Title: Perceptron
URL: https://en.wikipedia.org/wiki/Perceptron
PageID: 172777
Categories: Category:Artificial neural networks, Category:Classification algorithms

---

Supervised learning
Unsupervised learning
Semi-supervised learning
Self-supervised learning
Reinforcement learning
Meta-learning
Online learning
Batch learning
Curriculum learning
Rule-based learning
Neuro-symbolic AI
Neuromorphic engineering
Quantum machine learning
Classification
Generative modeling
Regression
Clustering
Dimensionality reduction
Density estimation
Anomaly detection
Data cleaning
AutoML
Association rules
Semantic analysis
Structured prediction
Feature engineering
Feature learning
Learning to rank
Grammar induction
Ontology learning
Multimodal learning

Apprenticeship learning

Decision trees

Ensembles Bagging Boosting Random forest

Bagging

Boosting

Random forest

k -NN

Linear regression

Naive Bayes

Artificial neural networks

Logistic regression

Perceptron

Relevance vector machine (RVM)

Support vector machine (SVM)

BIRCH

CURE

Hierarchical

k -means

Fuzzy

Expectation–maximization (EM)

DBSCAN

OPTICS

Mean shift

Factor analysis

CCA

ICA

LDA

NMF

PCA

PGD

t-SNE

SDL

Graphical models Bayes net Conditional random field Hidden Markov

Bayes net

Conditional random field

Hidden Markov

RANSAC

k -NN

Local outlier factor

Isolation forest

Autoencoder

Deep learning

Feedforward neural network

Recurrent neural network LSTM GRU ESN reservoir computing

LSTM

GRU

ESN

reservoir computing

Boltzmann machine Restricted

Restricted

GAN

Diffusion model

SOM

Convolutional neural network U-Net LeNet AlexNet DeepDream

U-Net

LeNet

AlexNet

DeepDream

Neural field Neural radiance field Physics-informed neural networks

Neural radiance field

Physics-informed neural networks

Transformer Vision

Vision

Mamba

Spiking neural network

Memtransistor

Electrochemical RAM (ECRAM)

Q-learning

Policy gradient

SARSA

Temporal difference (TD)

Multi-agent Self-play

Self-play

Active learning

Crowdsourcing

Human-in-the-loop

Mechanistic interpretability

RLHF

Coefficient of determination

Confusion matrix

Learning curve

ROC curve

Kernel machines

Bias–variance tradeoff

Computational learning theory

Empirical risk minimization

Occam learning

PAC learning

Statistical learning

VC theory

Topological deep learning

AAAI

ECML PKDD

NeurIPS

ICML

ICLR

IJCAI

ML

JMLR

Glossary of artificial intelligence

List of datasets for machine-learning research List of datasets in computer vision and image processing

List of datasets in computer vision and image processing

Outline of machine learning

v

t

e

In machine learning , the perceptron is an algorithm for supervised learning of binary classifiers . A binary classifier is a function that can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. [1] It is a type of linear classifier , i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector .

History

The artificial neuron network was invented in 1943 by Warren McCulloch and Walter Pitts in A logical calculus of the ideas immanent in nervous activity . [5]

In 1957, Frank Rosenblatt was at the Cornell Aeronautical Laboratory . He simulated the perceptron on an IBM 704 . [6] [7] Later, he obtained funding by the Information Systems Branch of the United States Office of Naval Research and the Rome Air Development Center , to build a custom-made computer, the Mark I Perceptron . It was first publicly demonstrated on 23 June 1960. [8] The machine was "part of a previously secret four-year NPIC [the US' National Photographic Interpretation Center] effort from 1963 through 1966 to develop this algorithm into a useful tool for photo-interpreters". [9]

Rosenblatt described the details of the perceptron in a 1958 paper. [10] His organization of a perceptron is constructed of three kinds of cells ("units"): AI, AII, R, which stand for " projection ", "association" and "response". He presented at the first international symposium on AI, Mechanisation of Thought Processes , which took place in 1958 November. [11]

Rosenblatt's project was funded under Contract Nonr-401(40) "Cognitive Systems Research Program", which lasted from 1959 to 1970, [12] and Contract Nonr-2381(00) "Project PARA" ("PARA" means "Perceiving and Recognition Automata"), which lasted from 1957 [6] to 1963. [13]

In 1959, the Institute for Defense Analysis awarded his group a $10,000 contract. By September 1961, the ONR awarded further $153,000 worth of contracts, with $108,000 committed for 1962. [14]

The ONR research manager, Marvin Denicoff, stated that ONR, instead of ARPA , funded the Perceptron project, because the project was unlikely to produce technological results in the near or medium term. Funding from ARPA go up to the order of millions dollars, while from ONR are on the order of 10,000 dollars. Meanwhile, the head of IPTO at ARPA, J.C.R. Licklider , was interested in 'self-organizing', 'adaptive' and other biologically-inspired methods in the 1950s; but by the mid-1960s he was openly critical of these, including the perceptron. Instead he strongly favored the logical AI approach of Simon and Newell . [15]

Mark I Perceptron machine

The perceptron was intended to be a machine, rather than a program, and while its first implementation was in software for the IBM 704 , it was subsequently implemented in custom-built hardware as the Mark I Perceptron with the project name "Project PARA", [16] designed for image recognition . The machine is currently in Smithsonian National Museum of American History . [17]

The Mark I Perceptron had three layers. One version was implemented as follows:

An array of 400 photocells arranged in a 20x20 grid, named "sensory units" (S-units), or "input retina". Each S-unit can connect to up to 40 A-units.

A hidden layer of 512 perceptrons, named "association units" (A-units).

An output layer of eight perceptrons, named "response units" (R-units).

Rosenblatt called this three-layered perceptron network the alpha-perceptron , to distinguish it from other perceptron models he experimented with. [8]

The S-units are connected to the A-units randomly (according to a table of random numbers) via a plugboard (see photo), to "eliminate any particular intentional bias in the perceptron". The connection weights are fixed, not learned. Rosenblatt was adamant about the random connections, as he believed the retina was randomly connected to the visual cortex, and he wanted his perceptron machine to resemble human visual perception. [18]

The A-units are connected to the R-units, with adjustable weights encoded in potentiometers , and weight updates during learning were performed by electric motors. [2] : 193 The hardware details are in an operators' manual. [16]

In a 1958 press conference organized by the US Navy, Rosenblatt made statements about the perceptron that caused a heated controversy among the fledgling AI community; based on Rosenblatt's statements, The New York Times reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." [19]

The Photo Division of Central Intelligence Agency , from 1960 to 1964, studied the use of Mark I Perceptron machine for recognizing militarily interesting silhouetted targets (such as planes and ships) in aerial photos . [20] [21]

Principles of Neurodynamics (1962)

Rosenblatt described his experiments with many variants of the Perceptron machine in a book Principles of Neurodynamics (1962). The book is a published version of the 1961 report. [22]

Among the variants are:

"cross-coupling" (connections between units within the same layer) with possibly closed loops,

"back-coupling" (connections from units in a later layer to units in a previous layer),

four-layer perceptrons where the last two layers have adjustible weights (and thus a proper multilayer perceptron),

incorporating time-delays to perceptron units, to allow for processing sequential data,

analyzing audio (instead of images).

The machine was shipped from Cornell to Smithsonian in 1967, under a government transfer administered by the Office of Naval Research. [9]

Perceptrons (1969)

Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognise many classes of patterns. This caused the field of neural network research to stagnate for many years, before it was recognised that a feedforward neural network with two or more layers (also called a multilayer perceptron) had greater processing power than perceptrons with one layer (also called a single-layer perceptron).

Single-layer perceptrons are only capable of learning linearly separable patterns. [23] For a classification task with some step activation function, a single node will have a single line dividing the data points forming the patterns. More nodes can create more dividing lines, but those lines must somehow be combined to form more complex classifications. A second layer of perceptrons, or even linear nodes, are sufficient to solve many otherwise non-separable problems.

In 1969, a famous book entitled Perceptrons by Marvin Minsky and Seymour Papert showed that it was impossible for these classes of network to learn an XOR function. It is often incorrectly believed that they also conjectured that a similar result would hold for a multi-layer perceptron network. However, this is not true, as both Minsky and Papert already knew that multi-layer perceptrons were capable of producing an XOR function. (See the page on Perceptrons (book) for more information.) Nevertheless, the often-miscited Minsky and Papert text caused a significant decline in interest and funding of neural network research. It took ten more years until neural network research experienced a resurgence in the 1980s. [23] [verification needed] This text was reprinted in 1987 as "Perceptrons - Expanded Edition" where some errors in the original text are shown and corrected.

Subsequent work

Rosenblatt continued working on perceptrons despite diminishing funding. The last attempt was Tobermory, built between 1961 and 1967, built for speech recognition. [24] It occupied an entire room. [25] It had 4 layers with 12,000 weights implemented by toroidal magnetic cores . By the time of its completion, simulation on digital computers had become faster than purpose-built perceptron machines. [26] He died in a boating accident in 1971.

A simulation program for neural networks was written for IBM 7090/7094 , and was used to study various pattern recognition applications, such as character recognition , particle tracks in bubble-chamber photographs; phoneme, isolated word, and continuous speech recognition ; speaker verification ; and center-of-attention mechanisms for image processing . [27] [28]

The kernel perceptron algorithm was already introduced in 1964 by Aizerman et al. [29] Margin bounds guarantees were given for the Perceptron algorithm in the general non-separable case first

by Freund and Schapire (1998), [1] and more recently by Mohri and Rostamizadeh (2013) who extend previous results and give new and more favorable L1 bounds. [30] [31]

The perceptron is a simplified model of a biological neuron . While the complexity of biological neuron models is often required to fully understand neural behavior, research suggests a perceptron-like linear model can produce some behavior seen in real neurons. [32]

The solution spaces of decision boundaries for all binary functions and learning behaviors are studied in. [33]

Definition

In the modern sense, the perceptron is an algorithm for learning a binary classifier called a threshold function : a function that maps its input x {\displaystyle \mathbf {x} } (a real-valued vector) to an output value f (x) {\displaystyle f(\mathbf {x})} (a single binary value):

f (x) = h (w ⋅ x + b) {\displaystyle f(\mathbf {x})=h(\mathbf {w} \cdot \mathbf {x} +b)}

where h {\displaystyle h} is the Heaviside step-function (where an input of > 0 {\textstyle >0} outputs 1; otherwise 0 is the output), w {\displaystyle \mathbf {w} } is a vector of real-valued weights, w ⋅ x {\displaystyle \mathbf {w} \cdot \mathbf {x} } is the dot product ∑ i = 1 m w i x i {\textstyle \sum _{i=1}^{m}w_{i}x_{i}} , where m is the number of inputs to the perceptron, and b is the bias . The bias shifts the decision boundary away from the origin and does not depend on any input value.

Equivalently, since w ⋅ x + b = (w , b) ⋅ (x , 1) {\displaystyle \mathbf {w} \cdot \mathbf {x} +b=(\mathbf {w} ,b)\cdot (\mathbf {x} ,1)} , we can add the bias term b {\displaystyle b} as another weight w m + 1 {\displaystyle \mathbf {w} _{m+1}} and add a coordinate 1 {\displaystyle 1} to each input x {\displaystyle \mathbf {x} } , and then write it as a linear classifier that passes the origin: f (x) = h (w ⋅ x) {\displaystyle f(\mathbf {x})=h(\mathbf {w} \cdot \mathbf {x})}

The binary value of f (x) {\displaystyle f(\mathbf {x})} (0 or 1) is used to perform binary classification on x {\displaystyle \mathbf {x} } as either a positive or a negative instance. Spatially, the bias shifts the position (though not the orientation) of the planar decision boundary .

In the context of neural networks, a perceptron is an artificial neuron using the Heaviside step function as the activation function. The perceptron algorithm is also termed the single-layer perceptron , to distinguish it from a multilayer perceptron , which is a misnomer for a more complicated neural network. As a linear classifier, the single-layer perceptron is the simplest feedforward neural network .

Power of representation

Information theory

From an information theory point of view, a single perceptron with K inputs has a capacity of 2K bits of information. [34] This result is due to Thomas Cover . [35]

Specifically let T (N , K) {\displaystyle T(N,K)} be the number of ways to linearly separate N points in K dimensions, then T (N , K) = { 2 N K ≥ N 2 ∑ k = 0 K − 1 (N − 1 k) K < N {\displaystyle T(N,K)=\left\{{\begin{array}{cc}2^{N}&K;\geq N\\2\sum _{k=0}^{K-1}\left({\begin{array}{c}N-1\\k\end{array}}\right)&K; 2 K {\displaystyle N>2K} . In words, one perceptron unit can almost certainly memorize a random assignment of binary labels on N points when N ≤ 2 K {\displaystyle N\leq 2K} , but almost certainly not when N > 2 K {\displaystyle N>2K} .

Boolean function

When operating on only binary inputs, a perceptron is called a linearly separable Boolean function , or threshold Boolean function. The sequence of numbers of threshold Boolean functions on n inputs is OEIS A000609 . The value is only known exactly up to n = 9 {\displaystyle n=9} case, but the order of magnitude is known quite exactly: it has upper bound 2 n 2 − n log 2 ■ n + O (n) {\displaystyle 2^{n^{2}-n\log _{2}n+O(n)}} and lower bound 2 n 2 − n log 2 ■ n − O (n) {\displaystyle 2^{n^{2}-n\log _{2}n-O(n)}} . [36]

Any Boolean linear threshold function can be implemented with only integer weights. Furthermore, the number of bits necessary and sufficient for representing a single integer weight parameter is Θ (n ln ■ n) {\displaystyle \Theta (n\ln n)} . [36]

Universal approximation theorem

A single perceptron can learn to classify any half-space. It cannot solve any linearly nonseparable vectors, such as the Boolean exclusive-or problem (the famous "XOR problem").

A perceptron network with one hidden layer can learn to classify any compact subset arbitrarily closely. Similarly, it can also approximate any compactly-supported continuous function arbitrarily closely. This is essentially a special case of the theorems by George Cybenko and Kurt Hornik .

Conjunctively local perceptron

Perceptrons (Minsky and Papert, 1969) studied the kind of perceptron networks necessary to learn various Boolean functions.

Consider a perceptron network with n {\displaystyle n} input units, one hidden layer, and one output, similar to the Mark I Perceptron machine. It computes a Boolean function of type f : 2 n → 2 {\displaystyle f:2^{n}\to 2} . They call a function conjunctively local of order k {\displaystyle k} , iff there exists a perceptron network such that each unit in the hidden layer connects to at most k {\displaystyle k} input units.

Theorem. (Theorem 3.1.1): The parity function is conjunctively local of order n {\displaystyle n} .

Theorem. (Section 5.5): The connectedness function is conjunctively local of order Ω (n 1 / 2) {\displaystyle \Omega (n^{1/2})} .

Learning algorithm for a single-layer perceptron

Below is an example of a learning algorithm for a single-layer perceptron with a single output unit. For a single-layer perceptron with multiple output units, since the weights of one output unit are completely separate from all the others', the same algorithm can be run for each output unit.

For multilayer perceptrons , where a hidden layer exists, more sophisticated algorithms such as backpropagation must be used. If the activation function or the underlying process being modeled by the perceptron is nonlinear , alternative learning algorithms such as the delta rule can be used as long as the activation function is differentiable . Nonetheless, the learning algorithm described in the steps below will often work, even for multilayer perceptrons with nonlinear activation functions.

When multiple perceptrons are combined in an artificial neural network, each output neuron operates independently of all the others; thus, learning each output can be considered in isolation.

Definitions

We first define some variables:

r {\displaystyle r} is the learning rate of the perceptron. Learning rate is a positive number usually chosen to be less than 1. The larger the value, the greater the chance for volatility in the weight changes.

y = f (z) {\displaystyle y=f(\mathbf {z})} denotes the output from the perceptron for an input vector z {\displaystyle \mathbf {z} } .

D = { (x 1 , d 1) , … , (x s , d s) } {\displaystyle D=\{(\mathbf {x} _{1},d_{1}),\dots ,(\mathbf {x} _{s},d_{s})\}} is the training set of s {\displaystyle s} samples, where: x j {\displaystyle \mathbf {x} _{j}} is the n {\displaystyle n} -dimensional input vector. d j {\displaystyle d_{j}} is the desired output value of the perceptron for that input.

x j {\displaystyle \mathbf {x} _{j}} is the n {\displaystyle n} -dimensional input vector.

d j {\displaystyle d_{j}} is the desired output value of the perceptron for that input.

We show the values of the features as follows:

x j , i {\displaystyle x_{j,i}} is the value of the i {\displaystyle i} th feature of the j {\displaystyle j} th training input vector .

x j , 0 = 1 {\displaystyle x_{j,0}=1} .

To represent the weights:

w i {\displaystyle w_{i}} is the i {\displaystyle i} th value in the weight vector , to be multiplied by the value of the i {\displaystyle i} th input feature.

Because x j , 0 = 1 {\displaystyle x_{j,0}=1} , the w 0 {\displaystyle w_{0}} is effectively a bias that we use instead of the bias constant b {\displaystyle b} .

To show the time-dependence of w {\displaystyle \mathbf {w} } , we use:

w i (t) {\displaystyle w_{i}(t)} is the weight i {\displaystyle i} at time t {\displaystyle t} .

Steps

Initialize the weights. Weights may be initialized to 0 or to a small random value. In the example below, we use 0.

For each example j in our training set D , perform the following steps over the input x j {\displaystyle \mathbf {x} _{j}} and desired output d j {\displaystyle d_{j}} : Calculate the actual output: y j (t) = f [w (t) · x j] = f [w 0 (t) x j , 0 + w 1 (t) x j , 1 + w 2 (t) x j , 2 + ■ + w n (t) x j , n] {\displaystyle {\begin{aligned}y_{j}(t)&=f[\mathbf {w} (t)\cdot \mathbf {x} _{j}]\\&=f[w_{0}(t)x_{j,0}+w_{1}(t)x_{j,1}+w_{2}(t)x_{j,2}+\dotsb +w_{n}(t)x_{j,n}]\end{aligned}}} Update the weights: w i (t + 1) = w i (t) + r · (d j – y j (t)) x j , i {\displaystyle w_{i}(t+1)=w_{i}(t)\;{\boldsymbol {+}}\;r\cdot (d_{j}-y_{j}(t))x_{j,i}} , for all features 0 ≤ i ≤ n {\displaystyle 0\leq i\leq n} , r {\displaystyle r} is the learning rate .

Calculate the actual output: y j (t) = f [w (t) · x j] = f [w 0 (t) x j , 0 + w 1 (t) x j , 1 + w 2 (t) x j , 2 + ■ + w n (t) x j , n] {\displaystyle {\begin{aligned}y_{j}(t)&=f[\mathbf {w} (t)\cdot \mathbf {x} _{j}]\\&=f[w_{0}(t)x_{j,0}+w_{1}(t)x_{j,1}+w_{2}(t)x_{j,2}+\dotsb +w_{n}(t)x_{j,n}]\end{aligned}}}

Update the weights: w i (t + 1) = w i (t) + r · (d j – y j (t)) x j , i {\displaystyle w_{i}(t+1)=w_{i}(t)\;{\boldsymbol {+}}\;r\cdot (d_{j}-y_{j}(t))x_{j,i}} , for all features 0 ≤ i ≤ n {\displaystyle 0\leq i\leq n} , r {\displaystyle r} is the learning rate .

For offline learning , the second step may be repeated until the iteration error 1 s ∑ j = 1 s | d j – y j (t) | {\displaystyle {\frac {1}{s}}\sum _{j=1}^{s}|d_{j}-y_{j}(t)|} is less than a user-specified error threshold γ {\displaystyle \gamma } , or a predetermined number of iterations have been completed, where s is again the size of the sample set.

The algorithm updates the weights after every training sample in step 2b.

Convergence of one perceptron on a linearly separable dataset

A single perceptron is a linear classifier . It can only reach a stable state if all input vectors are classified correctly. In case the training set D is not linearly separable , i.e. if the positive examples cannot be separated from the negative examples by a hyperplane, then the algorithm would not converge since there is no solution. Hence, if linear separability of the training set is not known a priori, one of the training variants below should be used. Detailed analysis and extensions to the convergence theorem are in Chapter 11 of Perceptrons (1969).

Linear separability is testable in time min (O (n d / 2) , O (d 2 n) , O (n d – 1 ln ■ n)) {\displaystyle \min(O(n^{d/2}),O(d^{2n}),O(n^{d-1}\ln n))} , where n {\displaystyle n} is the number of data points, and d {\displaystyle d} is the dimension of each point. [37]

If the training set is linearly separable, then the perceptron is guaranteed to converge after making finitely many mistakes. [38] The theorem is proved by Rosenblatt et al.

Perceptron convergence theorem — Given a dataset D {\textstyle D} , such that max (x , y) ∈ D ■ x ■ 2 = R {\textstyle \max _{(x,y)\in D}\|x\|_{2}=R} , and it is linearly separable by some unit vector w ∗ {\textstyle w^{*}} , with margin γ {\textstyle \gamma } : γ := min (x , y) ∈ D y (w ∗ · x) {\displaystyle \gamma :=\min _{(x,y)\in D}y(w^{*}\cdot x)}

Then the perceptron 0-1 learning algorithm converges after making at most (R / γ) 2 {\textstyle (R/\gamma)^{2}} mistakes, for any learning rate, and any method of sampling from the dataset.

The following simple proof is due to Novikoff (1962). The idea of the proof is that the weight vector is always adjusted by a bounded amount in a direction with which it has a negative dot product , and thus can be bounded above by O (√ t) , where t is the number of changes to the weight vector. However, it can also be bounded below by O (t) because if there exists an (unknown) satisfactory weight vector, then every change makes progress in this (unknown) direction by a positive amount that depends only on the input vector.

Suppose at step t {\textstyle t} , the perceptron with weight w t {\textstyle w_{t}} makes a mistake on data point (x , y) {\textstyle (x,y)} , then it updates to w t + 1 = w t + r (y – f w t (x)) x {\textstyle w_{t+1}=w_{t}+r(y-f_{w_{t}}(x))x} .

If y = 0 {\textstyle y=0} , the argument is symmetric, so we omit it.

WLOG , y = 1 {\textstyle y=1} , then f w t (x) = 0 {\textstyle f_{w_{t}}(x)=0} , f w ∗ (x) = 1 {\textstyle f_{w^{*}}(x)=1} , and w t + 1 = w t + r x {\textstyle w_{t+1}=w_{t}+rx} .

By assumption, we have separation with margins: w ∗ ⋅ x ≥ γ {\displaystyle w^{*}\cdot x\geq \gamma } Thus, w ∗ ⋅ w t + 1 – w ∗ ⋅ w t = w ∗ ⋅ (r x) ≥ r γ {\displaystyle w^{*}\cdot w_{t+1}-w^{*}\cdot w_{t}=w^{*}\cdot (rx)\geq r\gamma }

Also ■ w t + 1 ■ 2 2 – ■ w t ■ 2 2 = ■ w t + r x ■ 2 2 – ■ w t ■ 2 2 = 2 r (w t ⋅ x) + r 2 ■ x ■ 2 2 {\displaystyle \|w_{t+1}\|_{2}^{2}-\|w_{t}\|_{2}^{2}=\|w_{t}+rx\|_{2}^{2}-\|w_{t}\|_{2}^{2}=2r(w_{t}\cdot x)+r^{2}\|x\|_{2}^{2}} and since the perceptron made a mistake, w t ⋅ x ≤ 0 {\textstyle w_{t}\cdot x\leq 0} , and so ■ w t + 1 ■ 2 2 – ■ w t ■ 2 2 ≤ ■ x ■ 2 2 ≤ r 2 R 2 {\displaystyle \|w_{t+1}\|_{2}^{2}-\|w_{t}\|_{2}^{2}\leq \|x\|_{2}^{2}\leq r^{2}R^{2}}

Since we started with w 0 = 0 {\textstyle w_{0}=0} , after making N {\textstyle N} mistakes, ■ w ■ 2 ≤ N r 2 R 2 {\displaystyle \|w\|_{2}\leq {\sqrt {Nr^{2}R^{2}}}} but also ■ w ■ 2 ≥ w ⋅ w ∗ ≥ N r γ {\displaystyle \|w\|_{2}\geq w\cdot w^{*}\geq Nr\gamma }

Combining the two, we have N ≤ (R / γ) 2 {\textstyle N\leq (R/\gamma)^{2}}

While the perceptron algorithm is guaranteed to converge on some solution in the case of a linearly separable training set, it may still pick any solution and problems may admit many solutions of varying quality. [39] The perceptron of optimal stability , nowadays better known as the linear support-vector machine , was designed to solve this problem (Krauth and Mezard , 1987). [40]

Perceptron cycling theorem

When the dataset is not linearly separable, then there is no way for a single perceptron to converge. However, we still have [41]

Perceptron cycling theorem — If the dataset D {\displaystyle D} has only finitely many points, then there exists an upper bound number M {\displaystyle M} , such that for any starting weight vector w 0 {\displaystyle w_{0}} all weight vector w t {\displaystyle w_{t}} has norm bounded by ■ w t ■ ≤ ■ w 0 ■ + M {\displaystyle \|w_{t}\|\leq \|w_{0}\|+M}

This is proved first by Bradley Efron . [42]

Learning a Boolean function

Consider a dataset where the x {\displaystyle x} are from { – 1 , + 1 } n {\displaystyle \{-1,+1\}^{n}} , that is, the vertices of an n-dimensional hypercube centered at origin, and y = θ (x i) {\displaystyle y=\theta (x_{i})} . That is, all data points with positive x i {\displaystyle x_{i}} have y = 1 {\displaystyle y=1} , and vice versa. By the perceptron convergence theorem, a perceptron would converge after making at most n {\displaystyle n} mistakes.

If we were to write a logical program to perform the same task, each positive example shows that one of the coordinates is the right one, and each negative example shows that its complement is a positive example. By collecting all the known positive examples, we eventually eliminate all but one coordinate, at which point the dataset is learned. [43]

This bound is asymptotically tight in terms of the worst-case. In the worst-case, the first presented example is entirely new, and gives n {\displaystyle n} bits of information, but each subsequent example would differ minimally from previous examples, and gives 1 bit each. After n + 1 {\displaystyle n+1} examples, there are 2 n {\displaystyle 2n} bits of information, which is sufficient for the perceptron (with 2 n {\displaystyle 2n} bits of information). [34]

However, it is not tight in terms of expectation if the examples are presented uniformly at random, since the first would give n {\displaystyle n} bits, the second n / 2 {\displaystyle n/2} bits, and so on, taking O (ln ■ n) {\displaystyle O(\ln n)} examples in total. [43]

Variants

The pocket algorithm with ratchet (Gallant, 1990) solves the stability problem of perceptron learning by keeping the best solution seen so far "in its pocket". The pocket algorithm then returns the solution in the pocket, rather than the last solution. It can be used also for non-separable data sets, where the aim is to find a perceptron with a small number of misclassifications. However, these solutions appear purely stochastically and hence the pocket algorithm neither approaches them gradually in the course of learning, nor are they guaranteed to show up within a given number of learning steps.

The Maxover algorithm (Wendemuth, 1995) is "robust" in the sense that it will converge regardless of (prior) knowledge of linear separability of the data set. [44] In the linearly separable case, it will solve the training problem – if desired, even with optimal stability (maximum margin between the classes). For non-separable data sets, it will return a solution with a computable small number of misclassifications. [45] In all cases, the algorithm gradually approaches the solution in the course of learning, without memorizing previous states and without stochastic jumps. Convergence is to global optimality for separable data sets and to local optimality for non-separable data sets.

The Voted Perceptron (Freund and Schapire, 1999), is a variant using multiple weighted perceptrons. The algorithm starts a new perceptron every time an example is wrongly classified, initializing the weights vector with the final weights of the last perceptron. Each perceptron will also be given another weight corresponding to how many examples do they correctly classify before wrongly classifying one, and at the end the output will be a weighted vote on all perceptrons.

In separable problems, perceptron training can also aim at finding the largest separating margin between the classes. The so-called perceptron of optimal stability can be determined by means of iterative training and optimization schemes, such as the Min-Over algorithm (Krauth and Mezard, 1987) [40] or the AdaTron (Anlauf and Biehl, 1989)). [46] AdaTron uses the fact that the corresponding quadratic optimization problem is convex. The perceptron of optimal stability, together with the kernel trick , are the conceptual foundations of the support-vector machine .

The α {\displaystyle \alpha } -perceptron further used a pre-processing layer of fixed random weights, with thresholded output units. This enabled the perceptron to classify analogue patterns, by projecting them into a binary space . In fact, for a projection space of sufficiently high dimension, patterns can become linearly separable.

Another way to solve nonlinear problems without using multiple layers is to use higher order networks (sigma-pi unit). In this type of network, each element in the input vector is extended with each pairwise combination of multiplied inputs (second order). This can be extended to an n -order network.

It should be kept in mind, however, that the best classifier is not necessarily that which classifies all the training data perfectly. Indeed, if we had the prior constraint that the data come from equi-variant Gaussian distributions, the linear separation in the input space is optimal, and the nonlinear solution is overfitted .

Other linear classification algorithms include Winnow , support-vector machine , and logistic regression .

Multiclass perceptron

Like most other techniques for training linear classifiers, the perceptron generalizes naturally to multiclass classification . Here, the input x {\displaystyle x} and the output y {\displaystyle y} are

drawn from arbitrary sets. A feature representation function f (x , y) {\displaystyle f(x,y)} maps each possible input/output pair to a finite-dimensional real-valued feature vector. As before, the feature vector is multiplied by a weight vector w {\displaystyle w} , but now the resulting score is used to choose among many possible outputs:

Learning again iterates over the examples, predicting an output for each, leaving the weights unchanged when the predicted output matches the target, and changing them when it does not. The update becomes:

This multiclass feedback formulation reduces to the original perceptron when x {\displaystyle x} is a real-valued vector, y {\displaystyle y} is chosen from { 0 , 1 } {\displaystyle \{0,1\}} , and f (x , y) = y x {\displaystyle f(x,y)=yx} .

For certain problems, input/output representations and features can be chosen so that a r g m a x y f (x , y) · w {\displaystyle \mathrm {argmax} _{y}f(x,y)\cdot w} can be found efficiently even though y {\displaystyle y} is chosen from a very large or even infinite set.

Since 2002, perceptron training has become popular in the field of natural language processing for such tasks as part-of-speech tagging and syntactic parsing (Collins, 2002). It has also been applied to large-scale machine learning problems in a distributed computing setting. [47]

References

Further reading

Aizerman, M. A. and Braverman, E. M. and Lev I. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control, 25:821–837, 1964.

Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386–408. doi : 10.1037/h0042519 .

Rosenblatt, Frank (1962), Principles of Neurodynamics. Washington, DC: Spartan Books.

Minsky, M. L. and Papert, S. A. 1969. Perceptrons . Cambridge, MA: MIT Press.

Gallant, S. I. (1990). Perceptron-based learning algorithms. IEEE Transactions on Neural Networks, vol. 1, no. 2, pp. 179–191.

Olazaran Rodriguez, Jose Miguel. A historical sociology of neural network research . PhD Dissertation. University of Edinburgh, 1991.

Mohri, Mehryar and Rostamizadeh, Afshin (2013). Perceptron Mistake Bounds arXiv:1305.0208, 2013.

Novikoff, A. B. (1962). On convergence proofs on perceptrons. Symposium on the Mathematical Theory of Automata, 12, 615–622. Polytechnic Institute of Brooklyn.

Widrow, B. , Lehr, M.A., " 30 years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation ," Proc. IEEE , vol 78, no 9, pp. 1415–1442, (1990).

Collins, M. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with the perceptron algorithm in Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '02).

Yin, Hongfeng (1996), Perceptron-Based Algorithms and Analysis, Spectrum Library, Concordia University, Canada

External links

A Perceptron implemented in MATLAB to learn binary NAND function

Chapter 3 Weighted networks - the perceptron and chapter 4 Perceptron learning of Neural Networks - A Systematic Introduction by Raúl Rojas (ISBN 978-3-540-60505-8)

History of perceptrons

Mathematics of multilayer perceptrons

Applying a perceptron model using scikit-learn - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

v

t

e

Differentiable programming

Information geometry

Statistical manifold

Automatic differentiation

Neuromorphic computing

Pattern recognition

Ricci calculus

Computational learning theory

Inductive bias

IPU

TPU

VPU

Memristor

SpiNNaker

TensorFlow

PyTorch

Keras

scikit-learn

Theano

JAX

Flux.jl

MindSpore

Portals Computer programming Technology

Computer programming

Technology

GND

United States

Japan

Israel

Yale LUX