Perceptron

"Perceptrons" redirects here. For the book of that title, see Perceptrons (book).

In machine learning, the **perceptron** is an algorithm for supervised classification of an input into one of several possible non-binary outputs. 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. The algorithm allows for online learning, in that it processes elements in the training set one at a time.

The perceptron algorithm dates back to the late 1950s; its first implementation, in custom hardware, was one of the first artificial neural networks to be produced.

1 History

> *See also: History of artificial intelligence, AI winter*

The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt,[1] funded by the United States Office of Naval Research.[2] 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 1 perceptron". This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.[3]:193

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."[2]

Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognise many classes of patterns. This led to the field of neural network research stagnating for many years, before it was recognised that a feedforward neural network with two or more layers (also called a multilayer perceptron) had far 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; 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 believed that they also conjectured (incorrectly) 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* for more information.) Three years later Stephen Grossberg published a series of papers introducing networks capable of modelling differential, contrast-enhancing and XOR functions. (The papers were published in 1972 and 1973, see e.g.: Grossberg, Contour enhancement, short-term memory, and constancies in reverberating neural networks. Studies in Applied Mathematics, 52 (1973), 213-257, online). Nevertheless the often-miscited Minsky/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. This text was reprinted in 1987 as "Perceptrons - Expanded Edition" where some errors in the original text are shown and corrected.

The kernel perceptron algorithm was already introduced in 1964 by Aizerman et al.[4] Margin bounds guarantees were given for the Perceptron algorithm in the general non-separable case first by Freund and Schapire (1998),[5] and more recently by Mohri and Rostamizadeh (2013) who extend previous results and give new L1 bounds.[6]

2 Definition

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

$$f(x) = \begin{cases} 1 & \text{if} w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

where w is a vector of real-valued weights, $w \cdot x$ is the dot product (which here computes a weighted sum), and b is the 'bias', a constant term that does not depend on any input value.

The value of $f(x)$ (0 or 1) is used to classify x as either a positive or a negative instance, in the case of a binary classification problem. If b is negative, then the weighted combination of inputs must produce a positive value greater than $|b|$ in order to push the classifier neuron over the 0 threshold. Spatially, the bias alters the position (though not the orientation) of the decision boundary. The perceptron learning algorithm does not terminate if the learning set is not linearly separable. If the vectors are not linearly separable learning will never reach a point where all vectors are classified properly. The most famous example of the perceptron's inability to solve problems with linearly nonseparable vectors is the Boolean exclusive-or problem. The solution spaces of decision boundaries for all binary functions and learning behaviors are studied in the reference.[7]

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.

3 Learning algorithm

Below is an example of a learning algorithm for a (single-layer) perceptron. For multilayer perceptrons, where a hidden layer exists, more sophisticated algorithms such as backpropagation must be used. Alternatively, methods such as the delta rule can be used if the function is non-linear and differentiable, although the one below will work as well.

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.

3.1 Definitions

We first define some variables:

- $y = f(\mathbf{z})$ denotes the *output* from the perceptron for an input vector $\mathbf{z}$.
- b is the *bias* term, which in the example below we take to be 0.
- $D = \{(\mathbf{x}_1, d_1), \ldots, (\mathbf{x}_s, d_s)\}$ is the *training set* of s samples, where:
  - $\mathbf{x}_j$ is the n -dimensional input vector.
  - d_j is the desired output value of the perceptron for that input.

We show the values of the features as follows:

- $x_{j,i}$ is the value of the i th feature of the j th training *input vector*.
- $x_{j,0} = 1$.

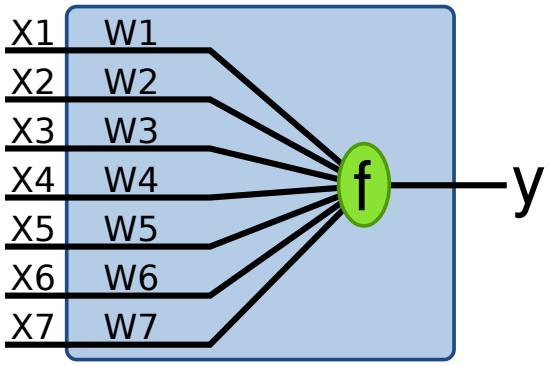
To represent the weights:

- w_i is the i th value in the *weight vector*, to be multiplied by the value of the i th input feature.
- Because $x_{j,0} = 1$, the w_0 is effectively a learned bias that we use instead of the bias constant b .

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

- $w_i(t)$ is the weight i at time t .
- α is the *learning rate*, where $0 < \alpha \leq 1$.

Too high a learning rate makes the perceptron periodically oscillate around the solution unless additional steps are taken.



The appropriate weights are applied to the inputs, and the resulting weighted sum passed to a function that produces the output y.

3.2 Steps

1. Initialise the weights and the threshold. Weights may be initialised to 0 or to a small random value. In the example below, we use 0.

2. For each example j in our training set D , perform the following steps over the input $\mathbf{x}_j$ and desired output d_j :

2a. Calculate the actual output:

$$y_j(t) = f[\mathbf{w}(t)\cdot\mathbf{x}_j] = f[w_0(t)+w_1(t)x_{j,1}+w_2(t)x_{j,2}+\cdots+w_n(t)x_{j,n}]$$

2b. Update the weights:

$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$, for all feature $0 \leq i \leq n$.

3. For offline learning, the step 2 may be repeated until the iteration error $\frac{1}{s}\sum_{j=1}^{s}|d_j - y_j(t)|$ is less than a user-specified error threshold γ , or a predetermined number of iterations have been completed.

The algorithm updates the weights after steps 2a and 2b. These weights are immediately applied to a pair in the training set, and subsequently updated, rather than waiting until all pairs in the training set have undergone these steps.

3.3 Convergence

The perceptron is a linear classifier, therefore it will never get to the state with all the input vectors classified correctly if the training set D is not linearly separable, i.e. if the positive examples can not be separated from the negative examples by a hyperplane.

But if the training set *is* linearly separable, then the perceptron is guaranteed to converge, and there is an upper bound on the number of times the perceptron will adjust its weights during the training.

Suppose that the input vectors from the two classes can be separated by a hyperplane with a margin γ , i.e. there exists a weight vector $\mathbf{w}$, $||\mathbf{w}|| = 1$, and a bias term b such that $\mathbf{w}\cdot\mathbf{x}_j + b > \gamma$ for all $j : d_j = 1$ and $\mathbf{w}\cdot\mathbf{x}_j + b < -\gamma$ for all $j : d_j = 0$. And also let R denote the maximum norm of an input vector. Novikoff (1962) proved that in this case the perceptron algorithm converges after making $O(R^2/\gamma^2)$ updates. The idea of the proof is that the weight vector is always adjusted by a bounded amount in a direction that it has a negative dot product with, and thus can be bounded above by $O(\sqrt{t})$ where t is the number of changes to the weight vector. But 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.

The decision boundary of a perceptron is invariant with respect to scaling of the weight vector; that is, a perceptron trained with initial weight vector $\mathbf{w}$ and learning rate α behaves identically to a perceptron trained with initial weight vector $\mathbf{w}/\alpha$ and learning rate 1. Thus, since the initial weights become irrelevant with increasing number of iterations, the learning rate does not matter in the case of the perceptron and is usually just set to 1.

4 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.

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)[8] or the AdaTron (Anlauf and Biehl, 1989)) .[9] 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 -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.

For example, consider the case of having to classify data into two classes. Here is a small such data set, consisting of two points coming from two Gaussian distributions.

- Two-class Gaussian data
- A linear classifier operating on the original space
- A linear classifier operating on a high-dimensional projection

A linear classifier can only separate points with a hyperplane, so no linear classifier can classify all the points here perfectly. On the other hand, the data can be projected into a large number of dimensions. In our example, a random matrix was used to project the data linearly to a 1000-dimensional space; then each resulting data point was transformed through the hyperbolic tangent function. A linear classifier can then separate the data, as shown in the third figure. However the data may still not be completely separable in this space, in which the perceptron algorithm would not converge. In the example shown, stochastic steepest gradient descent was used to adapt the parameters.

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.

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

5 Example

A perceptron learns to perform a binary NAND function on inputs x_1 and x_2 .

Inputs: x_0 , x_1 , x_2 , with input x_0 held constant at 1.

Threshold (t): 0.5

Bias (b): 0

Learning rate (r): 0.1

Training set, consisting of four samples: $\{((1,0,0),1),((1,0,1),1),((1,1,0),1),((1,1,1),0)\}$

In the following, the final weights of one iteration become the initial weights of the next. Each cycle over all the samples in the training set is demarcated with heavy lines.

This example can be implemented in the following Python code.

threshold = 0.5 learning_rate = 0.1 weights = [0, 0, 0] training_set = [((1, 0, 0), 1), ((1, 0, 1), 1), ((1, 1, 0), 1), ((1, 1, 1), 0)] def dot_product(values, weights): return sum(value * weight for value, weight in zip(values, weights)) while True: print('-' * 60) error_count = 0 for input_vector, desired_output in training_set: print(weights) result = dot_product(input_vector, weights) > threshold error = desired_output - result if error != 0: error_count += 1 for index, value in enumerate(input_vector): weights[index] += learning_rate * error * value if error_count == 0: break

6 Multiclass perceptron

Like most other techniques for training linear classifiers, the perceptron generalizes naturally to multiclass classification. Here, the input x and the output y are drawn from arbitrary sets. A feature representation function $f(x,y)$ 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 , but now the resulting score is used to choose among many possible outputs:

$$\hat{y} = \operatorname{argmax}_y f(x,y) \cdot w.$$

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:

$$w_{t+1} = w_t + f(x,y) - f(x,\hat{y}).$$

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

For certain problems, input/output representations and features can be chosen so that $\operatorname{argmax}_y f(x,y) \cdot w$ can be found efficiently even though y is chosen from a very large or even infinite set.

In recent years, 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).

7 References

[1] Rosenblatt, Frank (1957), The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

[2] Mikel Olazaran (1996). "A Sociological Study of the Official History of the Perceptrons Controversy". *Social Studies of Science* **26** (3).

[3] Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer.

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

[5] Freund, Y.; Schapire, R. E. (1999). "Large margin classification using the perceptron algorithm". *Machine Learning* **37** (3): 277–296. doi:10.1023/A:1007662407062.

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

[7] Liou, D.-R.; Liou, J.-W.; Liou, C.-Y. (2013). "Learning Behaviors of Perceptron". *ISBN 978-1-477554-73-9. iConcept Press.*

[8] W. Krauth and M. Mezard. Learning algorithms with optimal stabilty in neural networks. J. of Physics A: Math. Gen. 20: L745-L752 (1987)

[9] J.K. Anlauf and M. Biehl. The AdaTron: an Adaptive Perceptron algorithm. Europhysics Letters 10: 687-692 (1989)

- 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.
- Freund, Y. and Schapire, R. E. 1998. Large margin classification using the perceptron algorithm. In Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT' 98). ACM Press.
- Freund, Y. and Schapire, R. E. 1999. Large margin classification using the perceptron algorithm. In Machine Learning 37(3):277-296, 1999.
- Gallant, S. I. (1990). Perceptron-based learning algorithms. IEEE Transactions on Neural Networks, vol. 1, no. 2, pp. 179–191.
- 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

8 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)
- Explanation of the update rule by Charles Elkan
- History of perceptrons
- Mathematics of perceptrons
- Perceptron demo applet and an introduction by examples

9 Text and image sources, contributors, and licenses

9.1 Text

- **Perceptron** *Source:* http://en.wikipedia.org/wiki/Perceptron?oldid=629271648 *Contributors:* The Anome, Koyaanis Qatsi, Ap, Stevertigo, Lexor, Ahoerstemeier, Ronz, Muriel Gottrop, Glenn, IMSoP, Hike395, Furrykef, Benwing, Bernhard Bauer, Naddy, Rholton, Fuelbottle, Giftlite, Markus Krötzsch, BrendanH, Neilc, Pgan002, JimWae, Gene s, Hydrox, Luqui, Rama, Nwerneck, Robert.ensor, Poromenos, Caesura, Blahedo, Henry W. Schmitt, Hazard, Linas, Olethros, Male1979, Qwertyus, Margosbot, Predictor, YurikBot, RussBot, Gaius Cornelius, Hwasungmars, Gareth Jones, SamuelRiv, DaveWF, Nikkimaria, Closedmouth, Killerandy, SmackBot, Saravask, InverseHypercube, Pkirlin, CommodiCast, Eskimbot, El Cubano, Derkuci, Xyzzyplugh, Memming, Sumitkb, Tony esopi patra, Lambiam, Fishoak, Beetstra, CapitalR, Momet, RighteousRaven, Mcstrother, Tinyfool, Thijs!bot, Nick Number, Binarybits, Mat the w, Seaphoto, QuiteUnusual, Beta16, Jorgenumata, Pwmunro, Paskari, Joshua Issac, Shiggity, VolkovBot, TXiKiBoT, Xiaopengyou7, Ocolon, Kadiddlehopper, SieBot, Truecobb, AlanUS, CharlesGillingham, ClueBot, UKoch, MrKIA11, XLinkBot, Gnowor, Addbot, GVDoubleE, LinkFA-Bot, Lightbot, يمان, Luckas-bot, Yobot, Timeroot, SergeyJ, AnomieBOT, Phantom Hoover, Engshr2000, Materialscientist, Twri, GnawnBot, PabloCastellano, Emchristiansen, Bizso, Tuetschek, Octavianvoicu, Olympi, FrescoBot, Perceptrive, X7q, Mydimle, LauraHale, Cjlim, Gregman2, Igogo3000, EmausBot, Orphan Wiki, ZéroBot, Ohyusheng, Chire, Amit159, MrWink, JE71, John.naveen, Algernonka, ChuispastonBot, Sigma0 1, ClueBot NG, Biewers, Arrandale, Jackrae, Ricekido, MchLrn, BG19bot, Damjanmk, Whym, Dexbot, JurgenNL, Kn330wn, Ianschillebeeckx, Toritris, Terrance26, Francisbach, Kevin Leutzinger, Joma.huguet, Bjaress and Anonymous: 158

9.2 Images

- **File:Fisher_iris_versicolor_sepalwidth.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/40/Fisher_iris_versicolor_sepalwidth.svg *License:* CC-BY-SA-3.0 *Contributors:* en:Image:Fisher iris versicolor sepalwidth.png *Original artist:* en:User:Qwfp (original); Pbroks13 (talk) (redraw)
- **File:Internet_map_1024.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg *License:* CC-BY-2.5 *Contributors:* Originally from the English Wikipedia; description page is/was here. *Original artist:* The Opte Project
- **File:Perceptron.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/3/31/Perceptron.svg *License:* CC-BY-SA-3.0 *Contributors:* Created by mat_the_w, based on raster image File:Perceptron.gif by 'Paskari', using Inkscape 0.46 for OSX. *Original artist:* Mat the w at English Wikipedia

9.3 Content license

- Creative Commons Attribution-Share Alike 3.0