Künstliche Neuronale Netze

Einführung

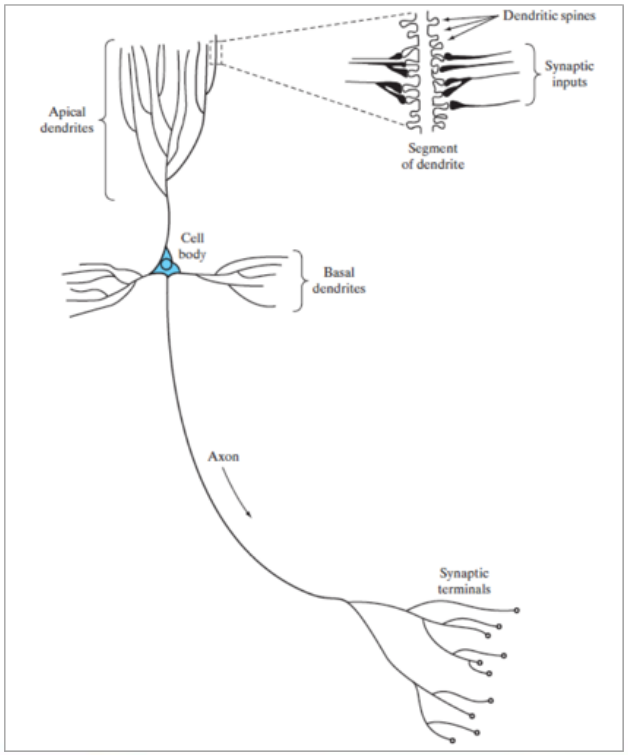
Zoran Nikolić

Agenda

1. Motivation
2. Neuronale Netze
3. Trainieren von neuronalen Netzen
4. Gradientenabstiegsverfahren
5. Ausblick

1. MOTIVATION

Motivation aus der Gehirnforschung

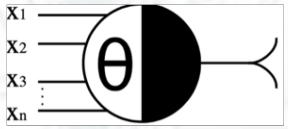


Haykin: Neural Networks and Learning Machines, Pearson, 2009

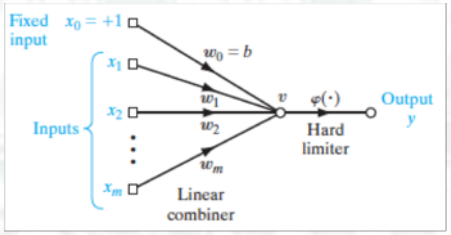
- Ramón y Cajál (1911): Die Idee von Neuronen
- Gehirn als extrem effizienter Rechner
- Nachahmung der Intelligenz

Neuronale Netze & Deep Learning

- Erstes KNN: McCulloch-Pitts, 1943
- Multi-Layer Perceptron: Rosenblatt, 1958, 1962



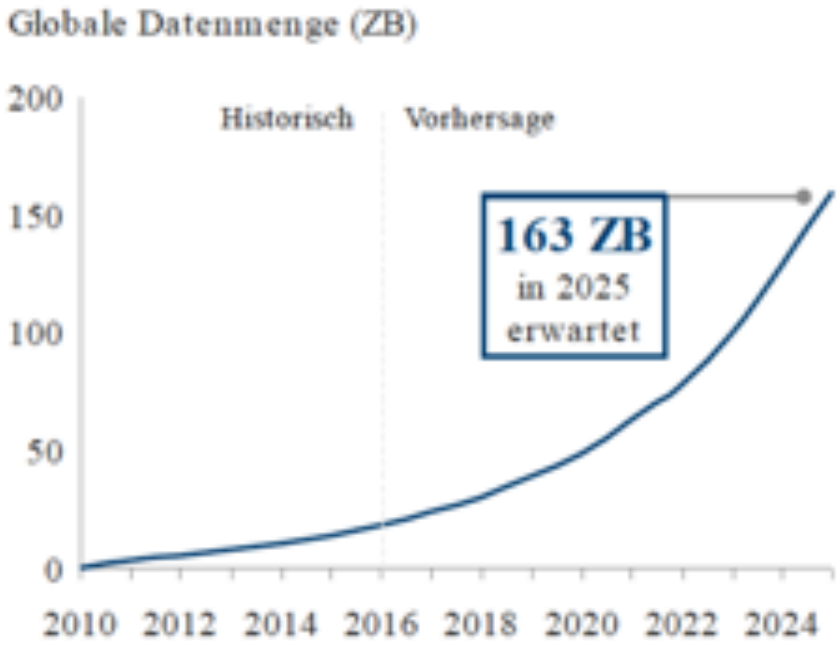
Wikipedia

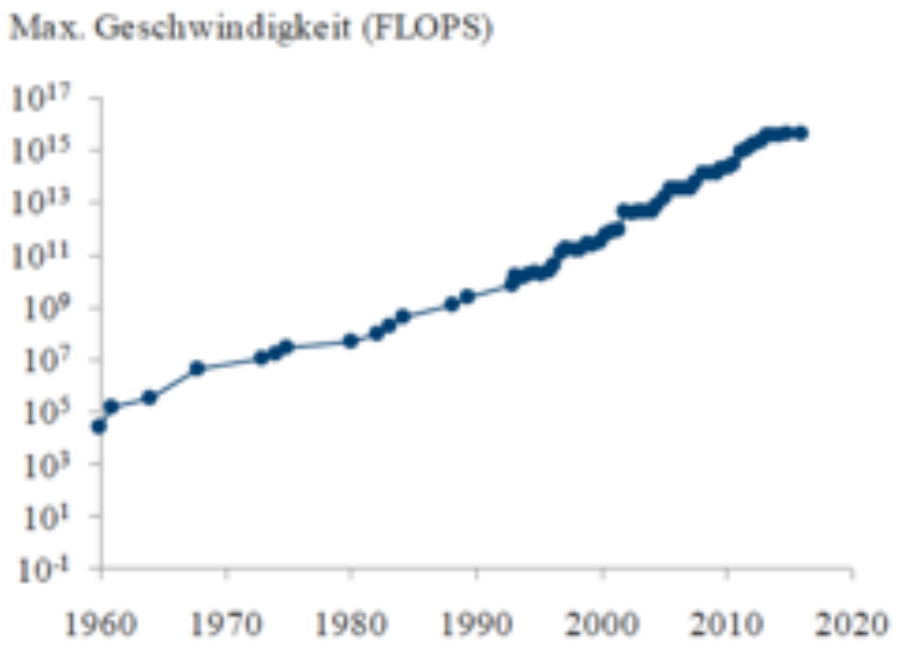


Haykin:

- Idee: Viele Berechnungseinheiten, die erst durch ihre Interaktion „intelligent“ werden
- Boom seit ca. 10 Jahren durch den enormen Anstieg an Rechenkapazitäten

Massive Zunahme von Rechenkapazitäten





ZB = Zettabyte, FLOPS = floating point operations per second.
Quelle linke Grafik: IDC, 2017, Data Age 2025. Online verfügbar: https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf, abgerufen am 19.02.2018.
Quelle rechte Grafik: Denning et al., 2017, Exponential Laws of Computing Growth. Online verfügbar: http://www0.cs.ucl.ac.uk/staff/ucacbbl/gggp/langdon_sse_8-feb-2017.pdf, abgerufen am 19.02.2018.

Anwendungsgebiete

- Klassifizierung
- Erkennung von Anomalien/ Unregelmäßigkeiten
- Transkription
- Regression
- Maschinelle Übersetzung
- Sampling

Sampling: Texterzeugung

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciously in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

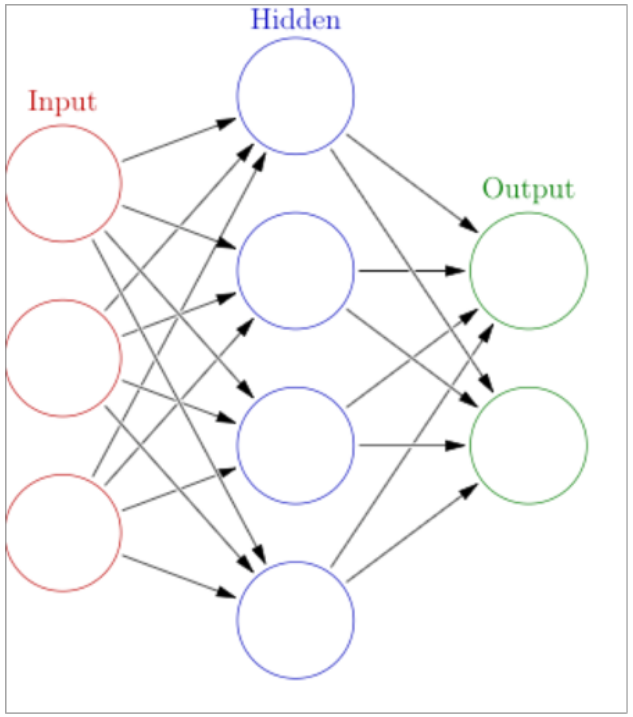
Oben: Fake Shakespeare

Links: Fake algebraische Geometrie

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

2. NEURONALE NETZE

Input-Versteckte Schichten-Output



Wikipedia

- Je eine Input- und Output-Schicht
- Keine, eine oder mehrere Verarbeitungsschichten (= versteckte Schichten)
- Nicht immer existieren alle Verbindungen zwischen zwei Schichten
- In den Neuronen Verarbeitung mit Aktivierungsfunktionen

Aktivierungsfunktionen

- **Tangens Hyperbolicus**
- **Logistische** $f(x, \theta) = \frac{1}{1+e^{-(x-\theta)}}$
- **Rectified Linear Unit (ReLU)** $f(x, \theta) = \max(0, x - \theta)$
- **Leaky ReLU** $f(x, \theta) = \max(\alpha(x - \theta), x - \theta), \alpha > 0$ **klein**

Mathematische Darstellung

- Schwierig, da viele verschachtelte Funktionen
- In der Regel grafisch, gelegentlich hilft die Matrixform
- Im Code meistens einfach
- Notation entscheidend

3. TRAINIEREN VON NEURONALEN NETZEN

Überwachtes und unüberwachtes Lernen

- Traditionell (in Analogie mit dem menschlichen Gehirn) Unterscheidung zwischen überwachtem und unüberwachtem Lernen
- Außerdem diverse Abstufungen wie verstärktes Lernen oder unvollständiges überwachtes Lernen

Überwachtes Lernen

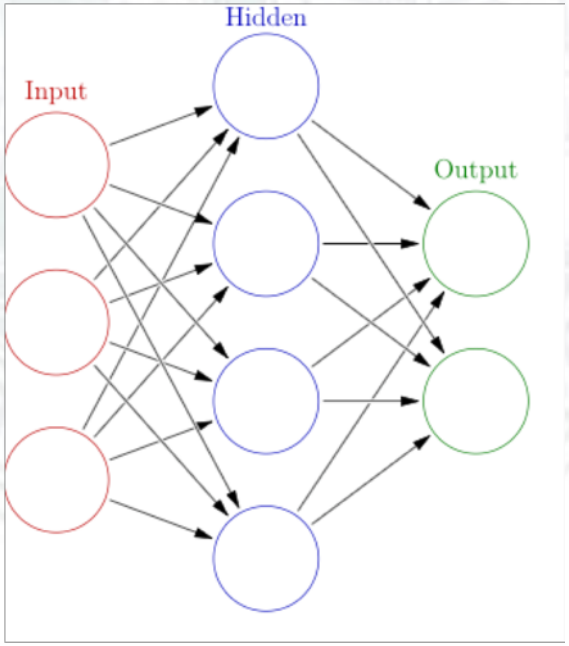
- Zielmerkmal (Label) ist anhand von erklärenden Merkmalen zu prognostizieren
  - Verallgemeinerung des Wissens auf neue Datenmuster erwünscht
  - Datenpunkte werden auch Beispiele genannt

Unüberwachtes Lernen

- Intrinsische Muster sind aus den Daten zu extrahieren
  - I. A. schwieriger als überwachtes Lernen
  - Anomalieerkennung, Clustering

Trainieren von neuronalen Netzen

- Kernidee: Die Gewichte zwischen den Neuronen werden verändert



Backpropagation of Error

- Wahl der Lernraten
- Schichtweise Veränderung der Lernraten (z. B. Multiplikation mit 0,1)
- Varianten mit und ohne Momentum (z. B. 0,75 der letzten Änderung)
- Resilient BoE (nur Vorzeichen relevant)

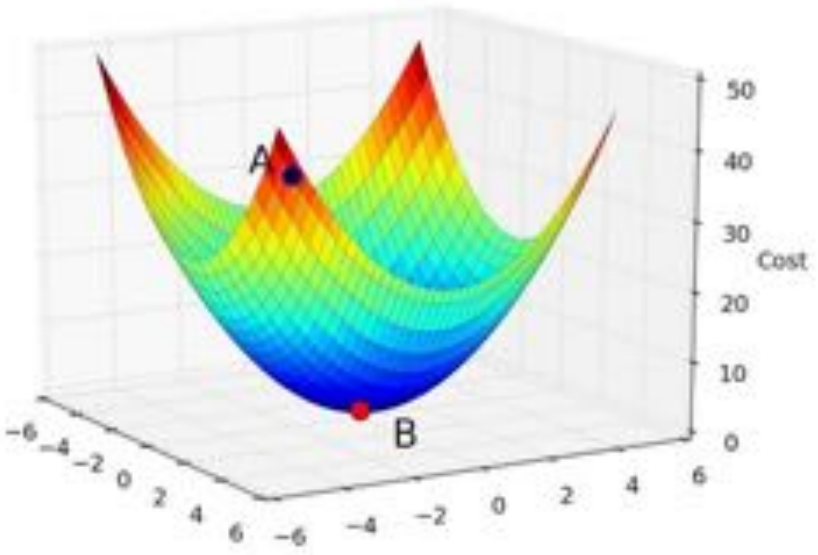
4. GRADIENTENABSTIEGS-VERFAHREN

Kostenfunktion

- Kostenfunktion = die zu minimierende Funktion:
  - Quadratensumme (Maximum Likelihood)
  - Euklidischer Abstand (rechenintensiver)
  - Mittlerer quadratischer Fehler

Gradient

- Richtung des steilsten An- bzw. Abstieges
- $\nabla E = \left(\frac{\partial E}{\partial w_{l,k}}\right)_{l,k}$ für Gewichte $w_{l,k}$



https://medium.com/abdullah-al-imran/intuition-of-gradient-descent-for-machine-learning-49e1b6b89c8b

Beispiel: Nur Input- und Output-Schicht

- **Input-Neuronen:** $i_1, \ldots, i_r \in \mathbb{R}$
- **Output-Neuronen:** $o_1, \ldots, o_n : \mathbb{R}^{2r} \to \mathbb{R}$

  $o_k : (i_1, w_{1,k}, \ldots, i_r, w_{r,k}) \mapsto$

$$f^k\left(\sum_{l=1}^{r} i_l w_{l,k}\right) = \sum_{l=1}^{r} i_l w_{l,k}$$

- **Gewichte:** $w_{l,k} \in \mathbb{R}, 1 \leq l \leq r, 1 \leq k \leq n$

Beispiel: Matrixform

- Input: $\mathbf{I} = [i_1, \ldots, i_r]^T$
- Gewichte: $\Psi = \begin{bmatrix} w_{1,1} & \cdots & w_{r,1} \\ \vdots & \ddots & \vdots \\ w_{1,n} & \cdots & w_{r,n} \end{bmatrix}$
- Output: $\mathbf{O} = f(\Psi \cdot \mathbf{I}) = \Psi \cdot \mathbf{I}$

Beispiel: Kostenfunktion

- Samples (= Beispiele = die wahren Werte der Funktion): $v_1, \ldots, v_n \in \mathbb{R}$
- Fehler pro Output: $\varepsilon_k = (o_k - v_k)^2$
- Kostenfunktion:

$$E\colon (i_1, w_{1,1}, f^1 \ldots, i_r, w_{r,n}, f^n) \longmapsto \frac{1}{2}\sum_{k=1}^{n} \varepsilon_k$$

Beispiel: Backpropagation

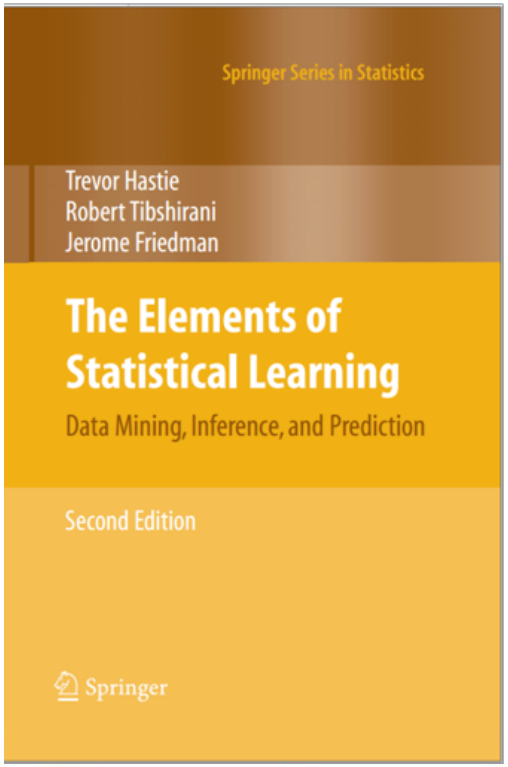
- Lege Lernrate η fest, z. B. 0,2
- Algorithmus:
  - Initialisiere $\Psi(0)$, Start Schleife
  - Berechne O für alle Beispiele
  - Berechne den Gradienten ∇E
  - Passe Gewichte an $\Psi(t+1) = \Psi(t) - \eta \nabla E$
  - Wiederhole abh. vom Abbruchkriterium

5. AUSBLICK

Weitere Machine-Learning-Ansätze

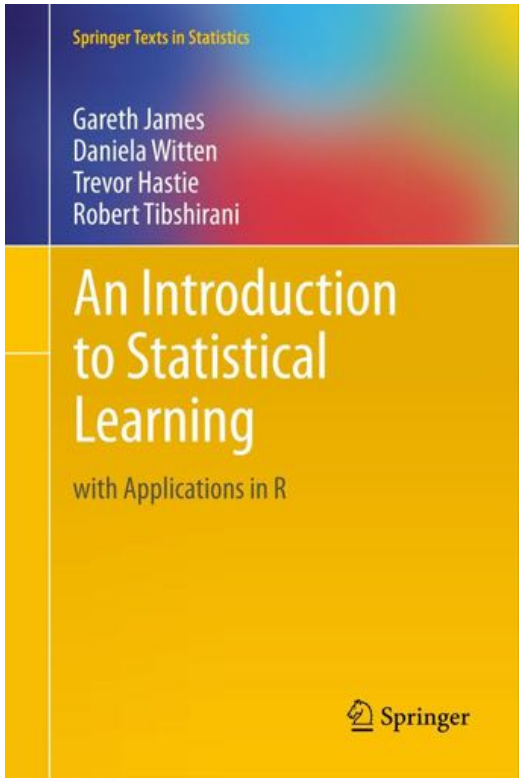
- Lineare Regression
- Polynomiale Regression
- Verallgemeinerte lineare Modelle
- Verallgemeinerte additive Modelle
- Kernel Regressionen
- Entscheidungsbäume

Literatur 1



- Lehrwerk mit anschaulichen und dennoch präzisen Erläuterungen
- Aus 2009 mit allen Grundlagen

Literatur 2



- **Praxisorientiert, einfach zugänglich**
- **Mit Code-Beispielen**
- **Aus 2013, frei verfügbar online**
- **Dazu Online-Kurs**

https://lagunita.stanford.edu/courses/HumanitiesSciences/StatLearning/Winter2016/about

Erkenntnisse aus der Praxis maschinellen Lernens

A Few Useful Things to Know about Machine Learning

Pedro Domingos
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350, U.S.A.
pedrod@cs.washington.edu

ABSTRACT
Machine learning algorithms can figure out how to perform important tasks by generalizing from examples. This is often feasible and cost-effective where manual programming is not. As more data becomes available, more ambitious problems can be tackled. As a result, machine learning is widely used in computer science and other fields. However, developing successful machine learning applications requires a substantial amount of "black art" that is hard to find in textbooks. This article summarizes twelve key lessons that machine learning researchers and practitioners have learned. These include pitfalls to avoid, important issues to focus on, and answers to common questions.

1. INTRODUCTION
Machine learning systems automatically learn programs from data. This is often a very attractive alternative to manually constructing them, and in the last decade the use of machine learning has spread rapidly throughout computer science and beyond. Machine learning is used in Web search, spam filters, recommender systems, ad placement, credit scoring, fraud detection, stock trading, drug design, and many other applications. A recent report from the McKinsey Global Institute asserts that machine learning (a.k.a. data mining or predictive analytics) will be the driver of the next big wave of innovation [16]. Several fine textbooks are available to interested practitioners and researchers (e.g., [17, 25]). However, much of the "folk knowledge" that is needed to successfully develop machine learning applications is not readily available in them. As a result, many machine learning projects take much longer than necessary or wind up producing less-than-ideal results. Yet much of this folk knowledge is fairly easy to communicate. This is the purpose of this article.

Many different types of machine learning exist, but for illustration purposes I will focus on the most mature and widely used one: classification. Nevertheless, the issues I will discuss apply across all of machine learning. A *classifier* is a system that inputs (typically) a vector of discrete and/or continuous *feature values* and outputs a single discrete value, the *class*. For example, a spam filter classifies email messages into "spam" or "not spam," and its input may be a Boolean vector $\mathbf{x} = (x_1, \ldots, x_j, \ldots, x_d)$, where $x_j = 1$ if the jth word in the dictionary appears in the email and $x_j = 0$ otherwise. A *learner* inputs a *training set* of *examples* $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})$ is an observed input and y_i is the corresponding output, and outputs a classifier. The test of the learner is whether this classifier produces the correct output y_t for future examples $\mathbf{x}_t$ (e.g., whether the spam filter correctly classifies previously unseen emails as spam or not spam).

2. LEARNING = REPRESENTATION + EVALUATION + OPTIMIZATION
Suppose you have an application that you think machine learning might be good for. The first problem facing you is the bewildering variety of learning algorithms available. Which one to use? There are literally thousands available, and hundreds more are published each year. The key to not getting lost in this huge space is to realize that it consists of combinations of just three components. The components are:

Representation. A classifier must be represented in some formal language that the computer can handle. Conversely, choosing a representation for a learner is tantamount to choosing the set of classifiers that it can possibly learn. This set is called the *hypothesis space* of the learner. If a classifier is not in the hypothesis space, it cannot be learned. A related question, which we will address in a later section, is how to represent the input, i.e., what features to use.

Evaluation. An evaluation function (also called *objective function* or *scoring function*) is needed to distinguish good classifiers from bad ones. The evaluation function used internally by the algorithm may differ from the external one that we want the classifier to optimize, for ease of optimization (see below) and due to the issues discussed in the next section.

Optimization. Finally, we need a method to search among the classifiers in the language for the highest-scoring one. The choice of optimization technique is key to the efficiency of the learner, and also helps determine the classifier produced if the evaluation function has more than one optimum. It is common for new learners to start out using off-the-shelf optimizers, which are later replaced by custom-designed ones.

Table 1 shows common examples of each of these three components. For example, k-nearest neighbor classifies a test example by finding the k most similar training examples and predicting the majority class among them. Hyperplane-based methods form a linear combination of the features per class and predict the class with the highest-valued combination. Decision trees test one feature at each internal node,

Contents:

- Learning = Representation + Evaluation + Optimization
- It's generalization that counts
- Data alone is not enough
- Overfitting has many faces
- Intuition fails in high dimensions
- Theoretical guarantees are not what they seem
- Feature engineering is the key
- More data beats a cleverer algorithm
- Learn many models, not just one
- Simplicity does not imply accuracy
- Representable does not imply learnable
- Correlation does not imply causation

https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf