1

Introduction

"No computer has ever been designed that is ever aware of what it's doing; but most of the time, we aren't either."
— Marvin Minsky, 1927-2016

Learning is the process of transforming *information* and *experience* into *knowledge* and *understanding*. Knowledge and understanding are measured by the ability to perform certain tasks independently. **Machine Learning** is therefore the study of algorithms and models for computer systems to carry out certain tasks independently, based on the results of a learning process. Learning tasks can range from solving simple classification problems, such as handwritten digit recognition, to more complex tasks, such as medical diagnosis or driving a car.

Machine learning is part of the broader field of **Artificial Intelligence**, but distinguishes itself from more traditional approaches to problem solving, in which machines follow a strict set of rules they are provided with. As such, it is most useful for tasks such as pattern recognition, that may be simple for humans but where precise rules are hard to come by with, or for tasks that allow for simple rules, but where the complexity of the problem makes any rule-based approach computationally infeasible. An illustrative example of the latter is the success of DeepMind's AlphaGo [1], a computer program based on reinforcement learning, at achieving super-human performance at the board game Go (围棋). Even though the rules of the game are simple, the task of beating the best human players seemed impossible only a decade ago due to the daunting complexity that ensues from the number of possible positions.

A General Framework for Learning

Machine learning lies at the intersection of approximation theory, probability theory, statistics, and optimization theory. We illustrate the interplay of these fields with a few examples.

[1] https://deepmind.com/research/case-studies/alphago-the-story-so-far

The goal of machine learning is to come up with (**learn**) a function

$$h\colon \mathcal{X} \to \mathcal{Y},$$

where $\mathcal{X}$ is a space of **inputs** or **features**, and $\mathcal{Y}$ consists of **outputs** or **responses**. The input space $\mathcal{X}$ is modelled as a metric space (such as $\mathbb{R}^d$), and the inputs could represent images, texts, emails, gene sequences, networks, financial time series, or demographic data. The output could consist of **quantitative** values, such as a temperature or the amount of a certain substance in the body, or of **qualitative** or **categorical** values, such as $\{\text{YES}, \text{NO}\}$ or $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. The first type of problem is usually called **regression**, while the latter is called **classification**. The function h is sometimes called a **hypothesis**, a **predictor**, or a **classifier**. A classifier h that takes only two values (typically 0 and 1, or -1 and 1) is called a **binary classifier**. In a machine learning scenario, a function h is chosen from a predetermined set of functions $\mathcal{H}$, called the **hypothesis space**.

Machine learning problems can be subdivided into supervised and unsupervised learning problems. In **supervised learning**, we have at our disposal a collection of input-output data pairs

$$\{\boldsymbol{x}^i, \boldsymbol{y}^i\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y},$$

and the goal is to learn a function $h\colon \mathcal{X} \to \mathcal{Y}$ from this data. The collection of pairs $\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^n$ is called the **training set**. In **unsupervised learning**, one does not have access to a training set. The prototypical example of an unsupervised learning task is **clustering**, where the tasks is to subdivide a given data set into groups based on similarities. This course will deal mainly with supervised learning.

Example 1.1 (Digit recognition)**.** Given a dataset of pixel matrices, each representing a grey-scale image, with associated **labels** telling us for each image the number it represents, the task is to use this data to train a computer program to recognise new numbers (see Figure 1.1). Such classification tasks are often carried out using **deep neural networks**, which constitute a powerful class of non-linear functions.

Example 1.2. (Clustering) In clustering applications, one observes data $\{\boldsymbol{x}^i\}_{i=1}^n$, and the goal is to subdivide the data into a number of distinct groups based on similarity, where similarity is measured using a distance function. Figure 1.2 shows an example of an artificial clustering problem and a possible solution. The notion of distance used depends on the application. For example, for binary sequences or DNA sequences one can use the *Hamming metric*, which simply counts the positions at which two sequences differ. Clustering is used extensively in genetics, biology and medicine. For example, clustering can be used to identify groups of genes (segments of DNA with a function) that encode proteins which are part of a common biological pathway. Other uses of clustering are market segmentation and community detection in networks.

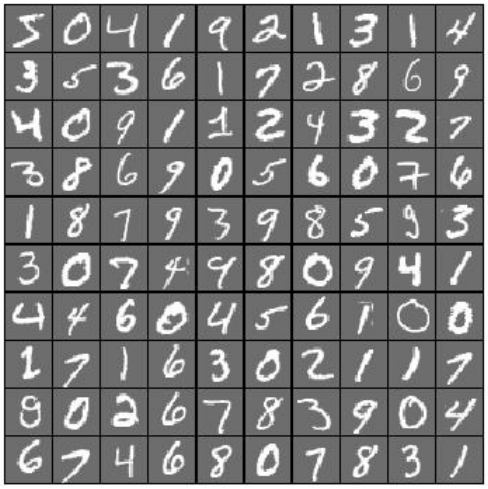

Figure 1.1: The MNIST (Modified National Institute of Standards and Technology, http://yann.lecun.com/exdb/mnist/) dataset is a large collection of images of hand-written digits, and is a frequently used benchmark in machine learning.

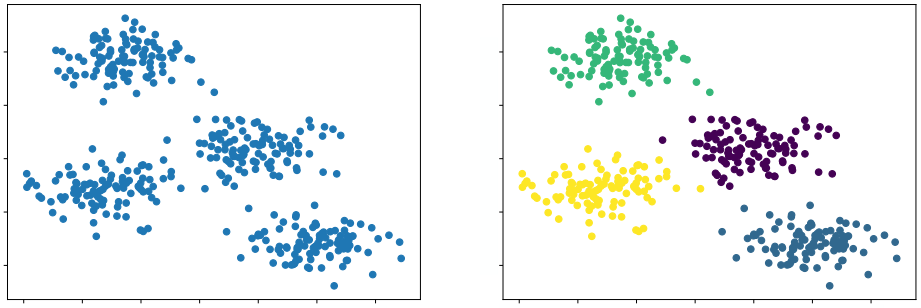

Figure 1.2: A collection of random points on the plane. The image on the right shows the four clusters as determined by the k-means algorithm.

Approximation Theory

One often makes the simplified assumption that the observed training data comes from an unknown function $f\colon \mathcal{X} \to \mathcal{Y}$. The goal is to *approximate* the function f with a function h from a hypothesis class $\mathcal{H}$, based only on the knowledge a finite set of samples $\{\boldsymbol{x}^i, \boldsymbol{y}^i\}_{i=1}^n$, where we assume $\boldsymbol{y}^i \approx f(\boldsymbol{x}^i)$ for $i \in [n] := \{1, \ldots, n\}$. Which class of functions is adequate depends on the application at hand, as well as on computational and statistical considerations. In many cases a linear function will do well, while in other situations polynomials or more complex functions, like neural

networks, are better suited.

Example 1.3. (Linear regression) Linear Regression is the problem of finding a relationship of the form

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,$$

where the $X_1, \ldots, X_p$ are **covariates** that describe certain characteristics of a system and Y is the **response**. Given a set of input-output pairs $(\boldsymbol{x}^i, y^i)$, arranged in a matrix $\boldsymbol{X}$ and a vector $\boldsymbol{y}$, we can *guess* the correct $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^\top$ by solving the *least-squares* optimization problem

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \, \|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|_2^2.$$

Figure 1.3 shows an example of linear regression.

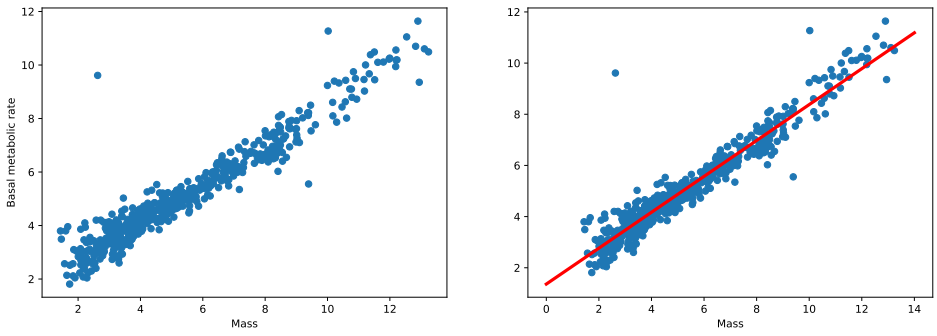

Figure 1.3: The relationship of mass to the logarithm of the basal metabolic rate in mammals. The data consists of 573 samples taken from the PanTHERA database, and the example featured in the episode Size Matters of the BBC series Wonders of Life. The right images shows the regression line determined using linear least squares.

Example 1.4. (Text classification) In text classification, the task is to decide to which of a given set of categories a given text belongs. The training data consists of a *bag of words*: this is a large sparse matrix, whose columns represent words and the rows represent articles, with the (i, j)-th entry containing the number of times word j is contained in text i.

	Goal	Soup
Article 1	5	0
Article 2	1	7

For example, in the above set we would classify the first article as "Sports" and the second one as "Food". One such training dataset is the Reuters Corpus Volume I

(RCV1) [2], an archive of over 800,000 categorised newswire stories. A typical binary classifier for such a problem would be a **linear classifier** of the form

$$h(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b,$$

with $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. Given a text, represented as a row of the dataset $\boldsymbol{x}$, it is classified into one of two classes $\{+1, -1\}$, depending on whether $h(\boldsymbol{x}) > 0$ or $h(\boldsymbol{x}) < 0$.

Example 1.5. (Deep Neural Networks) Artificial neural networks[3] are functions of the form

$$f_\ell \circ f_{\ell-1} \circ \cdots \circ f_1,$$

where each f_k is the component-wise composition of an **activation function** σ with a linear map $\mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}, \boldsymbol{x} \mapsto \boldsymbol{W}^k \boldsymbol{x} + \boldsymbol{b}^k$:

$$f_k(\boldsymbol{x}) = \sigma(\boldsymbol{W}^k \boldsymbol{x} + \boldsymbol{b}^k).$$

An activation function could be the **sigmoid** $\sigma(x) = 1/(1 + \mathrm{e}^{-x})$, which takes values between $(0, 1)$, and which can be interpreted as "selecting" certain coordinates of a vector depending on whether they are positive or negative. The coefficients w_{ij}^k of the matrix $\boldsymbol{W}^k$ in each layer are the **weights**, while the entries of $\boldsymbol{b}^k$ are called the **bias** terms. The weights and bias terms are to be adapted in order to fit observed input-output pairs. A neural network is usually represented as a graph, see Figure 1.4. Neural networks have been extremely successful in pattern recognition, and are widely used in applications ranging from natural language processing to machine translation and medical diagnostics.

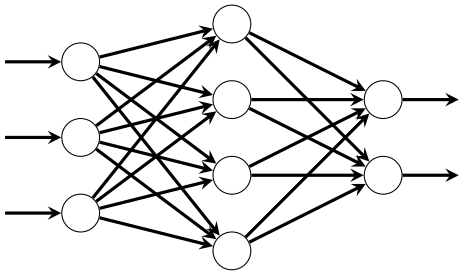

Figure 1.4: A neural network. Each layer correspond to applying a linear map to the outputs of the previous layer, followed by an activation function. Each arrow represents a *weight*. For example, the transition from the first layer to the second is a map $\mathbb{R}^3 \to \mathbb{R}^4$, and the weight associated with the arrow from the second node in layer 1 to the first node in layer 2 is the $(1, 2)$-entry in the matrix defining the corresponding linear map.

[2] Reuters Corpus Volume I (RCV1)

[3] In what follows, we will drop the 'artificial'. An alternative designation is multilayer perceptron.

One of the earliest theoretical results in approximation theory is a theorem by Weierstrass that shows that we can approximate any continuous function on an interval to arbitrary precision by polynomials.

Theorem 1.6 (Weierstrass)**.** *Let f be a continuous function on $[a, b]$. Then for any $\epsilon > 0$ there exists a polynomial $p(x)$ such that*

$$\|f - p\|_\infty = \max_{x\in[a,b]} |f(x) - p(x)| \leq \epsilon.$$

This theorem is remarkable because it shows that we can approximate any continuous function on a compact interval using only a *finite* number of parameters, the coefficients of a polynomial. The problem with this theorem is that it gives no bound on the size of the polynomial, which can be rather large. It also does not give a procedure of actually computing such an approximation, let alone finding one efficiently. We will see that neural networks have the same approximation properties, i.e., for every continuous function on an interval can be approximated to arbitrary accuracy by a neural network. There are many variations on such results for approximating a class of functions through a simpler class, and we will be interested in cases where such approximations can be efficiently computed. One way of finding good approximating functions is by using optimization methods.

Optimization

The notion of *best fit* is formalized by using a **loss function**. A loss function $L\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ measures the mismatch between a prediction on a given input $\boldsymbol{x} \in \mathcal{X}$ and an element $\boldsymbol{y} \in \mathcal{Y}$. The **empirical risk** of a function $h\colon \mathcal{X} \to \mathcal{Y}$ is the average loss $L(h(\boldsymbol{x}^i), \boldsymbol{y}^i)$ over the training data,

$$\hat{R}(h) := \frac{1}{n} \sum_{i=1}^{n} L(h(\boldsymbol{x}^i), \boldsymbol{y}^i)$$

One would then aim to find a function h among a set $\mathcal{H}$ of candidates that minimizes the loss when applying the function to the training data:

$$\operatorname*{minimize}_{h\in\mathcal{H}} \hat{R}(h). \tag{1.1}$$

Problem (1.1) is an **optimization problem**. Minimizing over a set of functions my look abstract, but functions in $\mathcal{H}$ are typically parametrized by few parameters. For example, when the class $\mathcal{H}$ consists of linear functions of the form $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$, as in Example 1.3, then the optimization problem (1.1) amounts to minimizing a function over $\mathbb{R}^{p+1}$. In the case of neural networks, Example 1.5, one optimizes over the weights w_{ij}^k and bias terms b_i^k.

The form of the loss function depends on the problem at hand and is usually derived from statistical considerations. Two common candidates are the **square error**

for regression problems, which applied to a function $h\colon \mathbb{R}^d \to \mathbb{R}$ takes the form

$$L(h(\boldsymbol{x}), y) = (h(\boldsymbol{x}) - y)^2,$$

and the **indicator loss function**, which takes the general form

$$L(h(\boldsymbol{x}), y) = \mathbf{1}\{h(\boldsymbol{x}) \neq y\} = \begin{cases} 1 & \text{if } h(\boldsymbol{x}) = y \\ 0 & \text{if } h(\boldsymbol{x}) \neq y \end{cases}.$$

As this function is not continuous, in practice one often encounters continuous approximations. A binary classifier is often implemented by a function $h\colon \mathcal{X} \to [0, 1]$ that provides a *probability* of an input belonging to a class. If $h(\boldsymbol{x}) > 1/2$, then $\boldsymbol{x}$ is assigned to class 1, while if $h(\boldsymbol{x}) \leq 1/2$, then $\boldsymbol{x}$ is assigned to class 0. A common loss function for this setting is the **log-loss** function, or **cross-entropy**,

$$\begin{aligned} L(h(\boldsymbol{x}), y) &= -y \log(h(\boldsymbol{x})) - (1 - y) \log(1 - h(\boldsymbol{x})) \\ &= \begin{cases} -\log(h(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h(\boldsymbol{x})) & \text{if } y = 0 \end{cases}. \end{aligned} \tag{1.2}$$

The function is designed to take on large values if the class predicted by $h(\boldsymbol{x})$ does not match y, and can be interpreted in the context of maximum-likelihood estimation.

Finding or approximating a minimizer of a function falls into the realm of **numerical optimization**. While for linear regression we can solve the relevant optimization problem (least-squares minimization) in closed form, for more involved problems such as neural networks we use optimization algorithms such as **gradient descent**: we start with an initial guess and try to minimize our function by taking steps in direction of *steepest descent*, that is, along the negative gradient of the function. In the case of composite functions such as neural networks, computing the gradient requires the chain rule, which leads to the famous **backpropagation** algorithm for training a neural network that will be discussed in detail.

There are many challenges related to optimization models and algorithms. The function to be minimized may have many *local* minima or saddle points, and algorithms that look for minimizers may find any one of these, instead of a global minimizer. The functions to be minimized may not be differentiable, and methods from the field of **non-smooth optimization** come into play. One of the biggest challenges for optimization algorithms in the context of machine learning, however, lies in the particular form of the objective function: it is given as a sum of many terms, one for each data point. Evaluating such a function and computing its gradient can be time and memory consuming. An old class of algorithms that includes **stochastic gradient descent** circumvents this issue by not computing the gradient of the whole function at each step, but only of a small random subset of the terms. These algorithms work surprisingly well, considering that they do not make use of all the information available at every step.

Statistics

Suppose we have a binary classification task at hand, with $\mathcal{Y} = \{0, 1\}$. We could *learn* the following function from our training data $\{\boldsymbol{x}^i, y^i\}_{i=1}^n$:

$$h(\boldsymbol{x}) = \begin{cases} y^i & \text{if } \boldsymbol{x} = \boldsymbol{x}^i, \\ 1 & \text{otherwise.} \end{cases}$$

This is called **learning by memorization**, since the function simply memorizes the value y^i for every seen example $\boldsymbol{x}^i$. The empirical risk with respect to the unit loss function for this problem is

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{h(\boldsymbol{x}^i) \neq y^i\} = 0.$$

Nevertheless, this is not a good classifier: it will not perform very well outside of the training set. This is an example of **overfitting**: when the function is adapted too closely to the seen data, it does not generalize to unseen data. The problem of **generalization** is the problem of finding a classifier that works well on unseen data.

To make the notion of generalization more precise, we assume that the training data points $(\boldsymbol{x}^i, \boldsymbol{y}^i)$ are realizations of a pair of random variables (X, Y), sampled from an (unknown) probability distribution on the product $\mathcal{X} \times \mathcal{Y}$. The variables X and Y are in general not independent (otherwise there would be nothing to learn), but are related by a relationship of the form $Y = f(X) + \epsilon$, where ϵ is a random perturbation with expected value $\mathbb{E}[\epsilon] = \mathbf{0}$. One could interpret the presence of the random noise ϵ as indicative of uncertainty or missing information. For example, when trying to predict a certain disease based on genetic markers, the genetic data might simply not carry enough information to always make a correct prediction. The function f is called the **regression function**. It is the conditional expectation of Y given a value of X,

$$f(\boldsymbol{x}) = \mathbb{E}[Y \mid X = \boldsymbol{x}].$$

Given a classifier $h \in \mathcal{H}$ and a loss function L, the **generalization risk** is the expected value

$$R(h) := \mathbb{E}[L(h(X), Y)].$$

If $L(h(\boldsymbol{x}), \boldsymbol{y}) = \mathbf{1}\{h(\boldsymbol{x}) \neq \boldsymbol{y}\}$ is the unit loss, then this is simply $\mathbb{P}\{h(X) \neq Y\}$, i.e., the probability of misclassifying a randomly chosen input-output pair. The training data can be modelled as sampling form n pairs of random variables

$$(X_1, Y_1), \ldots, (X_n, Y_n)$$

that are identically distributed and independent copies of (X, Y). Given a classifier h, the empirical risk becomes a random variable

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(X_i), Y_i).$$

The expected value of this random variable is

$$\mathbb{E}[\hat{R}(h)] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[L(h(X_i), Y_i)] = \mathbb{E}[L(h(X), Y)] = R(h),$$

where we used the linearity of expectation and the fact that the (X_i, Y_i) are independent and identically distributed (i.i.d). The empirical risk $\hat{R}(h)$ is thus an **unbiased estimator** of the generalization risk $R(h)$.

Example 1.7. The loss function is often chosen so that the problem of empirical risk minimization becomes a **maximum likelihood** estimation problem. Consider the example where Y takes values in $\{0, 1\}$ with probability $\mathbb{P}\{Y = 1 \mid X = \boldsymbol{x}\} = f(\boldsymbol{x})$. Conditioned on $X = \boldsymbol{x}$, Y is a Bernoulli random variable with parameter $p = f(\boldsymbol{x})$, and the log-loss function (1.2) is precisely the negative **log-likelihood** function for the problem of estimating the parameter p.

When looking for a good hypothesis h, all we have at our disposal is the empirical risk function constructed from the training data. It turns out that the quality of an empirical risk minimizer $\hat{h}$ from a hypothesis class $\mathcal{H}$ can be measured by the **estimation error**, which compares the generalization risk of $\hat{h}$ to the smallest possible generalization risk in $\mathcal{H}$, and the **approximation error**, which measures how small the generalization risk can become within $\mathcal{H}$. There is usually a trade-off between these to errors: if the class of functions $\mathcal{H}$ is large, then it is likely to contain functions with small generalization risk and thus have small approximation error, but the empirical risk minimizer $\hat{h}$ is likely to "overfit" the data and not generalize well. On the other hand, if $\mathcal{H}$ is small (in the extreme case, consisting of only one function), then the empirical risk minimizer is likely to be close to the best possible hypothesis in $\mathcal{H}$, but the approximation error will be large. Figure 1.5 shows an example in which data from a function with noise is approximated by polynomials of different degrees.

The field of **Statistical Learning Theory** aims to understand the relation between the generalization risk, the empirical risk, the capacity of a hypothesis class $\mathcal{H}$, and the number of samples n. In particular, notions such as the capacity of a hypothesis class are given a precise meaning through concepts such as VC dimension, Rademacher complexity, and covering numbers.

Notes

The ideas from approximation theory, optimization and statistics that underlie modern machine learning are old. Linear regression was known to Legendre and Gauss. The Weierstrass Approximation Theorem was published by Weierstrass in [8], see [6, Chapter 6] for an account and more history on approximation theory. Neural networks go back to the seminal work by McCulloch and Pitts from 1943 [2], followed by Rosenblatt's Perceptron [4]. The term "Machine Learning" was first coined by Arthur Samuel in 1959 [5]; at the time, "Cybernetics" was still widely used. Gradient descent

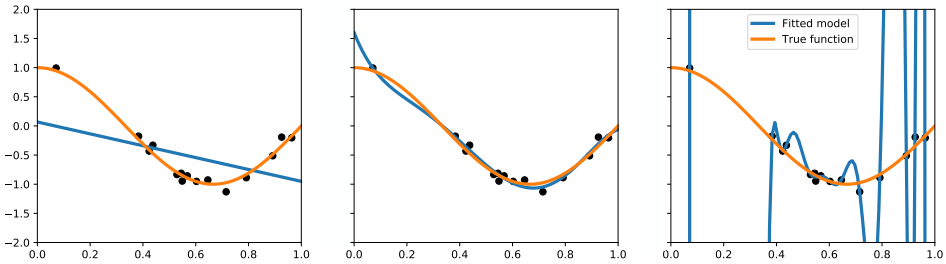

Figure 1.5: The data consists of 15 samples from the graph of a cosine function with added noise. The three displays show an approximation with a linear function, with a polynomial of degree 5, and with a polynomial of degree 15. The linear function has a large error on both the training set and in relation to the true function. The polynomial of degree 15, on the other hand, has zero error on the training data (a polynomial of degree d can fit $d + 1$ points with distinct x-values exactly), but it will likely perform poorly on new data. This is an example of **overfitting**: more parameters in the model will not necessarily lead to a better performance.

was known to Cauchy, and the most important algorithm for deep learning today, Stochastic Gradient Descent, was introduced by Robbins and Monro in 1951 [3]. The field of Statistical Learning Theory arose in the 1960s through seminal work by Vapnik and Chervonenkis, see [7] for an overview. For an account of mathematical learning theory, see [1].

Research in machine learning exploded in the 1990s, with striking new results and applications appearing at breathtaking pace. Still, apart from some of the more theoretical developments in learning theory and high-dimensional probability, these breakthroughs rarely relied on mathematics that was not available 50 years ago. So what has changed since the early days of cybernetics? The main reason for the sudden surge in popularity is the availability of vast amounts of data, and equally important, the computational resources to process the data. New applications have in turn led to new mathematical problems, and to new connections between various fields.

[1] Felipe Cucker and Ding Xuan Zhou. *Learning theory: an approximation theory viewpoint*, volume 24. Cambridge University Press, 2007.

[2] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[3] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[4] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[5] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, pages 71–105, 1959.

[6] Lloyd N Trefethen. *Approximation theory and approximation practice*, volume 128. Siam, 2013.

[7] Vladimir Vapnik. *The nature of statistical learning theory*. Springer, 2013.

[8] Karl Weierstrass. Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, 2:633–639, 1885.