A Historical Overview of Artificial Intelligence (AI)

Yuan YAO

HKUST

Course Infomation

- Course web:
  - https://aifin-hkust.github.io/
- Time and Venure:
  - Lecture: **Wed, 7:30-10:20pm**, **Lecture Theatre G**
- Instructor:
  - **Yuan YAO** <yuany@ust.hk> (https://yao-lab.github.io/)
- Teaching Assistant:
  - **CAO, He** : hcaoaf@connect.ust.hk
  - **LIU, Xuantong** : xliude@connect.ust.hk
- Grading:
  - 3 projects (warmup, midterm, final)
  - 40% (A-AA+)

Course Content

- Supervised Learning:
  - working knowledge about linear regression, classification, logistic regression, decision trees (CART), boosting, random forests, support vector machines, neural networks, etc.
- Unsupervised and Self-supervised Learning:
  - PCA, Generative Models, Generative Adversarial Networks, Denoising Diffusion Models
  - Self-supervision, e.g. masked language models etc.
- Reinforcement Learning:
  - Markov Decision Process and online learning, etc.

- **No exams. Project-based evaluation.**

Course Schedule

Session	Topic	Application & Case Study
1	– Overview of History and Supervised learning	
2	– Introduction to Linear Regression and Classification	– Price and Default Risk Prediction
3	– Model Assessment and Selection with Regularization: Ridge Regression and LASSO	– Model selection and causal recovery
4	– Decision Trees, Random Forests and Boosting	– Credit analysis
5	– Support Vector Machines	– Tutorial on Machine Learning with Python
6	– An Introduction to Convolutional Neural Networks	– Price change patterns
7	– Transfer Learning and Neurostyle	– Google deep dream
8	– An Introduction to Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM)	– Time series prediction
9	– Introduction to Unsupervised Learning: PCA, AutoEncoder, VAE and GANs	– Robust factor analysis – Anomaly detection
10	– An Introduction to Reinforcement Learning	– Deep Reinforcement Learning in Trading
11	– Projects and Class Wrap	

Note: Details may change depending on class progress, development of relevant technologies, as well as information and feedback from students' surveys.

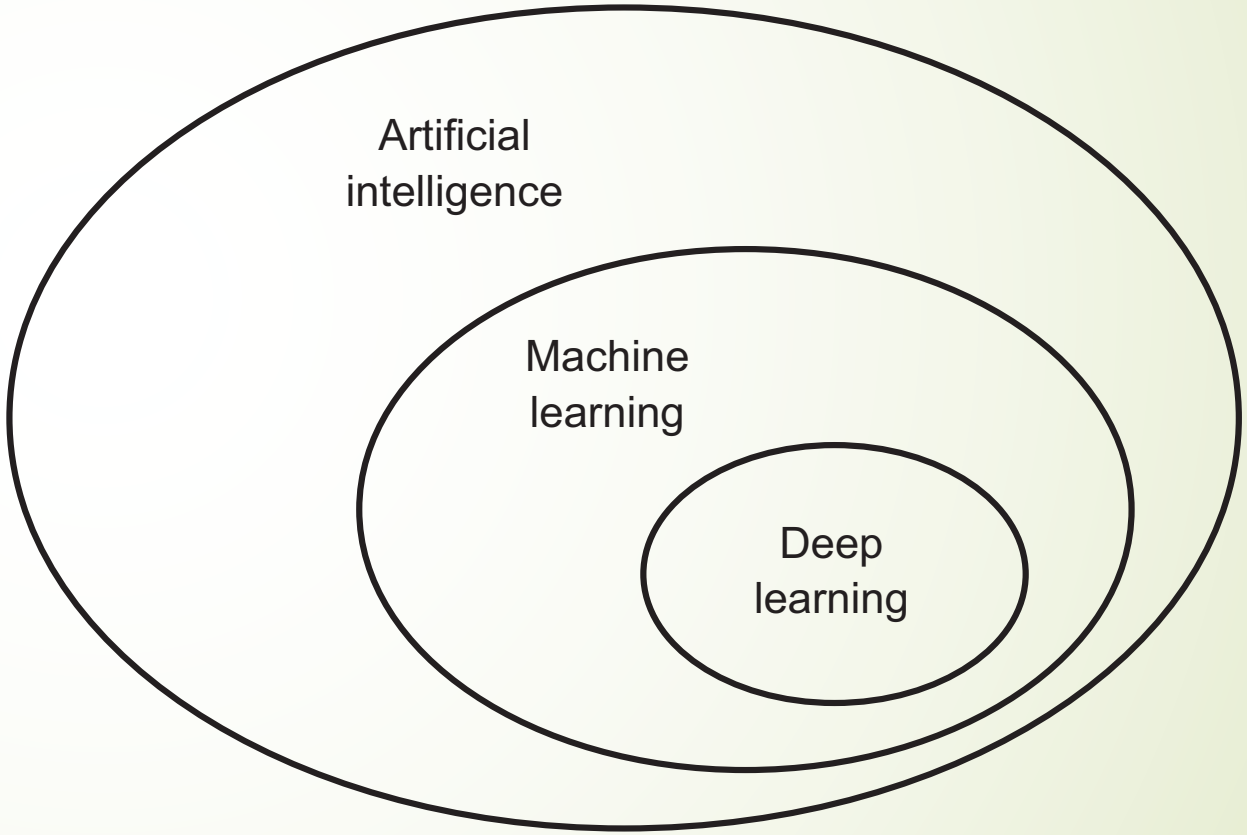
A Brief History of AI, Machine Learning, and Deep Learning

If you asked ChatGPT (3.5):

- **Title: A Historical Overview of Artificial Intelligence (AI)**
- Introduction:
  Artificial Intelligence (AI) has a rich and fascinating history that spans several decades. This article provides a comprehensive overview of the historical milestones and key developments in the field of AI. By tracing the evolution of AI from its inception to the present day, we gain valuable insights into the progression of this groundbreaking discipline.
- The Birth of AI:
  The origins of AI can be traced back to the Dartmouth Conference in 1956, where the term "artificial intelligence" was first coined. This event marked the beginning of a new era, with researchers envisioning the possibility of creating machines that could exhibit intelligent behavior.
- Early AI Approaches:
  In the 1950s and 1960s, researchers explored various approaches to AI, such as symbolic reasoning and logical inference. The development of the Logic Theorist program by Allen Newell and Herbert Simon and the General Problem Solver by Newell and J.C. Shaw were significant milestones during this period.
- The Rise of Machine Learning:
  In the 1980s and 1990s, there was a shift in focus towards machine learning approaches within AI. The emergence of expert systems and the development of algorithms like the backpropagation algorithm for training neural networks led to significant advancements in pattern recognition and decision-making.
- Knowledge-Based Systems:
  During the 1980s, knowledge-based systems gained prominence in AI research. These systems utilized expert knowledge and rule-based reasoning to solve complex problems. The development of the MYCIN system for medical diagnosis and the DENDRAL system for chemical analysis were notable achievements in this era.
- AI Winter and Resurgence:
  In the late 1980s and early 1990s, AI faced a period of reduced funding and disillusionment, known as the "AI winter." However, advancements in computing power and the emergence of new techniques, such as Bayesian networks and support vector machines, sparked a resurgence of interest in AI research in the late 1990s.
- Big Data and Deep Learning:
  In recent years, the availability of vast amounts of data and the development of powerful computational resources have revolutionized AI. Deep learning, a subfield of machine learning focused on neural networks with multiple layers, has achieved remarkable success in areas such as image and speech recognition, natural language processing, and autonomous vehicles.
- AI in the Present and Future:
  AI is now an integral part of our daily lives, with applications ranging from virtual assistants and recommendation systems to autonomous robots and self-driving cars. The field continues to evolve rapidly, with ongoing research in areas like explainable AI, reinforcement learning, and ethical considerations surrounding AI deployment.
- Conclusion:
  The historical journey of AI showcases the remarkable progress made in this field. From its early beginnings to the present day, AI has transformed the way we interact with technology and holds immense potential for future advancements. By understanding its history, we gain a deeper appreciation for the challenges overcome and the possibilities that lie ahead in the exciting world of artificial intelligence.

Artificial Intelligence, Machine Learning, and Deep Learning

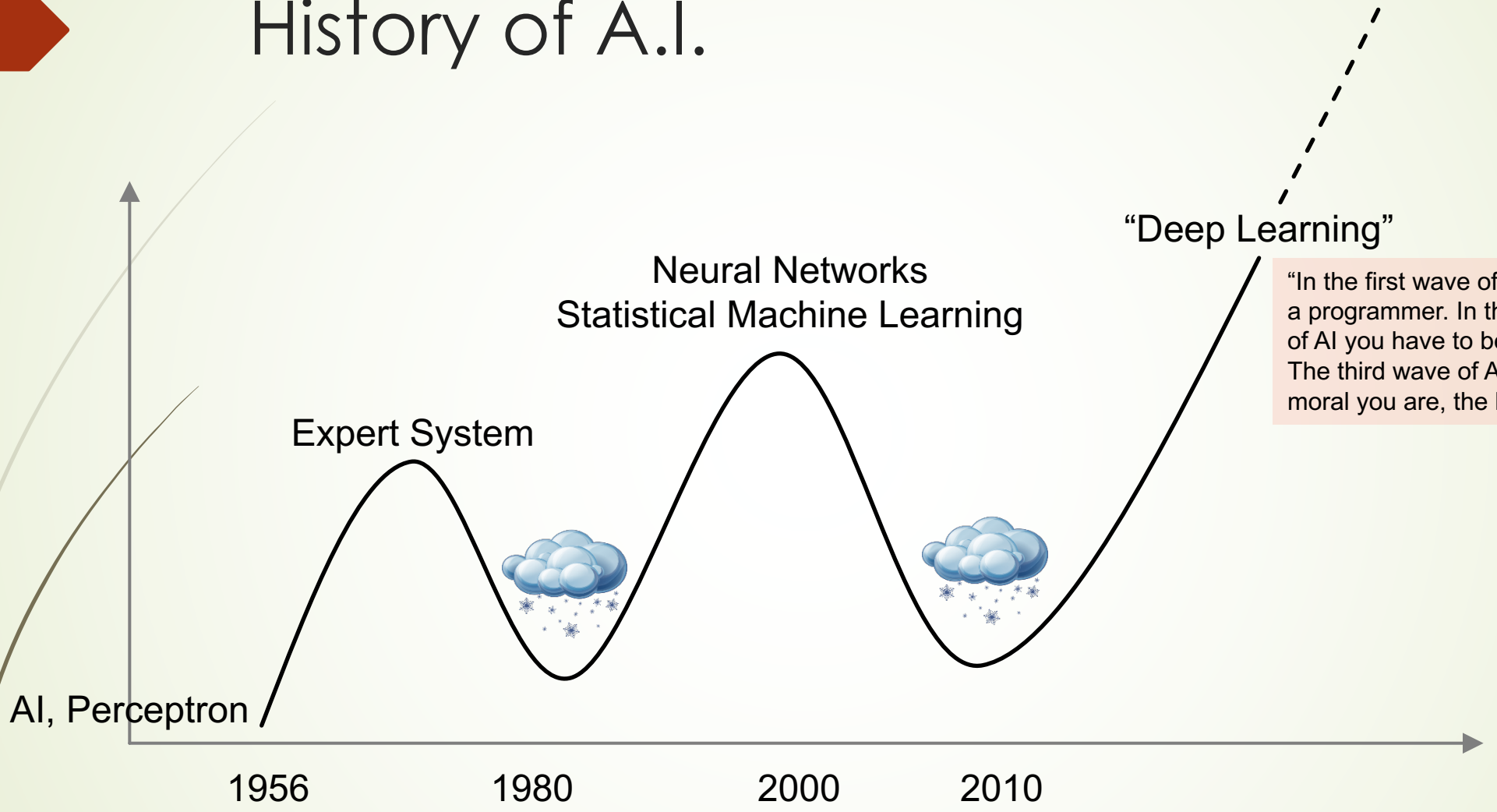
- AI is born in 1950s, when a handful of pioneers from the nascent field of computer science started asking **whether computers could be made to "think"**—a question whose ramifications we're still exploring today.



A brief history of AI

- **1943:** McCulloch & Pits proposed a boolean circuit model of neurons
- **1949:** Donald Hebb proposed **Hebbian learning rule**.
- **1950:** Alan Turing published **"Computing Machinery and Intelligence"** with **Turing test**.
- **1956:** John McCarthy at the Dartmouth Conference coined terminology "**Artificial Intelligence**"
- **1957**: Rosenblatt invented **Perceptron**
- **1960s:** golden years till **1969 Minsky-Papert's** critical book **Perceptron**
- **1970s**: the first AI winter
- **1980s**: boom of AI with **Expert System**
- **1990s:** the second AI winter, rise of **statistical machine learning**
- **1997: IBM Deep Blue** beats world chess champion Kasparov
- **2012:** return of neural networks as **deep learning** (speech, ImageNet in computer vision, NLP, …)
- **2016-2017: Google AlphaGo "Lee" and Zero**
- **2020: Google AlphaFold**
- **2022: OpenAI ChatGPT**
- …

History of A.I.

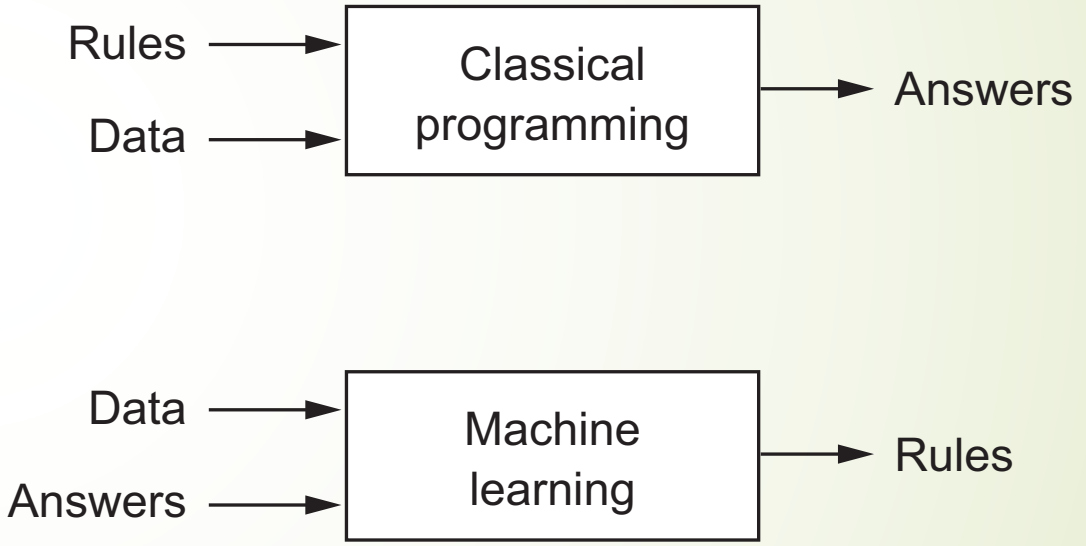


“In the first wave of AI you had to be a programmer. In the second wave of AI you have to be a data scientist. The third wave of AI—the more moral you are, the better.”

Source: Dr. Lee Kai-Fu • Sinovation Ventures and 2018 Conference on Neural Information Processing Systems

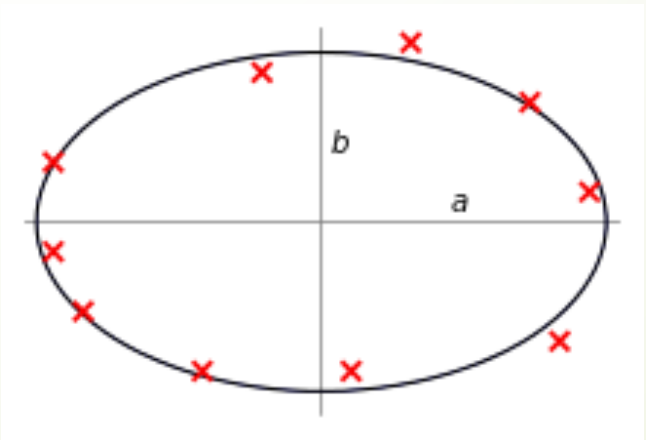
Statistical Machine Learning is a new paradigm of computer programming

- During 1950s-1980s, two competitive ideas of realizing AI exist
  - Rule based inference, or called **Expert System**
  - Statistics based inference, or called **Machine Learning**
- 1990s- Machine Learning becomes dominant



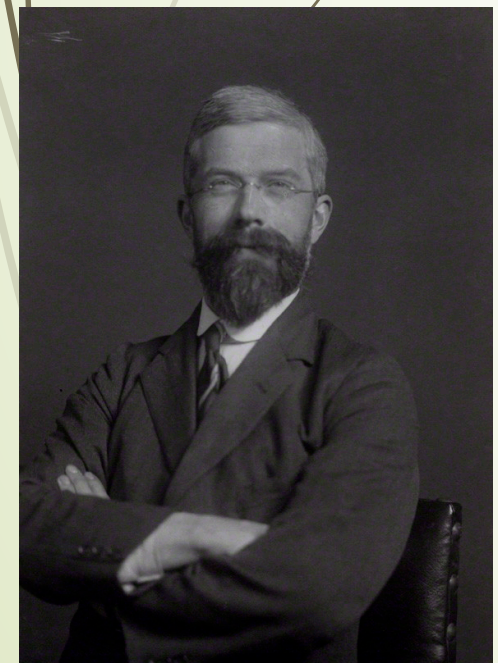
The 1st machine learning method: Least Squares

- Invention:
  - **Carl Friederich Gauss** (~1795/1809/1810),
  - Adrien-Marie Legendre (1805)
  - Robert Adrain (1808)
- Application:
  - Prediction of the location of asteroid Ceres after it emerged from behind the sun (Franz Xaver von Zach 1801)
  - Orbits of planets, Newton Laws
  - Statistics,
  - …



Fisher's Maximum Likelihood Principle (1912-1922)

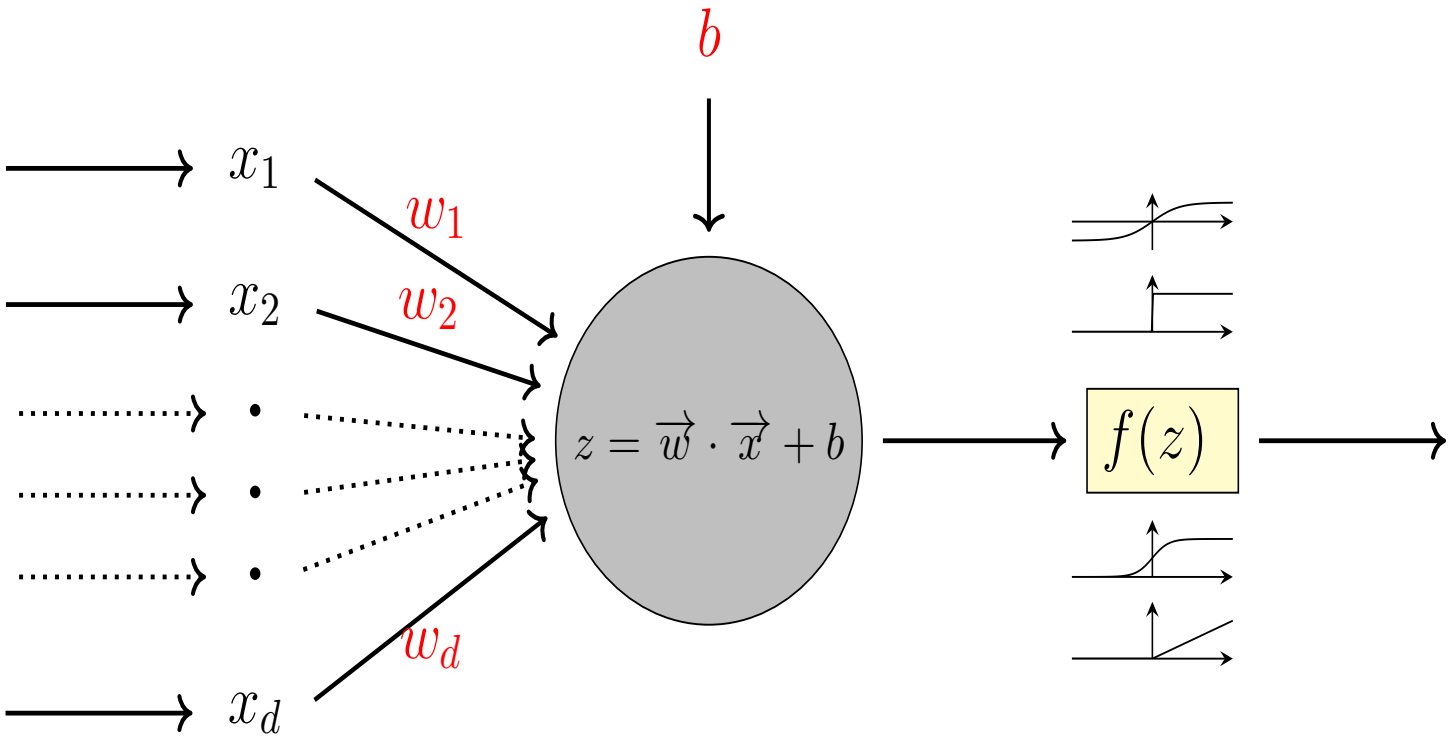
- **The least square method is the maximum likelihood estimate** (most probable values of the unknown parameters) when the noise is Gaussian.



- Fisher, R. A. (1912) **On an absolute criterion for fitting frequency curves**. *Messenger of Mathematics* 41:155-160.
- Fisher, R. A. (1922). **On the mathematical foundations of theoretical statistics**. *Philos. Trans. Roy. Soc. London Ser. A* 222:309-368.
- Aldrich, John (1997). **R. A. Fisher and the Making of Maximum Likelihood 1912 -- 1922**. *Statistical Science*, 12(3):162-176.

The 1st neural network: Perceptron

- Invented by Frank Rosenblatt (1957)



The Perceptron Algorithm for classification

$$\ell(w) = -\sum_{i \in \mathcal{M}_w} y_i \langle w, \mathbf{x}_i \rangle, \quad \mathcal{M}_w = \{i : y_i \langle \mathbf{x}_i, w \rangle < 0, y_i \in \{-1, 1\}\}.$$

The Perceptron Algorithm is a *Stochastic Gradient Descent* method (**Robbins-Monro 1951**, *Ann. Math. Statist.* 22(3): 400-407):

$$\begin{aligned} w_{t+1} &= w_t - \eta_t \nabla_i \ell(w) \\ &= \begin{cases} w_t - \eta_t y_i \mathbf{x}_i, & \text{if } y_i w_t^T \mathbf{x}_i < 0, \\ w_t, & \text{otherwise.} \end{cases} \end{aligned}$$

Finiteness of Stopping Time and Margin

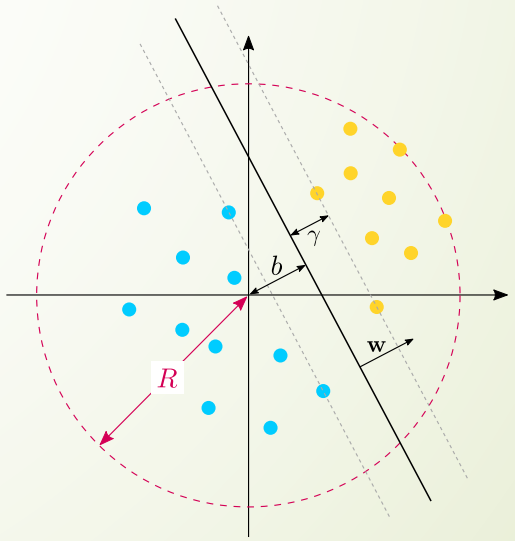
The perceptron convergence theorem was proved by Block (1962) and Novikoff (1962). The following version is based on that in Cristianini and Shawe-Taylor (2000).

Theorem 1 (Block, Novikoff). *Let the training set* $S = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_n, t_n)\}$ *be contained in a sphere of radius* R *about the origin. Assume the dataset to be linearly separable, and let* $\mathbf{w}_{\text{opt}}$, $\|\mathbf{w}_{\text{opt}}\| = 1$, *define the hyperplane separating the samples, having functional margin* $\gamma > 0$. *We initialise the normal vector as* $\mathbf{w}_0 = \mathbf{0}$. *The number of updates,* k, *of the perceptron algorithms is then bounded by*

$$k \leq \left(\frac{2R}{\gamma}\right)^2. \tag{10}$$

Input ball: $R = \max_i \|\mathbf{x}_i\|.$

Margin: $\gamma := \min_i y_i f(x_i)$



Hilbert's 13th Problem

Algebraic equations (under a suitable transformation) of degree up to 6 can be solved by functions of two variables. What about

$$x^7 + ax^3 + bx^2 + cx + 1 = 0?$$

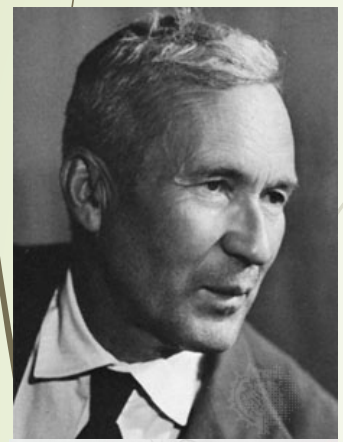
Hilbert's conjecture: $x(a, b, c)$ cannot be expressed by a superposition (sums and compositions) of bivariate functions.

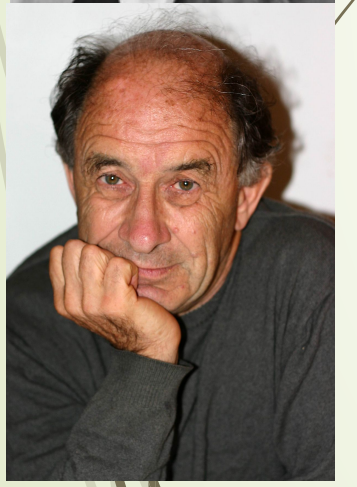
Question: can every continuous (analytic, C^∞, etc) function of n variables be represented as a superposition of continuous (analytic, C^∞, etc) functions of $n-1$ variables?

Theorem (D. Hilbert)

There is an analytic function of three variables that cannot be expressed as a superposition of bivariate ones.

Kolmogorov's Superposition Theorem





Theorem (A. Kolmogorov, 1956; V. Arnold, 1957)

Given $n \in \mathbb{Z}^+$, every $f_0 \in C([0,1]^n)$ can be reprensented as

$$f_0(x_1, x_2, \cdots, x_n) = \sum_{q=1}^{2n+1} g_q \left(\sum_{p=1}^{n} \phi_{pq}(x_p) \right),$$

where $\phi_{pq} \in C[0,1]$ are increasing functions independent of f_0 and $g_q \in C[0,1]$ depend on f_0.

- Can choose g_q to be all the same $g_q \equiv g$ (Lorentz, 1966).
- Can choose ϕ_{pq} to be Hölder or Lipschitz continuous, but not C^1 (Fridman, 1967).
- Can choose $\phi_{pq} = \lambda_p \phi_q$ where $\lambda_1, \cdots, \lambda_n > 0$ and $\sum_p \lambda_p = 1$ (Sprecher, 1972).

If f is a multivariate continuous function, then f can be written as a superposition of composite functions of mixtures of continuous functions of single variables:
finite **composition** of continuous functions of a **single variable** and the **addition**.

Kolmogorov's Exact Representation is not stable or smooth

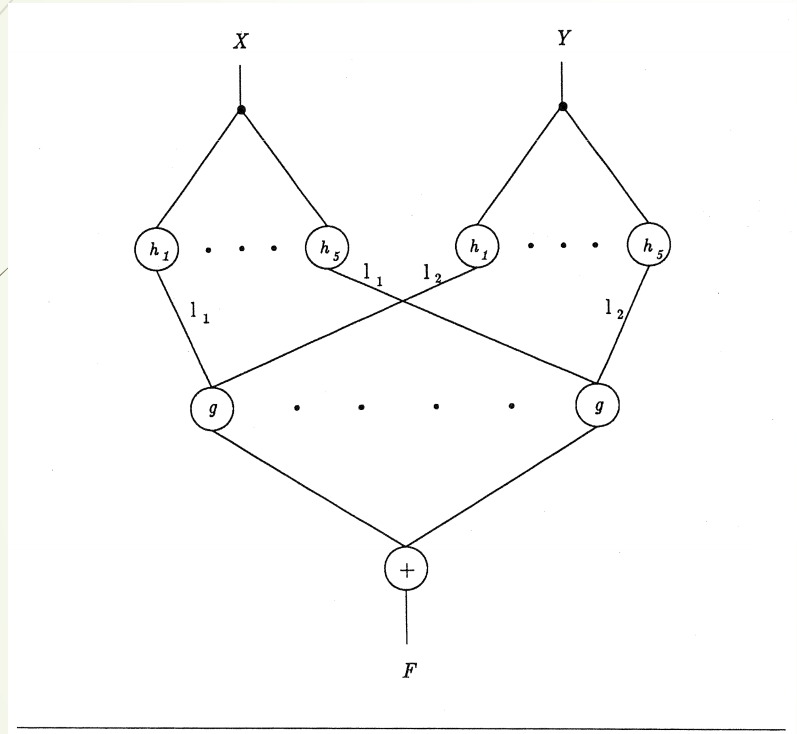


Figure 1: The network representation of an improved version of Kolmogorov's theorem, due to Kahane (1975). The figure shows the case of a bivariate function. The Kahane's representation formula is $f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} g[\sum_{p=1}^{n} l_p h_q(x_p)]$ where h_q are strictly monotonic functions and l_p are strictly positive constants smaller than 1.

- [Girosi-Poggio'1989] Representation Properties of Networks: Kolmogorov's Theorem Is Irrelevant, https://www.mitpressjournals.org/doi/pdf/10.1162/neco.1989.1.4.465
- Lacking smoothness in *h* and *g* [Vitushkin'1964] fails to guarantee the **generalization ability (stability)** against noise and perturbations
- The representation is **not universal** in the sense that *g* and *h* both depend on the function F to be represented.

A Simplified illustration by David McAllester

A Simpler, Similar Theorem

For (possibly discontinuous) $f : [0,1]^N \to \mathbb{R}$ there exists (possibly discontinuous) $g, h_i : \mathbb{R} \to \mathbb{R}$.

$$f(x_1, \ \ldots, \ x_N) = g\left(\sum_i h_i(x_i)\right)$$

Proof: Select h_i to spread out the digits of its argument so that $\sum_i h_i(x_i)$ contains all the digits of all the x_i.

Universal Approximate Representation

[Cybenko'1989, Hornik et al. 1989, Poggio-Girosi'1989, ...]

For continuous $f : [0,1]^N \to \mathbb{R}$ and $\varepsilon > 0$ there exists

$$F(x) = \alpha^\top \sigma(Wx + \beta)$$

$$= \sum_i \alpha_i \sigma \left(\sum_j W_{i,j}\, x_j + \beta_i \right)$$

such that for all x in $[0,1]^N$ we have $|F(x) - f(x)| < \varepsilon$.

Complexity (regularity, smoothness) thereafter becomes the central pursuit in Approximation Theory.

KAN: Kolmogorov-Arnold Networks

KAN: Kolmogorov–Arnold Networks

Ziming Liu[1,4*] **Yixuan Wang**[2] **Sachin Vaidya**[1] **Fabian Ruehle**[3,4]
James Halverson[3,4] **Marin Soljačić**[1,4] **Thomas Y. Hou**[2] **Max Tegmark**[1,4]
[1] Massachusetts Institute of Technology
[2] California Institute of Technology
[3] Northeastern University
[4] The NSF Institute for Artificial Intelligence and Fundamental Interactions

Abstract

Inspired by the Kolmogorov-Arnold representation theorem, we propose Kolmogorov-Arnold Networks (KANs) as promising alternatives to Multi-Layer Perceptrons (MLPs). While MLPs have *fixed* activation functions on *nodes* ("neurons"), KANs have *learnable* activation functions on *edges* ("weights"). KANs have no linear weights at all – every weight parameter is replaced by a univariate function parametrized as a spline. We show that this seemingly simple change makes KANs outperform MLPs in terms of accuracy and interpretability, on small-scale AI + Science tasks. For accuracy, smaller KANs can achieve comparable or better accuracy than larger MLPs in function fitting tasks. Theoretically and empirically, KANs possess faster neural scaling laws than MLPs. For interpretability, KANs can be intuitively visualized and can easily interact with human users. Through two examples in mathematics and physics, KANs are shown to be useful "collaborators" helping scientists (re)discover mathematical and physical laws. In summary, KANs are promising alternatives for MLPs, opening opportunities for further improving today's deep learning models which rely heavily on MLPs.

Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a) *fixed* activation functions on *nodes*; *learnable* weights on *edges*	(b) *learnable* activation functions on *edges*; sum operation on *nodes*
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)	(c) MLP(x): $\mathbf{W}_3$, σ_2, $\mathbf{W}_2$, σ_1, $\mathbf{W}_1$, x; *nonlinear, fixed*; *linear, learnable*	(d) KAN(x): $\mathbf{\Phi}_3$, $\mathbf{\Phi}_2$, $\mathbf{\Phi}_1$, x; *nonlinear, learnable*

Figure 0.1: Multi-Layer Perceptrons (MLPs) vs. Kolmogorov-Arnold Networks (KANs)

Ziming Liu et al. arXiv:2404.19756

KAN vs. MLP?

KAN or MLP: A Fairer Comparison

Runpeng Yu, Weihao Yu, and Xinchao Wang
National University of Singapore

https://github.com/yu-rp/KANbeFair

Abstract

This paper does not introduce a novel method. Instead, it offers a fairer and more comprehensive comparison of KAN and MLP models across various tasks, including machine learning, computer vision, audio processing, natural language processing, and symbolic formula representation. Specifically, we control the number of parameters and FLOPs to compare the performance of KAN and MLP. Our main observation is that, except for symbolic formula representation tasks, MLP generally outperforms KAN. We also conduct ablation studies on KAN and find that its advantage in symbolic formula representation mainly stems from its B-spline activation function. When B-spline is applied to MLP, performance in symbolic formula representation significantly improves, surpassing or matching that of KAN. However, in other tasks where MLP already excels over KAN, B-spline does not substantially enhance MLP's performance. Furthermore, we find that KAN's forgetting issue is more severe than that of MLP in a standard class-incremental continual learning setting, which differs from the findings reported in the KAN paper. We hope these results provide insights for future research on KAN and other MLP alternatives.

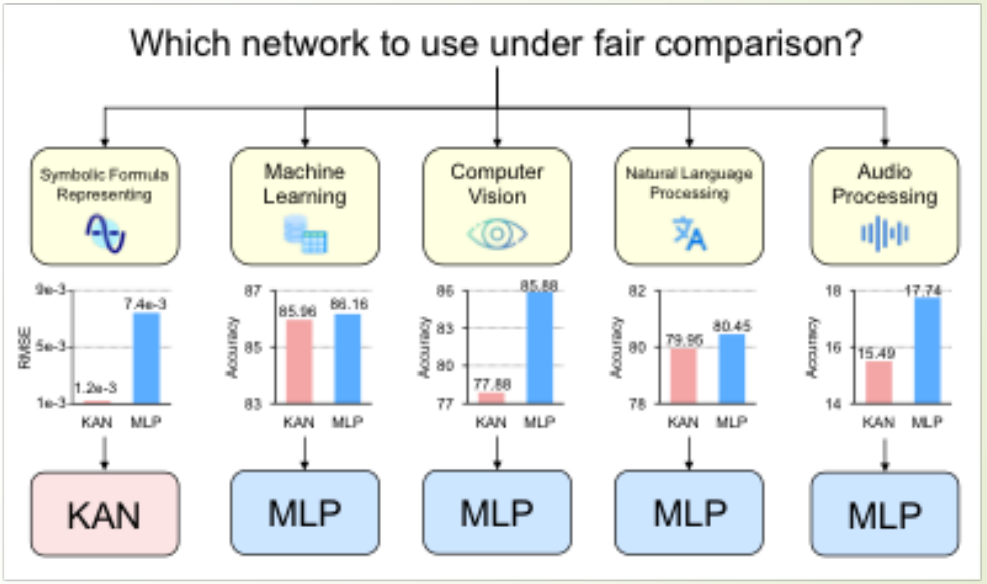


Figure 1: Performance comparison between KAN and MLP under fair setup. MLP yields higher average accuracy in machine learning, computer vision, natural language processing, and audio processing, while KAN leads to lower average root mean square error. For the Symbolic Formula Representation task, a lower RMSE is better.

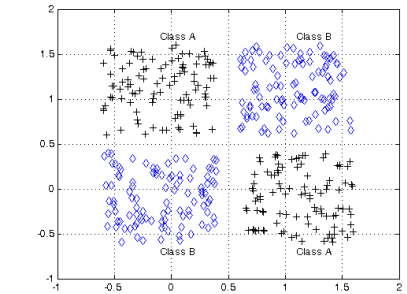
Runpeng Yu et al. arXiv: 2407.16674

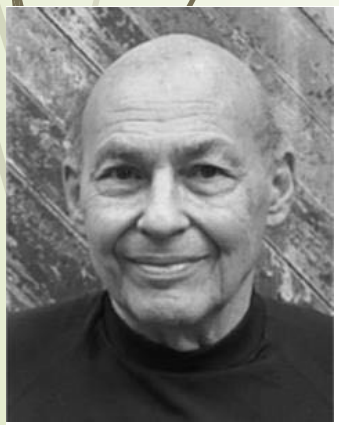
Locality or Sparsity of Computation

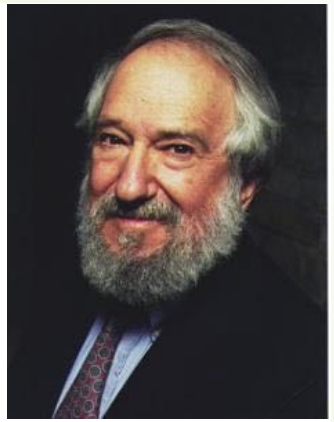
Minsky and Papert, 1969

Perceptron can't do **XOR** classification

Perceptron needs infinite global information to compute **connectivity**





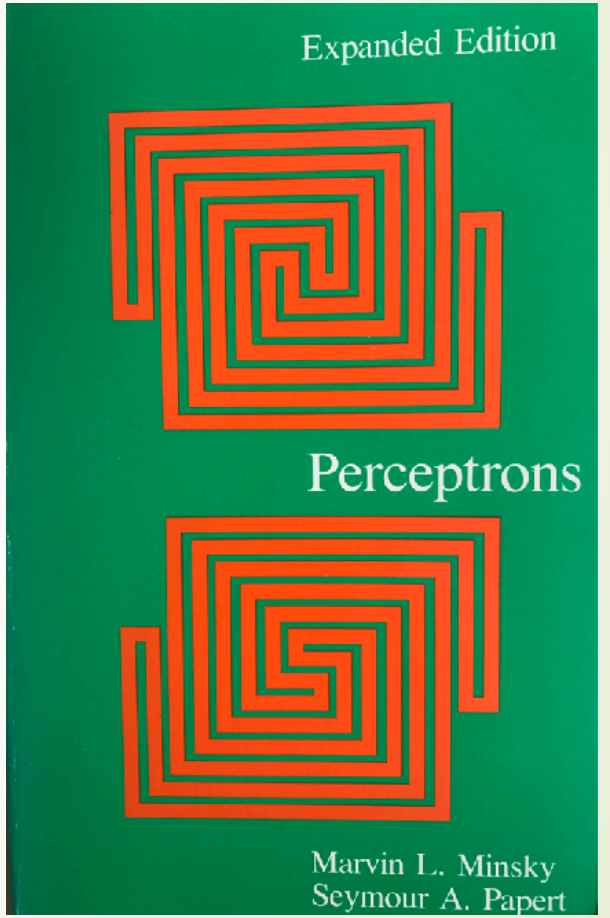


Marvin Minsky Seymour Papert

Locality or **Sparsity** is important:

Locality in time?

Locality in space?



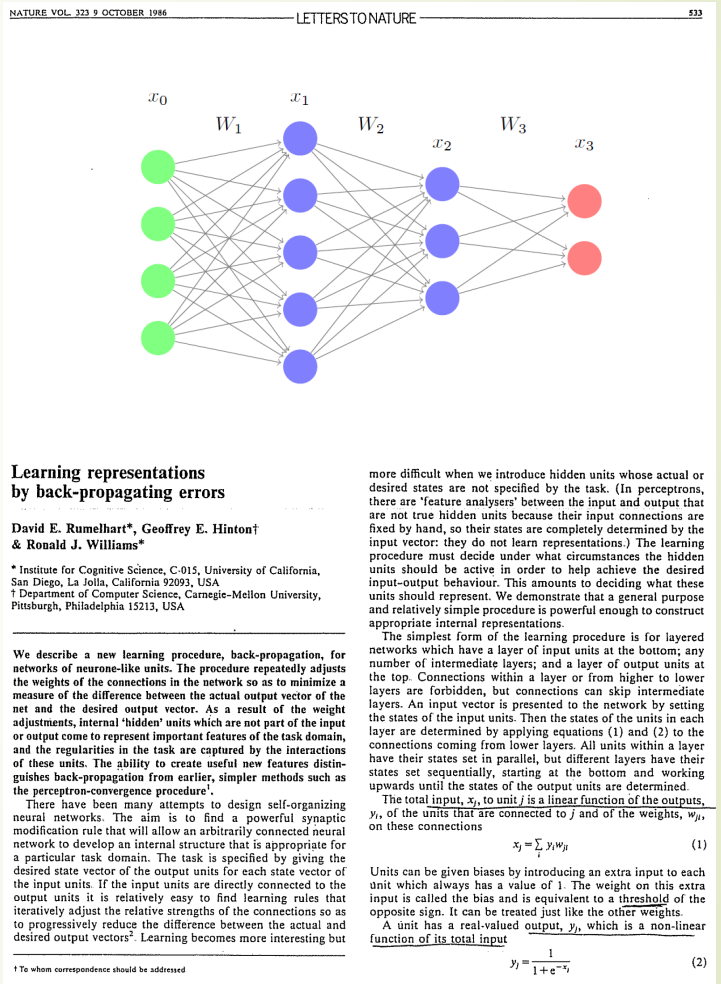
Multilayer Perceptrons (MLP) and Back-Propagation (BP) Algorithms

D.E. Rumelhart, G. Hinton, R.J. Williams (1986)
Learning representations by back-propagating errors, Nature, 323(9): 533-536

BP algorithms as **stochastic gradient descent** algorithms (**Robbins–Monro 1950; Kiefer-Wolfowitz 1951**) with Chain rules of Gradient maps

Deep network may classify **XOR**. Yet **topology**?

We address complexity and geometric invariant properties first.



NATURE VOL. 323 9 OCTOBER 1986 — LETTERS TO NATURE — 533

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors[2]. Learning becomes more interesting but more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \quad (1)$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j, which is a non-linear function of its total input

$$y_j = \frac{1}{1+e^{-x_j}} \quad (2)$$

† To whom correspondence should be addressed

Parallel Distributed Processing

by Rumelhart and McClelland, 1986

Minsky and Papert set out to show which functions can and cannot be computed by this class of machines. They demonstrated, in particular, that such perceptrons are unable to calculate such mathematical functions as parity (whether an odd or even number of points are on in the retina) or the topological function of connectedness (whether all points that are on are connected to all other points that are on either directly or via other points that are also on) without making use of absurdly large numbers of predicates. The analysis is extremely elegant and demonstrates the importance of a mathematical approach to analyz-

of multilayer networks that compute parity). Similarly, it is not difficult to develop networks capable of solving the connectedness or inside/outside problem. Hinton and Sejnowski have analyzed a version of such a network (see Chapter 7).

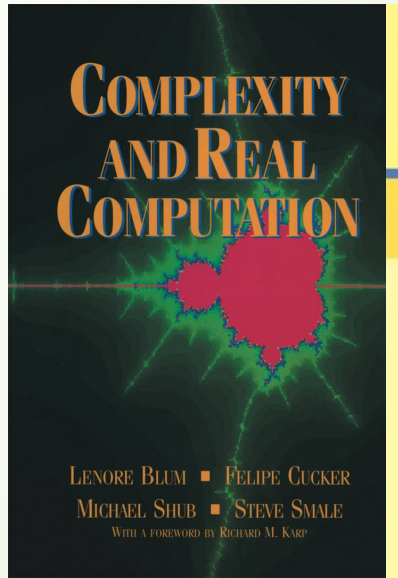
Essentially, then, although Minsky and Papert were exactly correct in their analysis of the *one-layer perceptron,* the theorems don't apply to systems which are even a little more complex. In particular, it doesn't apply to multilayer systems nor to systems that allow feedback loops.

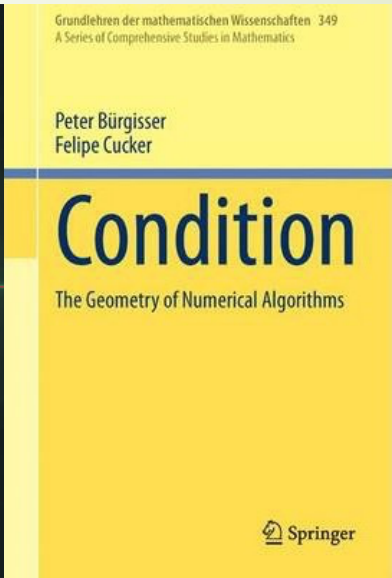
Topology can be learned with finite information if the manifold is *stable*

Blum-Shub-Smale models of Real Computation

A Model of Real Computation

- Starting from **Blum, Shub, Smale** (1989)
- It admits inputs and operations (addition, substraction, multiplication, and (in the case of fields) division) of **real (complex) numbers** with *infinite precision*
- "The key importance of the **condition number**, which measures the closeness of a problem instance to the manifold of ill-posed instances, is clearly developed." – Richard Karp





The Condition Number of a Manifold

Throughout our discussion, we associate to $\mathcal{M}$ a condition number $(1/\tau)$ where τ is defined as the largest number having the property: The open normal bundle about $\mathcal{M}$ of radius r is embedded in $\mathbb{R}^N$ for every $r < \tau$. Its image Tub_τ is a tubular neighborhood of $\mathcal{M}$ with its canonical projection map

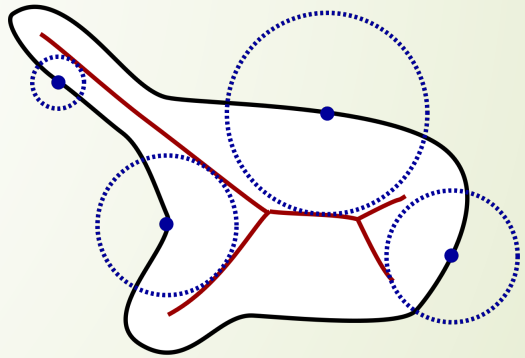
$$\pi_0 : \text{Tub}_\tau \to \mathcal{M}.$$

Smallest Local Feature Size

$G = \{x \in \mathbb{R}^N$ such that $\exists$ distinct $p, q \in \mathcal{M}$ where $d(x, \mathcal{M}) = \|x - p\| = \|x - q\|\}$,

where $d(x, \mathcal{M}) = \inf_{y \in \mathcal{M}} \|x - y\|$ is the distance of x to $\mathcal{M}$. The closure of G is called the medial axis and for any point $p \in \mathcal{M}$ the local feature size $\sigma(p)$ is the distance of p to the medial axis. Then it is easy to check that

$$\tau = \inf_{p \in \mathcal{M}} \sigma(p).$$



Find Homology with Finite Samples

[Niyogi, Smale, Weinberger (2008)]

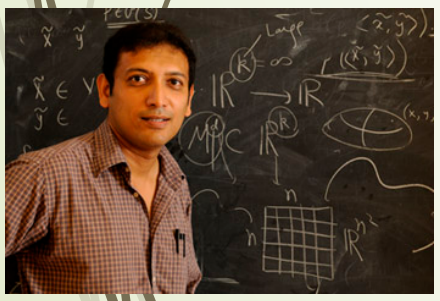
Theorem 3.1 *Let $\mathcal{M}$ be a compact submanifold of $\mathbb{R}^N$ with condition number τ. Let $\bar{x} = \{x_1, \ldots, x_n\}$ be a set of n points drawn in i.i.d. fashion according to the uniform probability measure on $\mathcal{M}$. Let $0 < \epsilon < \tau/2$. Let $U = \bigcup_{x \in \bar{x}} B_\epsilon(x)$ be a correspondingly random open subset of $\mathbb{R}^N$. Then for all*

$$n > \beta_1 \left(\log(\beta_2) + \log\left(\frac{1}{\delta}\right) \right),$$

the homology of U equals the homology of $\mathcal{M}$ with high confidence (probability $> 1 - \delta$).

$$\beta_1 = \frac{vol(\mathcal{M})}{(\cos^k(\theta_1))vol(B^k_{\epsilon/4})} \quad \text{and} \quad \beta_2 = \frac{vol(\mathcal{M})}{(\cos^k(\theta_2))vol(B^k_{\epsilon/8})}.$$

Here k is the dimension of the manifold $\mathcal{M}$ and $vol(B^k_\epsilon)$ denotes the k-dimensional volume of the standard k-dimensional ball of radius ϵ. Finally, $\theta_1 = \arcsin(\epsilon/8\tau)$ and $\theta_2 = \arcsin(\epsilon/16\tau)$.



Partha Niyogi@Chiccago, 1967-2010

BP algorithm = Gradient Descent Method

- Training examples $\{x_0^i\}_{i=1}^n$ and labels $\{y^i\}_{i=1}^n$
- Output of the network $\{x_L^i\}_{i=1}^m$
- Objective Square loss, cross-entropy loss, etc.

$$J(\{W_l\},\{b_l\}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\|y^i - x_L^i\|_2^2 \quad (1)$$

- Gradient descent

$$W_l = W_l - \eta\frac{\partial J}{\partial W_l}$$

$$b_l = b_l - \eta\frac{\partial J}{\partial b_l}$$

In practice: use Stochastic Gradient Descent (SGD)

Derivation of BP: Lagrangian Multiplier

LeCun et al. 1988

Given n training examples $(I_i, y_i) \equiv$ (input,target) and L layers

- Constrained optimization

$$\begin{aligned} \min_{W,x} \quad & \textstyle\sum_{i=1}^{n} \|x_i(L) - y_i\|_2 \\ \text{subject to} \quad & x_i(\ell) = f_\ell\Big[W_\ell x_i\,(\ell-1)\Big], \\ & i = 1, \ldots, n, \quad \ell = 1, \ldots, L,\ x_i(0) = I_i \end{aligned}$$

- Lagrangian formulation (Unconstrained)

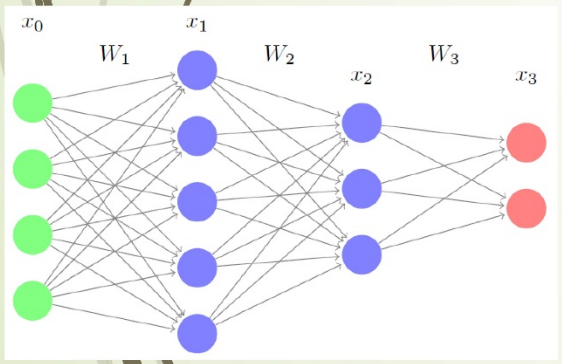
$$\min_{W,x,B} \mathcal{L}(W, x, B)$$

$$\mathcal{L}(W, x, B) = \textstyle\sum_{i=1}^{n} \left\{ \|x_i(L) - y_i\|_2^2 + \right.$$

$$\left. \textstyle\sum_{\ell=1}^{L} B_i(\ell)^T \left(x_i(\ell) - f_\ell\Big[W_\ell x_i\,(\ell-1)\Big] \right) \right\}$$

http://yann.lecun.com/exdb/publis/pdf/lecun-88.pdf

BP Algorithm: Forward Pass



- Cascade of repeated [linear operation followed by coordinatewise nonlinearity]'s
- Nonlinearities: sigmoid, hyperbolic tangent, (recently) ReLU.

Algorithm 1 Forward pass

Input: x_0

Output: x_L

1: **for** $\ell = 1$ to L **do**
2: $\quad x_\ell = f_\ell(W_\ell x_{\ell-1} + b_\ell)$
3: **end for**

back-propagation – derivation

- $\frac{\partial \mathcal{L}}{\partial B}$

Forward pass

$$x_i(\ell) = f_\ell\Big[\underbrace{W_\ell x_i\,(\ell-1)}_{A_i(\ell)}\Big] \quad \ell = 1,\ldots,L, \quad i = 1,\ldots,n$$

- $\frac{\partial \mathcal{L}}{\partial x}, z_\ell = [\nabla f_\ell] B(\ell)$

Backward (adjoint) pass

$$z(L) = 2\nabla f_L\Big[A_i(L)\Big]\,(y_i - x_i(L))$$

$$z_i(\ell) = \nabla f_\ell\Big[A_i(\ell)\Big] W_{\ell+1}^T z_i(\ell+1) \quad \ell = 0,\ldots,L-1$$

- $W \leftarrow W + \lambda \frac{\partial \mathcal{L}}{\partial W}$

Weight update

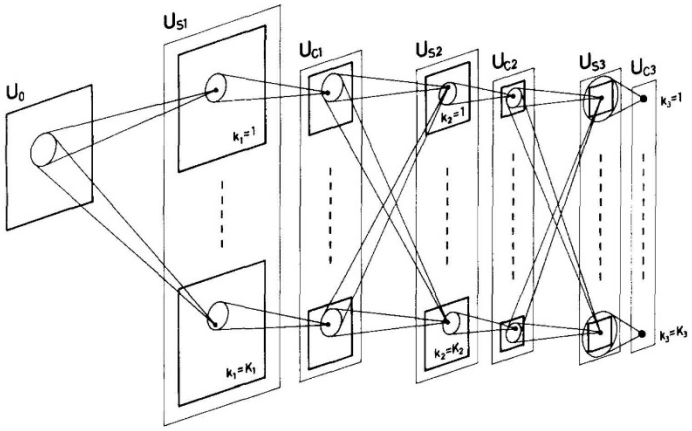
$$W_\ell \leftarrow W_\ell + \lambda \sum_{i=1}^n z_i(\ell) x_i^T(\ell-1)$$

Convolutional Neural Networks: shift invariances and locality

Biol. Cybernetics 36, 193–202 (1980)

Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position

Kunihiko Fukushima
NHK Broadcasting Science Research Laboratories, Kinuta, Setagaya, Tokyo, Japan



- Can be traced to *Neocognitron* of Kunihiko Fukushima (1979)
- Yann LeCun combined convolutional neural networks with back propagation (1989)
- Imposes **shift invariance** and **locality** on the weights
- Forward pass remains similar
- Backpropagation slightly changes – need to sum over the gradients from all spatial positions