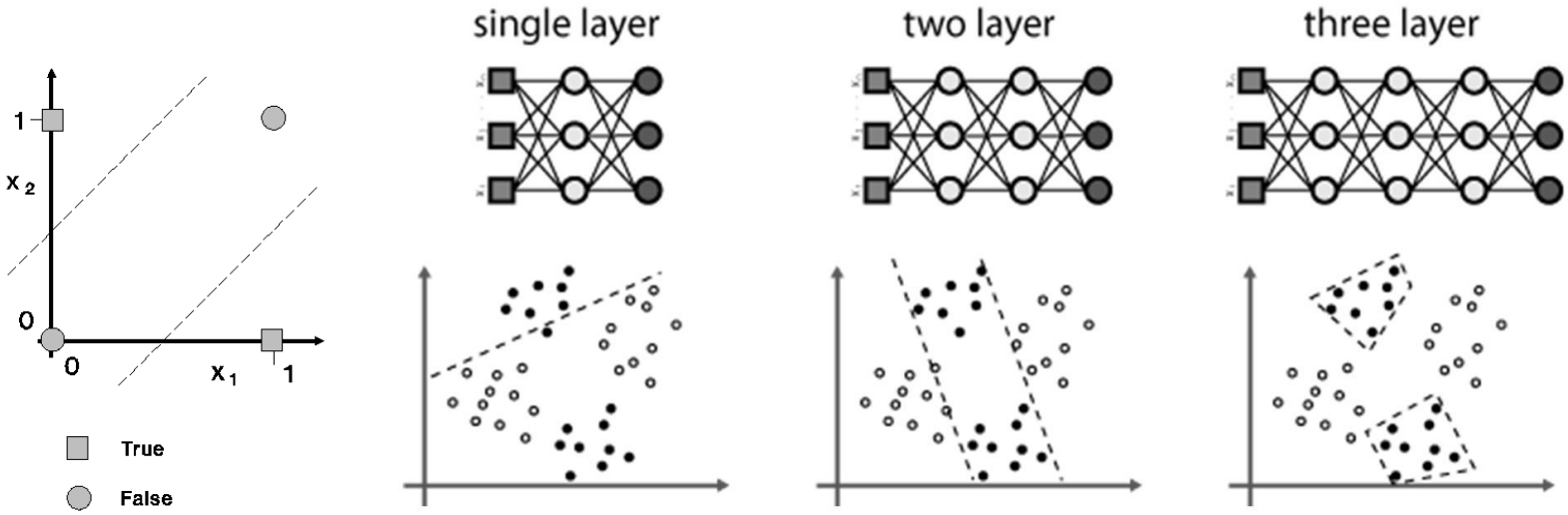
- ReLU: ReLU stands for rectified linear unit and is zero for all numbers smaller than zero, while a linear function for all positive numbers.
- Sigmoid: The sigmoid function, usually taken as the logistic function, is a smoothed version of the step function.
- Hyperbolic tangent: The hyperbolic tangent function has a similar behaviour as sigmoid but has both positive and negative values.



Second Generation

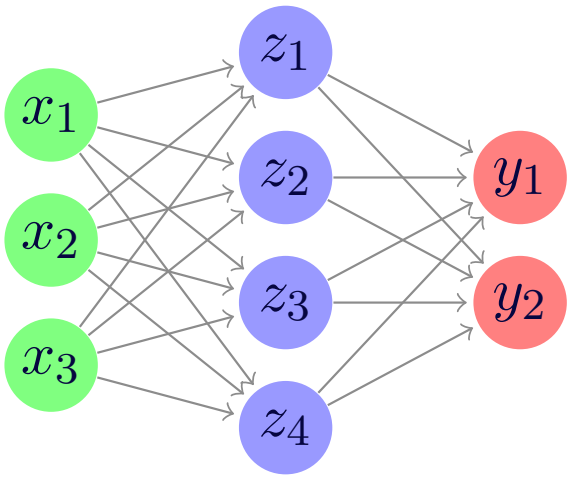
Multi-Layer Perception (MLP)

Make neurons deeper by make **hidden layers** of perception



- Solve more complex problems with multiple linear classifier.

Neural Networks with Perceptron as Building Block I



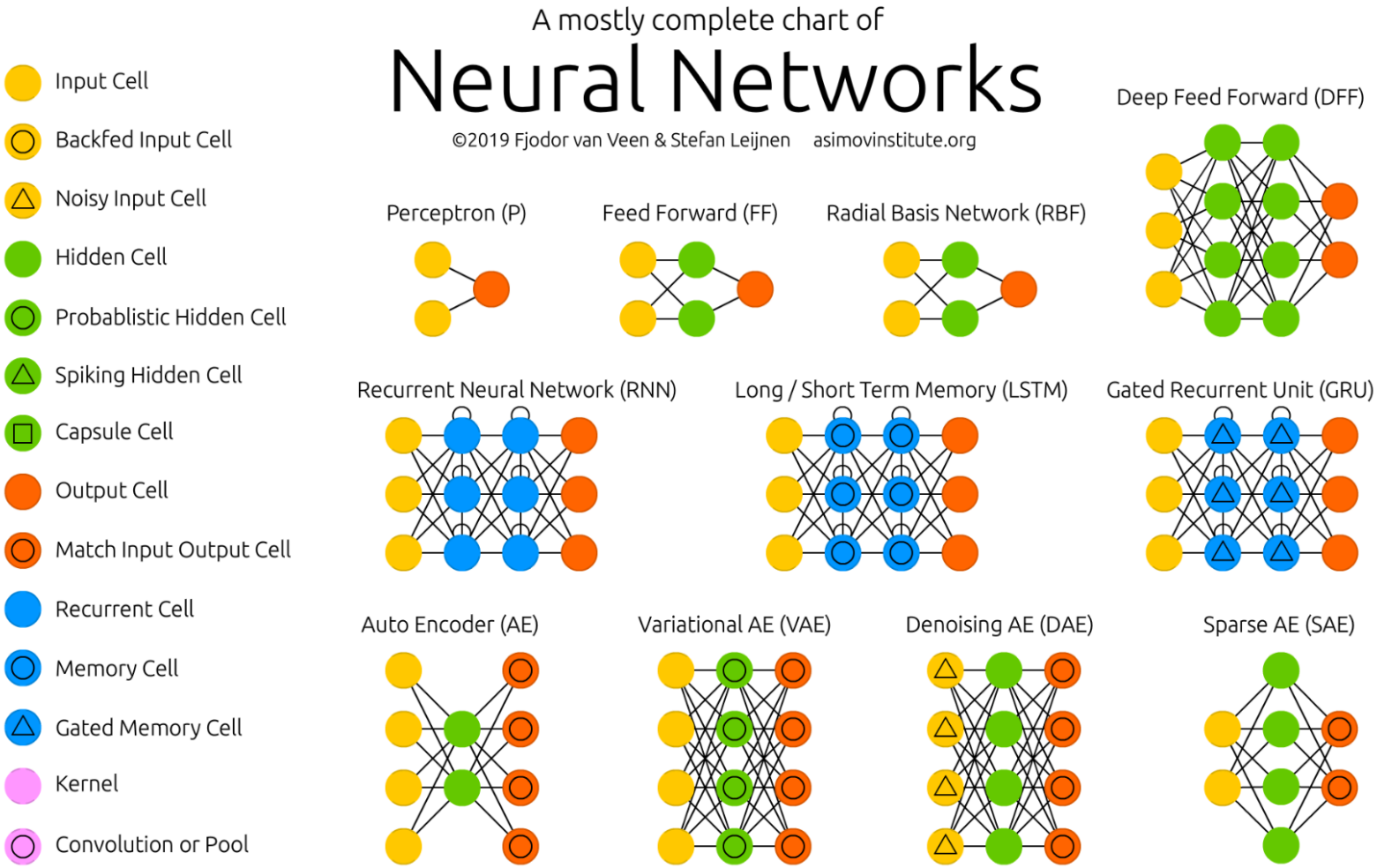
$$z_j = w_{0,j}^{(1)} + \sum_{i=1}^{m} x_i w_{i,j}^{(1)}$$

$$\hat{y}_j = \sigma\left(w_{0,j}^{(2)} + \sum_{i=1}^{d_1} g(z_j w_{i,j}^{(2)} \right)$$

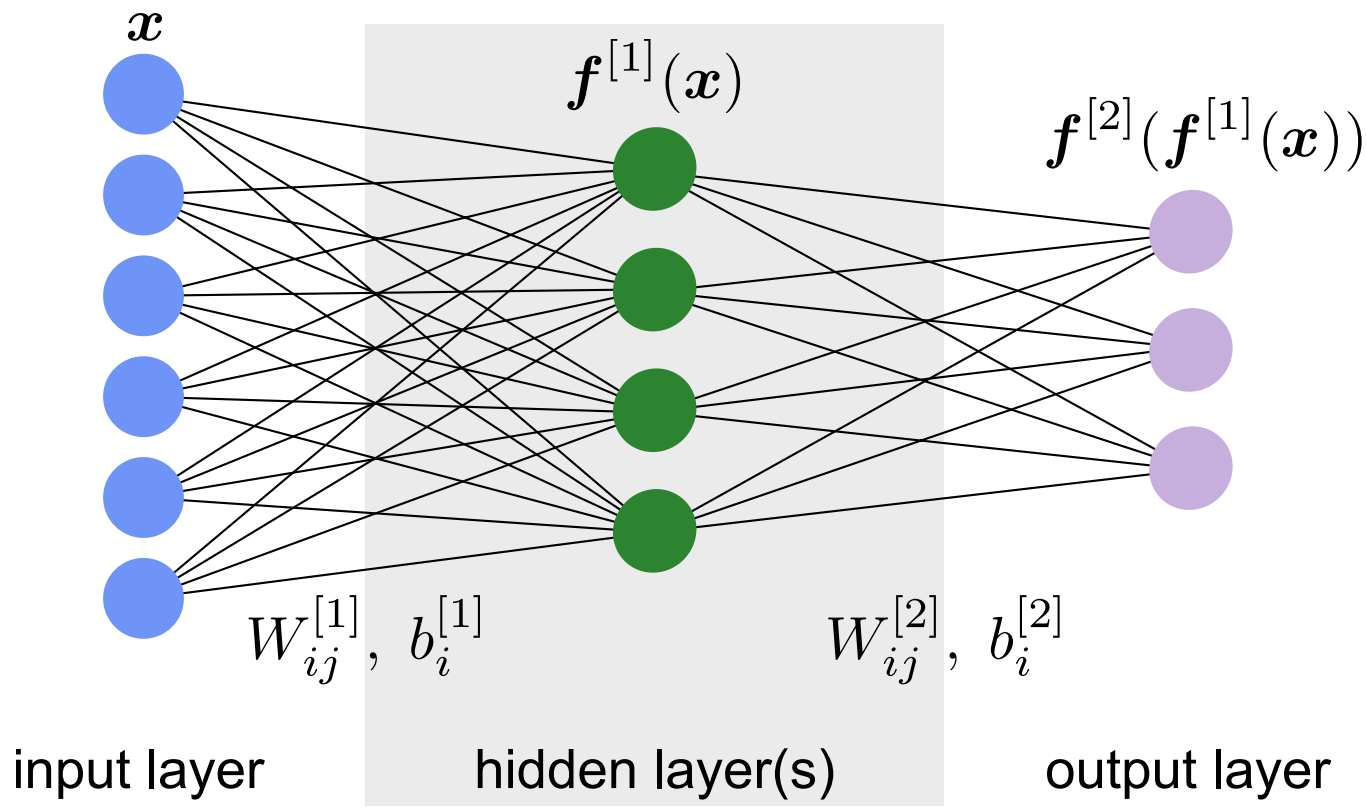
Remarks

- *single hidden-layer dense neural network*
- *input/output are called visible layers*

Neural Networks with Perceptron as Building Block II



Neural Networks with Perceptron as Building Block III



Neural Networks with Perceptron as Building Block IV

Assuming we can feed the input to the network as a vector, we denote the input data with $\boldsymbol{x}$. The network then transforms this input into the output $\hat{\boldsymbol{y}}(\boldsymbol{x})$, which in general is also a vector.

$$\hat{\boldsymbol{y}}(\boldsymbol{x}) = \boldsymbol{f}^{[2]}\left(W^{[2]}\boldsymbol{f}^{[1]}\left(W^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}\right) + \boldsymbol{b}^{[2]}\right).$$

Here, $W^{[n]}$ and $\boldsymbol{b}^{[n]}$ are the weight matrix and bias vectors of the n-th layer.

- $W^{[1]}$ is the $k \times l$ weight matrix of the hidden layer with k and l the number of neurons in the input and hidden layer
- $W_{ij}^{[1]}$ is the j-the entry of the weight vector of the i-th neuron in the hidden layer
- $b_i^{[1]}$ is the bias of this neuron

Neural Networks with Perceptron as Building Block V

- The $W_{ij}^{[2]}$ and $\boldsymbol{b}_i^{[2]}$ are the respective quantities for the output layer.

This network again is a fully connected or dense, because each neuron in a given layer takes as input the output from all the neurons in the previous layer, in other words all weights are allowed to be non-zero.

Training

PAUL SCHERRER INSTITUT

Training I

- given a labeled data set $\mathcal{D} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$
- we denote the variartional parameter $(\mathbf{W}, \mathbf{b}) =: \theta$
- adjusting all the weights and biases to achieve the task constitutes the training of the network. In other words, the training is the process that makes the network an approximation to the mathematical function

$$\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x}) \sim \hat{\boldsymbol{y}}$$

  that we want it to represent.
- each neuron has its own bias and weights, a potentially huge number of variatonial parameters, and we will need to adjust all of them.

Training II

The loss of our network measures the cost incurred from incorrect predictions and is defined as:

$$\mathcal{L}\left(\hat{\mathbf{y}}(\mathbf{x};\theta),\mathbf{y})\right)$$

In case we have a dataset with N datapoint $(\mathbf{x},\mathbf{y})$ the empirical loss (cost, objective) function can be defined:

$$\mathcal{L}_e = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\left(\hat{\mathbf{y}}(\mathbf{x})^{(i)};\theta),\mathbf{y}^{(i)}\right)$$

Training III

Remark

The choice of loss function may strongly impact the efficiency of the training and is based on heuristics (as was the case with the choice of activation functions).

A few common used loss functions are:

- mean square error (L2) loss:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} ||\hat{\boldsymbol{y}}(\boldsymbol{x}_i) - \boldsymbol{y}_i||_2^2. \tag{6}$$

Here, $||\boldsymbol{a}||_2 = \sqrt{\sum_i a_i^2}$ is the $L2$ norm and thus, this loss function is also referred to as $L2$ loss mean square error. An

Training IV

advantage of the L2 loss is that it is a smooth function of the variational parameters.

- mean absolute error:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} ||\hat{\boldsymbol{y}}(\boldsymbol{x}_i) - \boldsymbol{y}_i||_1, \tag{7}$$

where $||\boldsymbol{a}||_1 = \sum_i |a_i|$ denotes the $L1$ norm. Note that the $L2$ norm, given the squares, puts more weight on outliers than the $L1$ loss.

The two loss functions introduced so far are the most common loss functions for networks providing a continuous output.

PAUL SCHERRER INSTITUT

Loss Optimisation

Loss Optimisation

The goal is to find θ such that the achieved loss is minimized:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\hat{y}(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$$

- L is typically a high-dimensional function and may have many nearly degenerate minima
- finding the absolute minimum exactly is typically intractable analytically and prohibitive computationally

Remark

The practical goal is therefore rather to find a "good" instead than the absolute minimum through training.

PAUL SCHERRER INSTITUT

Minimize the Loss Function I

- iterative method called gradient descent
- intuitively, the method corresponds to "walking down the hill" in our many parameter landscape until we reach a (local) minimum
- we use the (discrete) derivative of the cost function to update all the weights and biases incrementally and search for the minimum of the function via tiny steps on the many-dimensional surface.

Minimize the Loss Function II

More specifically, we can update all weights and biases in each step as

$$\theta_\alpha \to \theta_\alpha - \eta \frac{\partial L(\theta)}{\partial \theta_\alpha}.$$

The variable η, also referred to as learning rate, specifies the size of step we use to walk the landscape.

Remark

If η is too small in the beginning, we might get stuck in a local minimum early on, while for too large η we might never find a minimum. The learning rate is a hyperparameter of the training algorithm.

Minimize the Loss Function III

Remark

Note that gradient descent is just a discrete many-variable version of the analytical search for extrema which we know from calculus

While the process of optimizing the many variables of the loss function is mathematically straightforward to understand, it presents a significant numerical challenge:

- for each variational parameter, for instance a weight in the k-th layer $W_{ij}^{[k]}$, the partial derivative $\partial L / \partial W_{ij}^{[k]}$ has to be computed
- this has to be done each time the network is evaluated for a new dataset during training

Minimize the Loss Function IV

- naively, one could assume that the whole network has to be evaluated each time.
- luckily there is an algorithm that allows for an efficient and parallel computation of all derivatives – it is known as backpropagation.

The algorithm derives directly from the chain rule of differentiation for nested functions and is based on two observations:

(1) The loss function is a function of the neural network $F(\boldsymbol{x})$, that is $L \equiv L(F)$.

Minimize the Loss Function V

(2) To determine the derivatives in layer k only the derivatives of the following layer, given as Jacobi matrix

$$\partial \boldsymbol{f}^{[l]} / \partial \boldsymbol{z}^{[l-1]},$$

with $l > k$ and $z^{[l-1]}$ the output of the previous layer, as well as

$$\frac{\partial \boldsymbol{z}^{[k]}}{\partial \theta_\alpha^{[k]}} = \frac{\partial \boldsymbol{g}^{[k]}}{\partial q_i^{[k]}} \frac{\partial q_i^{[k]}}{\partial \theta_\alpha} = \begin{cases} \frac{\partial \boldsymbol{g}^{[k]}}{\partial q_i^{[k]}} z_j^{[k-1]} & \theta_\alpha = W_{ij} \\ \frac{\partial \boldsymbol{g}^{[k]}}{\partial q_i^{[k]}} & \theta_\alpha = b_i \end{cases}$$

are required.

Minimize the Loss Function VI

Remark

- *the calculation of the Jacobi matrix thus has to be performed only once for every update*
- *this is in contrast to the evaluation of the network itself, which is propagating forward, (output of layer n is input to layer $n+1$), we find that a change in the Output propagates backwards though the network*

Minimize the Loss Function VII

Remark

Backpropagation is actually a special case of a set of techniques known as **automatic differentiation** *(AD). AD makes use of the fact that any computer program can be composed of elementary operations (addition, subtraction, multiplication, division) and elementary functions* ($\sin, \exp, \ldots$)*. By repeated application of the chain rule, derivatives of arbitrary order can be computed automatically.*

High Level Gradient Descent Algorithm

1. Initialization: $\theta = \mathcal{N}(\mathbf{0}, \sigma)$
2. Loop until converged
3. compute $\frac{\partial L(\theta)}{\partial \theta_\alpha}$
4. update $\theta \leftarrow \theta - \eta \frac{\partial \mathbf{L}(\theta)}{\partial \theta_\alpha}$
5. return θ

Stochastic Gradient Descent - naive

1. Initialization: $\theta = \mathcal{N}(\mathbf{0}, \sigma)$
2. Loop until converged
3. pick random a single data point j
4. compute $\frac{\partial L_j(\theta)}{\partial \theta_\alpha}$ at j
5. update $\theta \leftarrow \theta - \eta \frac{\partial L(\theta)}{\partial \theta_\alpha}$
6. return θ

Remark

- $\frac{\partial L_j(\theta)}{\partial \theta_\alpha}$ *easy to compute but noisy (stochastic)*

Stochastic Gradient Descent with Mini-Batches

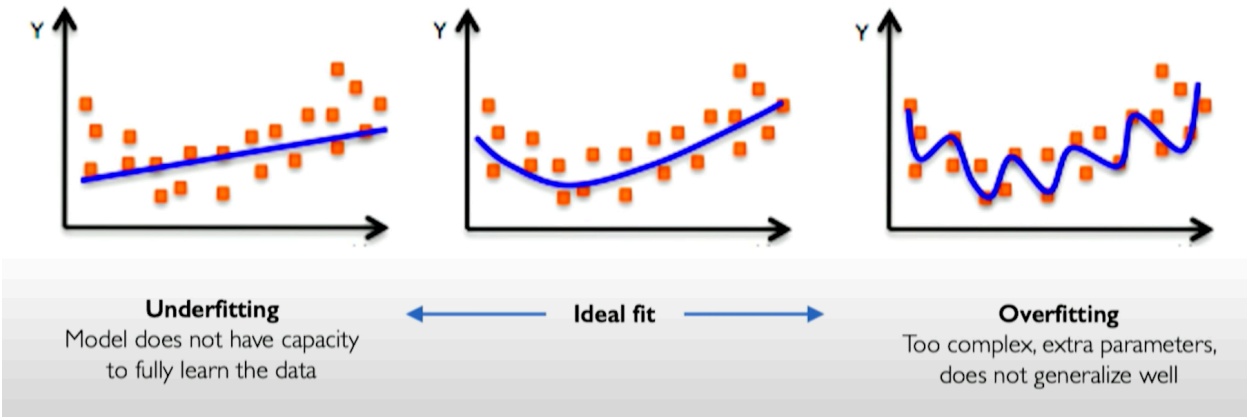
1. Initialization: $\mathbf{W} = \mathcal{N}(\mathbf{0}, \sigma)$
2. Loop until converged
3. pick batch B of data points
4. compute $\frac{\partial L(\theta)}{\partial \theta_\alpha} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial L_k(\theta)}{\partial \theta_\alpha}$
5. update $\theta \leftarrow \theta - \eta \frac{\partial L(\theta)}{\partial \theta_\alpha}$
6. return θ

Remark

- $\frac{\partial J_j(\mathbf{W})}{\partial \mathbf{W}}$ *now still fast & a much better estimate of the true gradient*
- *can be parallelized*

The problem of Overfitting

The problem of Overfitting



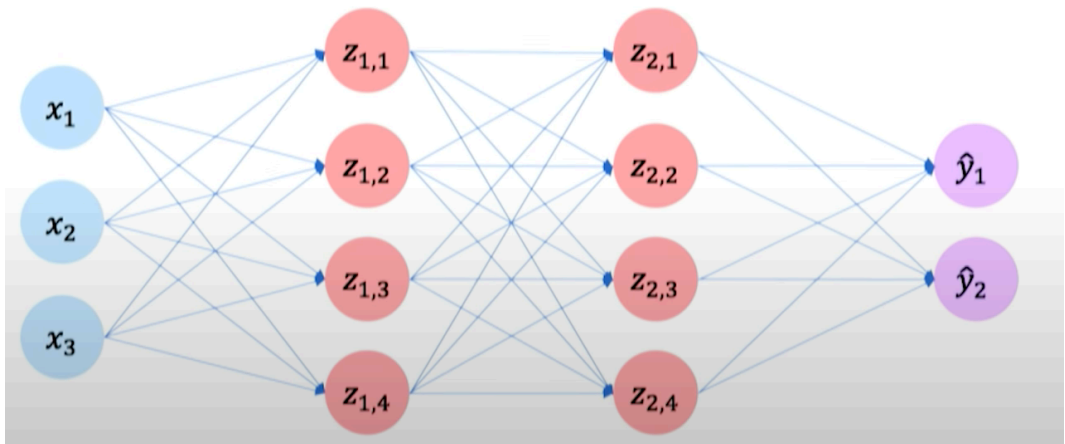
Remark

Solution is are **Regularization** *techniques to constrain our optimization problem in order to discourage complex models. This also helps to improve generalization of our model on unseen data.*

- drop out
- early stopping

Regularization - Drop Out

- during training set randomly some activations to 0
  - typically 50%

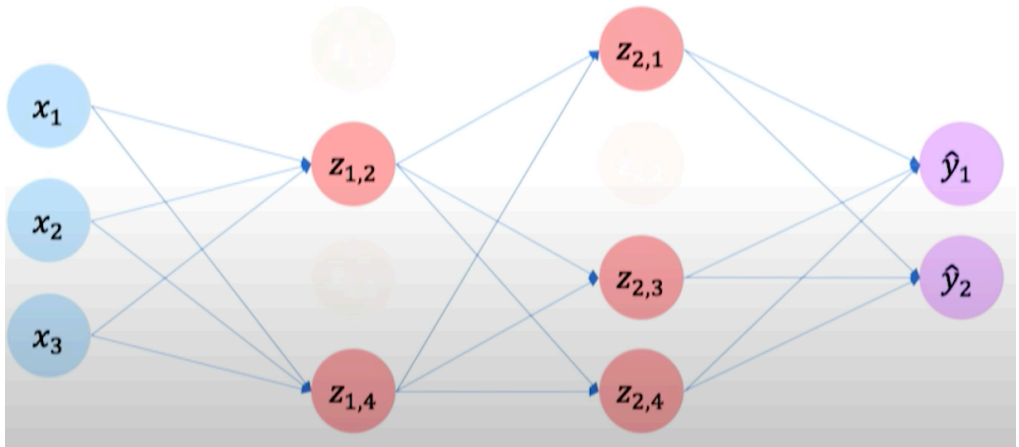


Remark

Forces the network to identify different pathways.

Regularization - Drop Out

- during training set randomly some activations to 0
  - typically 50%

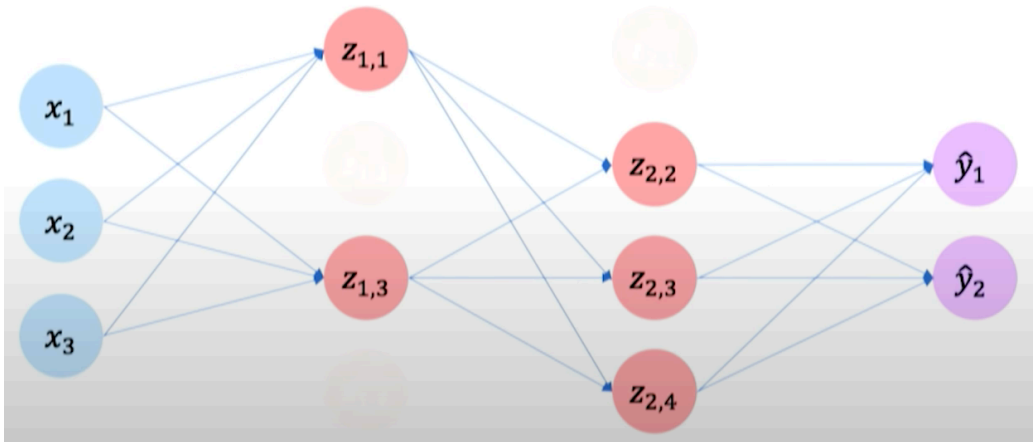


Remark

Forces the network to identify different pathways.

Regularization - Drop Out

- during training set randomly some activations to 0
  - typically 50%

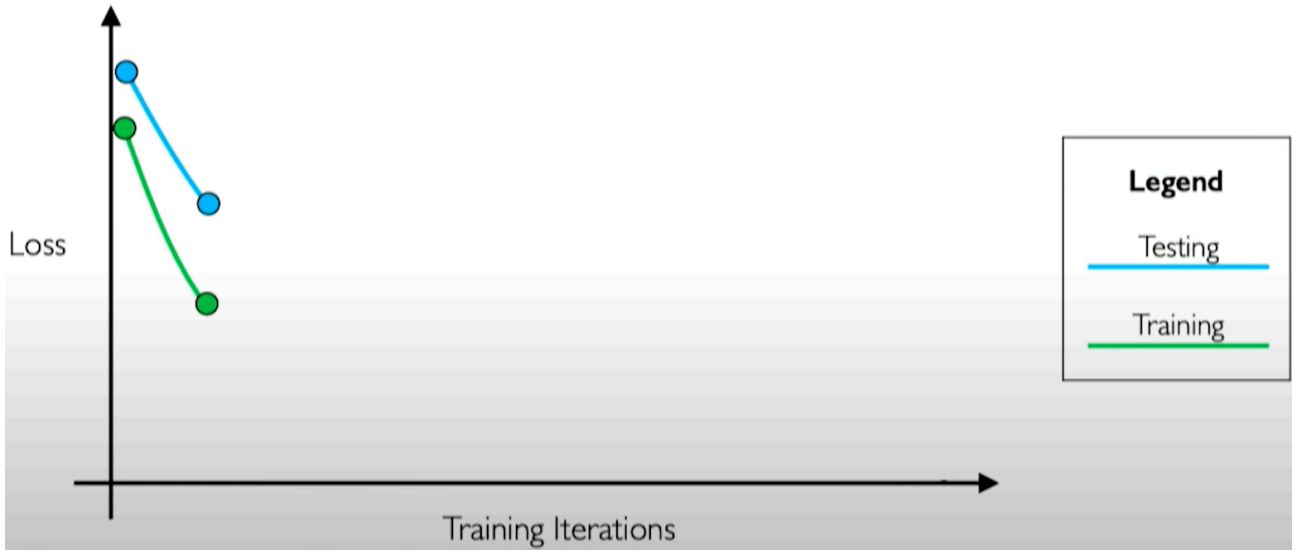


Remark

Forces the network to identify different pathways.

Regularization - Early Stopping

- stop training before we have a chance to overfit.
  - monitor loss function of test data set



Remark

- *training / test dataset split*
- *find point where divergence starts $\Rightarrow$ best model of generalization*

Regularization - Early Stopping

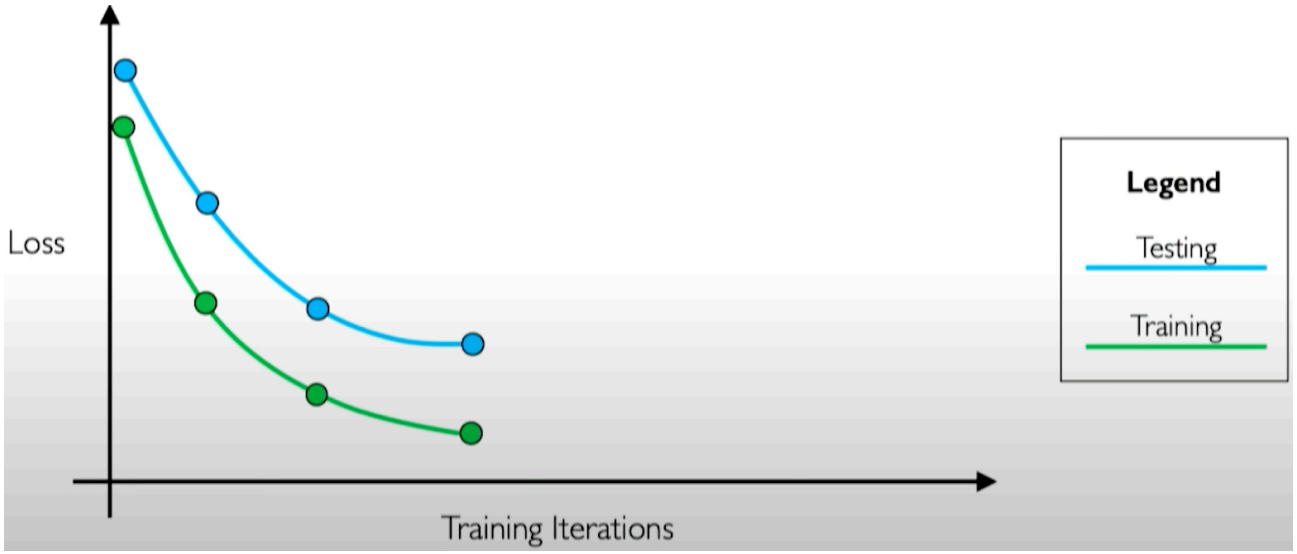
- stop training before we have a chance to overfit.
  - monitor loss function of test data set

Remark

- *training / test dataset split*
- *find point where divergence starts* $\Rightarrow$ *best model of generalization*

Regularization - Early Stopping

- stop training before we have a chance to overfit.
  - monitor loss function of test data set

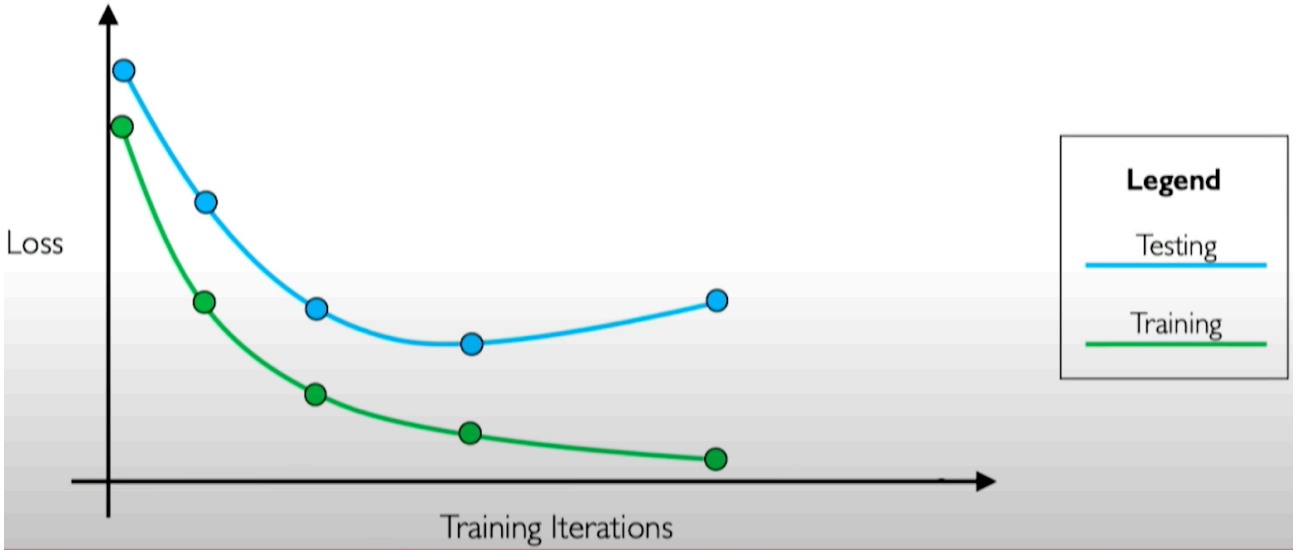


Remark

- *training / test dataset split*
- *find point where divergence starts $\Rightarrow$ best model of generalization*

Regularization - Early Stopping

- stop training before we have a chance to overfit.
  - monitor loss function of test data set

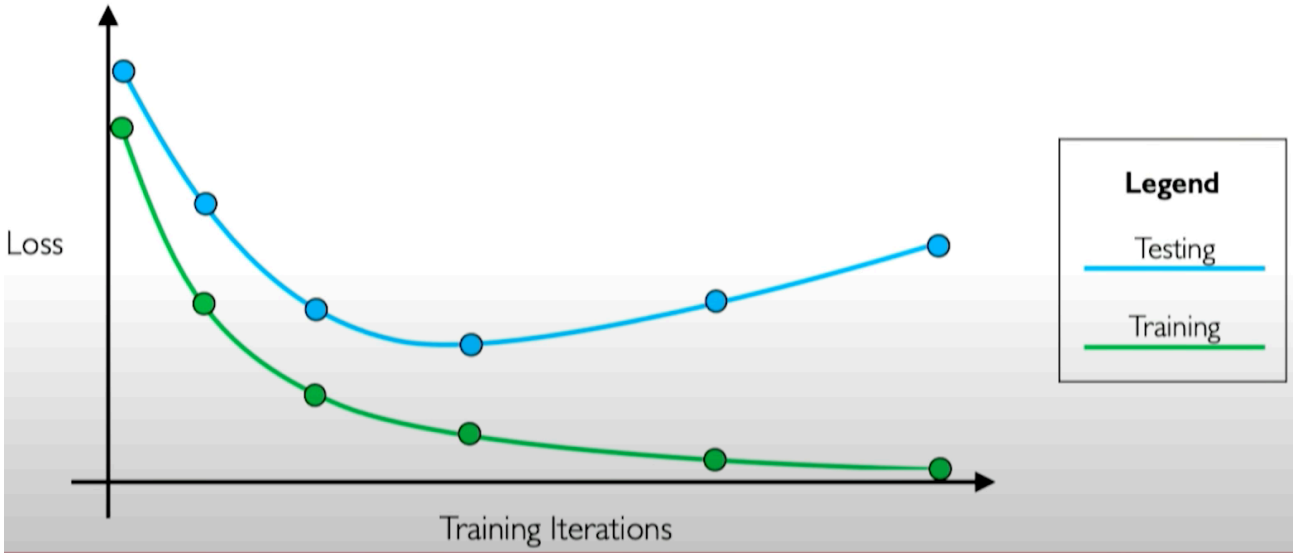


Remark

- *training / test dataset split*
- *find point where divergence starts $\Rightarrow$ best model of generalization*

Regularization - Early Stopping

- stop training before we have a chance to overfit.
  - monitor loss function of test data set



Remark

- *training / test dataset split*
- *find point where divergence starts ⇒ best model of generalization*

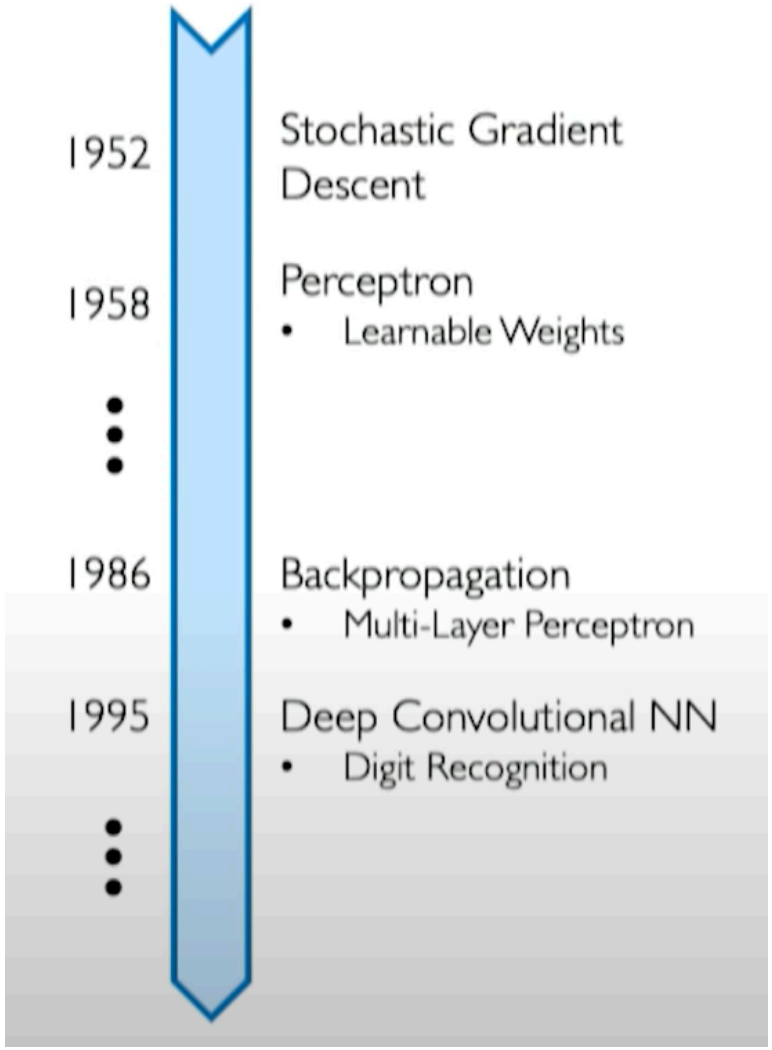
Regularization - Early Stopping

- stop training before we have a chance to overfit.
  - monitor loss function of test data set

Remark

- *training / test dataset split*
- *find point where divergence starts $\Rightarrow$ best model of generalization*

Why Now?


1952
Stochastic Gradient Descent
1958
Perceptron
• Learnable Weights
1986
Backpropagation
• Multi-Layer Perceptron
1995
Deep Convolutional NN
• Digit Recognition

Why Now?

Big Data

- larger data sets
- infrastructure for large data sets

Hardware

- architecture adv. GPUs / TPUs
- HPC (massive parallel)

Software

- improved techniques
- sophisticated toolboxes
- open-source

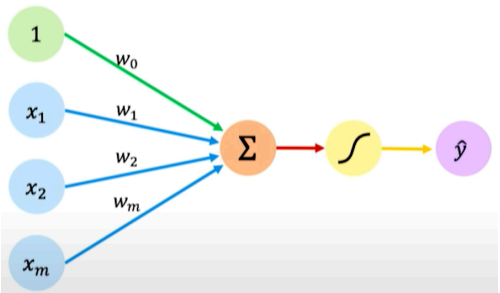
?

Wrapping up

The core things considered are:

The Perceptron

- the structural building block
- non-linear activation function



Neural Networks

- stacking Perceptrons to form a NN
- optimixation through back-propagation

How to Train ?

- adaptive learning
- batching
- regularisation

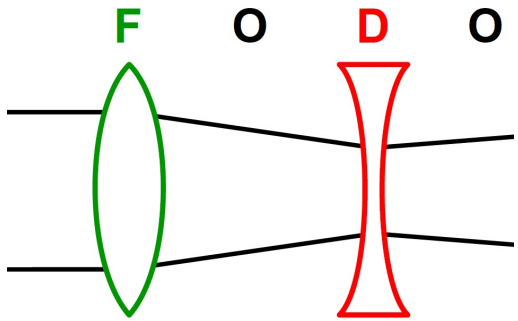
PAUL SCHERRER INSTITUT

FODO Example

References I

Koser Daniel, Waites Loyd, Winklehner Daniel, Frey Matthias, Adelmann Andreas, Jannet Conrad, Input Beam Matching and Beam Dynamics Design Optimizations of the IsoDAR RFQ Using Statistical and Machine Learning Techniques, Frontiers in Physics **10**, https://www.frontiersin.org/article/10.3389/fphy.2022.875889, 2022

Problem Setup I



- we computed the FODO cell simulations in
- ground throughs from OPAL
  `https://gitlab.psi.ch/OPAL/src/wikis/home`
- with the following design parameter and ranges

Problem Setup II

Table: Input beam design variables to the fixed FODO cell lattice generated using OPAL, and the range of their parameter space.

corx	-0.5 to 0.5
cory	-0.5 to 0.5
Beam current [mA]	2 to 10
RMS t [MeV deg]	0.0001 to 0.0005
RMS x [m]	0.001 to 0.005
RMS y [m]	0.001 to 0.005

PAUL SCHERRER INSTITUT

wish me luck ...

PAUL SCHERRER INSTITUT

Published Results

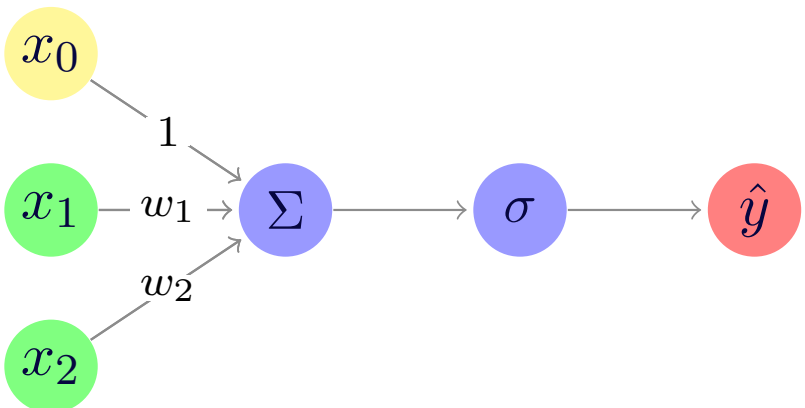
PAUL SCHERRER INSTITUT

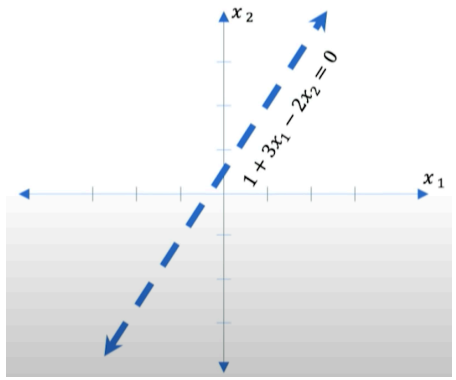
Backup Slides

Example of a Perceptron -1

We have $w_0 = 1$ and $\mathbf{W} = [3 - 2]^T$ i.e. a trained net.

$$\begin{aligned}\hat{y} &= \sigma\left(w_0 + \mathbf{X}^T\mathbf{W}\right) \\ &= \sigma(1 + 3x_1 - 2x_2)\end{aligned}$$

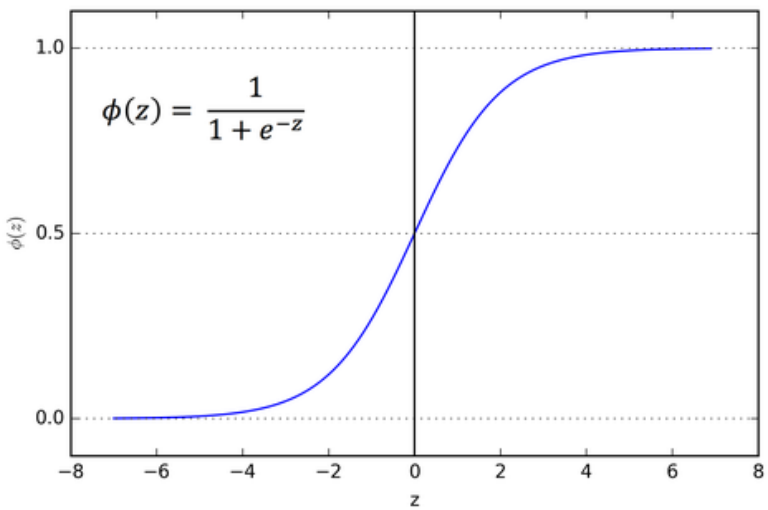


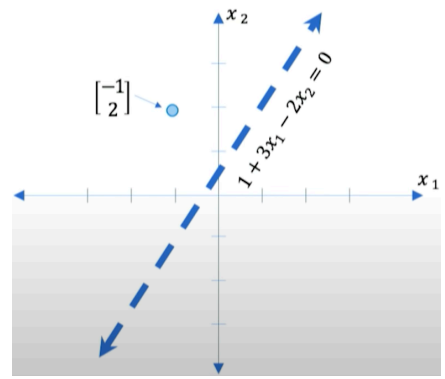


Example of a Perceptron - 2

We have $w_0 = 1$ and $\mathbf{W} = [3 - 2]^T$ i.e. a trained net. In addition we get a datapoint $[-1\ 2]^T$.

$$\hat{y} = \sigma\left(w_0 + \mathbf{X}^T\mathbf{W}\right) \tag{10}$$
$$= \sigma(1 + 3x_1 - 2x_2) \tag{11}$$
$$= \sigma(-6) \approx 0.002 \tag{12}$$





Example of a Perceptron - 2

We have $w_0 = 1$ and $\mathbf{W} = [3 - 2]^T$ i.e. a trained net. In addition we get a datapoint $[-1\ 2]^T$.

$$\hat{y} = \sigma\left(w_0 + \mathbf{X}^T\mathbf{W}\right) \tag{10}$$
$$= \sigma(1 + 3x_1 - 2x_2) \tag{11}$$
$$= \sigma(-6) \approx 0.002 \tag{12}$$