Data Science 2

Perceptron

Ondřej Týbl

Charles University, Prague

Recommended literature

- C.M. Bishop and N.M. Nasrabadi: Pattern Recognition and Machine Learning, [2]
- I. Goodfellow, Y. Bengio and A. Courville: Deep Learning, [3]

Outline

Perceptron

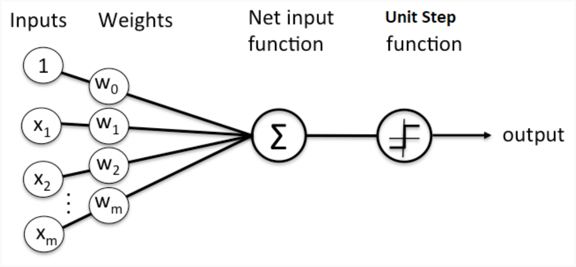
Definition

Definition (Perceptron)

Suppose a column vector $w = (w_1, \dots, w_m)^T \in \mathbb{R}^m$ and a bias term $w_0 \in \mathbb{R}$ are given. A classification function $f : \mathbb{R}^m \mapsto \{-1, 1\}$ of the form

$$f(x) = \operatorname{sign}\left(w^T x + w_0\right), \quad x \in \mathbb{R}^m$$

is called a *perceptron*.



Definition

If we classify the points $x \in \mathbb{R}^m$ using a label $f(x) \in \{-1, 1\}$ according to the law

$$f(x) = \begin{cases} 1, & w^T x + w_0 > 0 \\ -1, & w^T x + w_0 < 0, \end{cases}$$

we obtain sets of positive and negative examples

$$\{x \in \mathbb{R}^m : f(x) = 1\} \quad \text{and} \quad \{x \in \mathbb{R}^m : f(x) = -1\}$$

which are linearly separable with w being the normal vector of the separating hyperplane and $\frac{w_0}{||w||}$ being the offset from the origin.

Definition

A classification problem can be described by perceptron if and only if the sets of positive and negative examples are linearly separable (if and only if their convex hulls are disjoint).

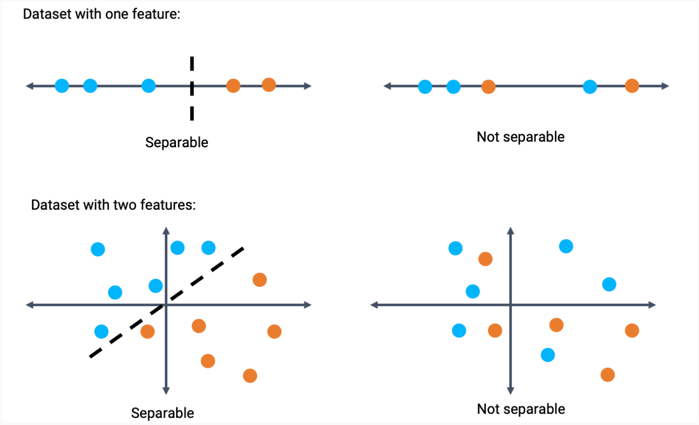


Figure 1: Linear Separability

Use cases

Suppose linearly separable data, that is training data $x_1, \dots, x_N \in \mathbb{R}^m$ with labels $y_1, \dots, y_N \in \{-1, 1\}$ are given such that there exist $w \in \mathbb{R}^m$ and $w_0 \in \mathbb{R}$ so that

$$y_j = 1 \iff w^T x_j + w_0 > 0$$

Algorithm (Training of perceptron, Rosenblatt, [6])

```
Initialize $w = 0$ and add the intercept $x_0 = 1$.
Iterate over samples $(x_j, y_j)$ until all of them are classified correctly:
    if $y_j w^T x_j < 0$:
        $w = w + y_j x_j$
```

Theorem (Convergence for perceptron, proof in Sec. 3.5.3 in [1])

The training always converges in finitely many steps to some correct set of weights w, w_0.

Use cases

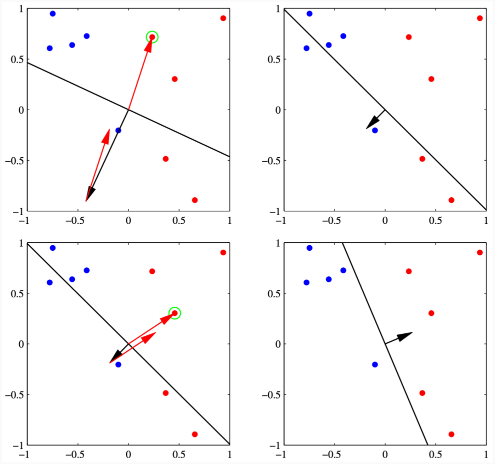


Figure 2: Example from [2] showing two steps of training with dataset in two dimensions.

Use cases

In [7] the authors introduced ADALINE, a variant of perceptron[1]

ADALINE has an input of 4×4 binary pixels and is capable of learning to distinguish between two letters – *J* and *T* in the demonstration (link) by one of the authors.

[1]Their model is exactly the same as our perceptron but the weights w, w_0 are trained slightly differently.

Use cases

As perceptron is capable of dealing only with linearly separable datasets, its applicability is indeed limited.

Claim (details in Sec. 40.3 in [4])

Let arbitrarily chosen points $x_1, \dots, x_N \in \mathbb{R}^m$ have independently assigned labels $y_1, \dots, y_N$ uniformly on $\{-1, 1\}$. Then as $N, m \to \infty$ the probability of the positive and negative examples being separable tends to 1 if $N < 2m$ and tends to 0 if $N > 2m$.

Use cases

A specific example of a dataset where the positive and negative examples are not separable is related to so called *XOR problem.*

We aim at training perceptron to respond to two binary inputs with 1 if and only if exactly one of them is positive[2].

[2]Modeled by training data $x_1 = (0, 0), x_2 = (0, 1), x_3 = (1, 0), x_4 = (1, 1)$ and $y_1 = y_4 = -1, y_2 = y_3 = 1$.

Use cases

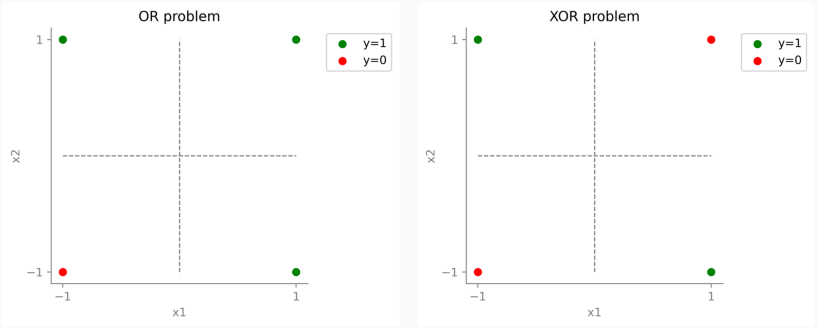


Figure 3: Perceptron is capable of learning the OR problem (left) but not the XOR problem (right). Here 0 (resp. 1 encodes negative (resp. positive) response.

Use cases

In [6] the author introduced Mark I Perceptron, another variant of perceptron.

Mark I Perceptron has an input of 20×20 binary pixels and is capable of learning to distinguish circles and squares[3] Two important tweaks are made

- In fact, two perceptrons are stacked, the first one with non-learnable fixed weights leading into a layer of 500 neurons,
- Also backward connections are present.

[3]With the accuracy of 99.8% on a test set after training on $10,000$ inputs.

Multilayer perceptron

Motivation

Problem

XOR is not learnable by any perceptron.

Observation

But OR, AND and $\neg$ are learnable and we have

$$\mathrm{XOR}(A, B) = (\mathrm{OR}(A, B))\,\mathrm{AND}\neg\,(\mathrm{AND}(A, B))$$

Idea

Train perceptrons to represent OR, AND and $\neg$ and stack them into a network.

Motivation

Define perceptrons $f_1, f_2, f_2 : \{0,1\}^2 \mapsto \{-1,1\}$ by

$$\begin{aligned} (\text{OR}) \quad & f_1(x) = \operatorname{sign}((1,1)x + 1/2) \\ (\text{AND}) \quad & f_2(x) = \operatorname{sign}((1,1)x - 1) \\ (\text{AND} + \neg) \quad & f_3(x) = \operatorname{sign}((1,-1)x - 1/2) . \end{aligned}$$

Then the composition

$$f(x) = f_3((f_1(x), f_2(x)))$$

represents XOR.

Definition

Definition (Activation function)

Any piecewise differentiable real function $\varphi : \mathbb{R} \mapsto \mathbb{R}$ is called an *activation function*.

We adopt a simplified notation for a component-wise application of activation function. That is, if φ is a real function, then

$$\varphi(v) = (\varphi(v_1), \ldots, \varphi(v_m)), \quad v \in \mathbb{R}^m$$

Definition

Definition (Multilayer perceptron)

Let $\varphi : \mathbb{R} \mapsto \mathbb{R}$ be an activation function. Suppose a matrix $W = \left(w_{ij}\right)_{i,j=1}^{n,m}$ and a bias vector $w_0 \in \mathbb{R}^n$ are given. A function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$ of the form

$$f(x) = \varphi\left(Wx + w_0\right), \quad x \in \mathbb{R}^m$$

is called a *multilayer perceptron*. Composition of *multilayer perceptrons* is also a *multilayer perceptron*.

In practice, φ will always be taken from a very short list of easy-to-implement functions which is the main limitation of f.

Definition

The activation function φ is the choice of the modeler and is chosen according to the type of problem at hand and so that the network has some desired properties. Both will be explained in detail later.

Usually, the activation function is one of the following: the identity, hyperbolic tangent, sigmoid, rectified linear unit,... all of them applied component-wise.

Definition

Multilayer perceptron can also be represented with an acyclic directed graph.

We distinguish *input layer*, *hidden layers* and *output layer*.

Usually, activation functions on all the hidden layers are the same and determine trainability properties; the activation function on the output layer is determined by the problem at hand.

Definition

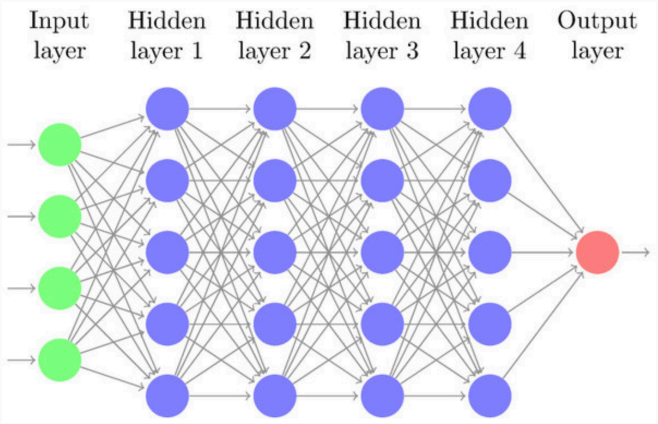


Figure 4: Graph representation of a multilayer perceptron with component-wise activations (i.e. more complex graph is needed in the case of softmax activation). Edges represent multiplication by scalar, nodes represent the activations. The bias vector is not directly presented.

Universal Approximation Theorem

Theorem (Universal Approximation, proof in Thm.3.1 [5])

Let $\varphi : \mathbb{R} \mapsto \mathbb{R}$ be a continuous function. Then φ *is not a polynomial* if and only if for every continuous mapping $F : K \mapsto \mathbb{R}^n$ from a compact subset $K \subset \mathbb{R}^m$ and every $\epsilon > 0$ there exist $W \in \mathbb{R}^{k \times m}, w_0 \in \mathbb{R}^k$ and $C \in \mathbb{R}^{n \times k}$ such that

$$\sup_{x \in K} \|F(x) - C\varphi\left(Wx + w_0\right)\| < \epsilon$$

In other words, any continuous function on a compact set can be approximated arbitrarily closely by a multilayer perceptron with a single hidden layer and any activation function which is not a polynomial[4].

[4]Theorem allows also non-differentiable activation functions. However, we do not consider such activations due to the way we will train the networks.

Universal Approximation Theorem

We give the proof in case of the rectified linear unit activation

$$ReLU(x) = \max(0, x).$$

Universal Approximation Theorem

The Universal Approximation Theorem can be applied as follows: suppose any dataset of distinct points $x_1, \ldots, x_N \in \mathbb{R}^m$ and arbitrary labels $y_1, \ldots, y_N \in \mathbb{R}$.

We can always construct a continuous function $F : \mathbb{R}^m \mapsto \mathbb{R}$ such that

$$F\left(x_j\right) = y_j, \quad j = 1, \ldots, N.$$

This function F can be then approximated arbitrarily closely with a neural network with one hidden layer.

Universal Approximation Theorem

Moreover, in the special case of binary classification $y_j \in \{-1, 1\}$ $j = 1, \ldots, N$ we obtain that even though the set of positive and negative examples might not be linearly separable, there always exist some *features* (represented by the hidden layer) that already constitute linearly separable sets[5].

[5]The Universal Approximation Theorem gives us an approximation only but we can approximate $-1, 1$ targets and then add signum function to obtain exact representation.

Some Notes – Training of Multilayer Perceptron

The procedure of finding the correct weights (if they exist) for a multilayer perceptron is not as straightforward as in the case of a single perceptron which was one of the reasons the research halted up until *backpropagation* came into play[6].

[6]Covered in the next lecture

Some Notes – Demonstration

Nice tool for investigation of the interplay between the training set and architecture of multilayer perceptron: link

References i

C. M. Bishop.
Neural networks for pattern recognition.
Oxford university press, 1995.

C. M. Bishop and N. M. Nasrabadi.
Pattern recognition and machine learning, volume 4.
Springer, 2006.

I. Goodfellow, Y. Bengio, and A. Courville.
Deep Learning.
MIT Press, 2016.
`http://www.deeplearningbook.org`.

D. J. MacKay.
Information theory, inference and learning algorithms.
Cambridge university press, 2003.

References ii

A. Pinkus.
Approximation theory of the mlp model in neural networks.
Acta numerica, 8:143–195, 1999.

F. Rosenblatt.
Principles of neurodynamics. perceptrons and the theory of brain mechanisms.
Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

B. Widrow and M. Hoff.
Adaptive switching circuits.
1960.
Reprinted in Anderson and Rosenfeld (1988).