

The Perceptron: the "not so artificial", artificial neuron

Berke Aslan (trash@berkeaslan.nl)

December 2024

Disclaimer: these are learning notes meant as a **work in progress**. As I learn more about the field of AI and all its intricacies, this and my other docs will be expanded on. This is by no means meant to be an exhaustive overview.

What is a perceptron?

The perceptron (or single-layer perceptron) is a supervised learning algorithm designed for binary classification and is one of the simplest forms of artificial neural networks. By some it is considered to be the first mathematical model that mimics the neuron's in the human brain (albeit not fully). I asked ChatGPT to explain biological neurons from this source (<https://www.sciencedirect.com/topics/neuroscience/dendrite>): "a biological neuron processes information through chemical and electrical signals: dendrites receive inputs, the soma integrates these signals, and if a threshold is reached, an action potential is sent via the axon to other neurons through synapses, with communication encoded by rates or timings of spikes." Hence the perceptron is a simple abstraction of the biological neuron (hence the title of this learning note).

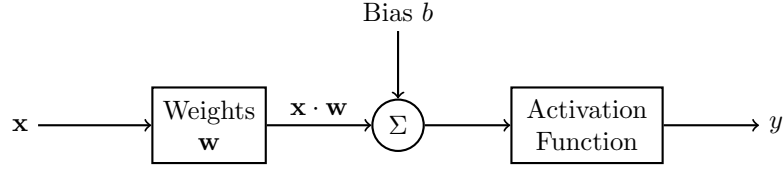
How it works

So how does a perceptron work? It's actually quite straightforward: a perceptron works by passing the sum of an input vector $\mathbf{x} \in \mathbb{R}^n$ (multiplied by weights \mathbf{w}) and a bias term b through an activation function. This activation function is often the Heaviside step function $H : \mathbb{R} \rightarrow \{0, 1\}$ defined as:

$$H(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases}$$

In other words, our model is given by $f : \mathbb{R}^n \rightarrow \{0, 1\}$ with f defined as:

$$f(x) = H(\mathbf{w}^T \mathbf{x} + b)$$



Input vector $\mathbf{x} \in \mathbb{R}^n$

Output scalar $y \in \{0, 1\}$

Figure 1: A schematic overview of the single-layer perceptron.

Alternatively, we could add the bias b as an element of vector \mathbf{w} . This can be done by setting the first element of \mathbf{x} as 1. Then the weight corresponding to this element would be the bias b . More formally:

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Now, you may wonder how the weights \mathbf{w} and the bias term b are determined. Recall that this is a supervised learning algorithm, so we have labeled training data. In other words, we already know the output scalars y for all input vectors \mathbf{x} in our training data set. The model learns by applying the "perceptron learning rule". Before we continue with an explanation, let us define the following (adapted from <https://en.wikipedia.org/wiki/Perceptron>):

- The learning rate $\alpha > 0$
- The output of the perceptron $\hat{y}_i = f(\mathbf{x}_i)$
- The training set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_k, y_k)\}$ with $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$ for $\forall i \in \{1, \dots, k\}$

Then the steps of the learning algorithm can be given by:

1. Initialize the weights: either to 0 or another small random value.
2. For each sample $j \in D$ perform the following steps:
 - (a) Compute the output \hat{y}_j of the perceptron for the current sample j :
 $\hat{y}_j(t) = f(\mathbf{w}(t) \cdot \mathbf{x}) = f(w_0(t)x_{j,0} + w_1(t)x_{j,1} + \dots + w_n(t)x_{j,n})$ with $x_{j,0} = 1$ and w_0 is the bias term.

- (b) Update the weights using the following rule: $w_i(t+1) = w_i(t) + \alpha(y_j - \hat{y}_j(t))x_{j,i}$.
3. Repeat the previous step until either a predetermined number of epochs is ran or until the termination condition (iteration error) is met: $\frac{1}{k} \sum_{i=1}^k |y_j - \hat{y}_j(t)| < \gamma$.

Linear separability

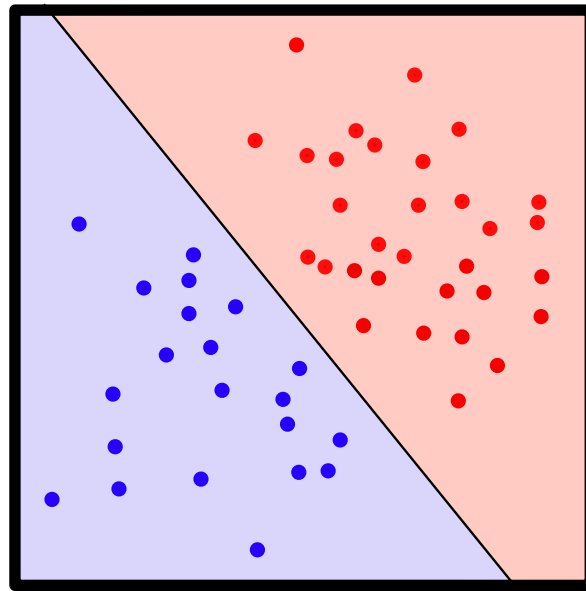


Figure 2: Graphic showing the linear separability of a feature space with a hyperplane (in this case a line) dividing the space in the separate classes (blue and red). Image source: Wikipedia.

Note, that, the data has to be linearly separable (https://en.wikipedia.org/wiki/Linear_separability) in order to utilize the perceptron. This means that there should be a hyperplane that divides the feature space such that points above and below the hyperplane belong to a certain class (that we try to predict). An example is shown in Figure 2.

Implementation in Python

In this section we will implement the perceptron in Python. Note, that, many different implementations are possible. We have not included any data to play

with, as this learning doc is more meant as a high-level explanation to understand the algorithm itself and its functioning. The code can be found in the same folder on GitHub.

```
1 import numpy as np
2 from typing import Optional
3
4 class Perceptron:
5     def __init__(self, alpha: float = 0.1, epochs: int = 100,
6         ↪ gamma: float = 0.01, weights: Optional[np.ndarray] =
7         ↪ None) -> None:
8         """
9         Initialize the Perceptron.
10
11         Parameters:
12         - alpha (float): Learning rate.
13         - epochs (int): Number of training iterations.
14         - gamma (float): Threshold for early stopping based on
15         ↪ iteration error.
16         - weights (Optional[np.ndarray]): Initial weights. If
17         ↪ None, initialized to zeros.
18         """
19         self.alpha = alpha
20         self.epochs = epochs
21         self.gamma = gamma
22         self.weights = weights
23
24     def __activation(self, x: float) -> int:
25         """
26         Activation function using the Heaviside step function.
27
28         Parameters:
29         - x (float): Weighted sum input.
30
31         Returns:
32         - int: 1 if x >= 0 else 0
33         """
34         return 1 if x >= 0 else 0 # Using 0 and 1 for binary
35         ↪ classification
36
37     def predict(self, X: np.ndarray) -> int:
38         """
39         Predict the class label for a single input sample.
40
41         Parameters:
```

```

37         - X (np.ndarray): Input feature vector.
38
39     Returns:
40     - int: Predicted class label (1 or 0).
41     """
42     X_augmented = np.insert(X, 0, 1) # Add bias term
43     weighted_sum = np.dot(self.weights, X_augmented)
44     return self.__activation(weighted_sum)
45
46 def fit(self, X: np.ndarray, y: np.ndarray) -> None:
47     """
48     Train the Perceptron model on the provided dataset.
49
50     Parameters:
51     - X (np.ndarray): Training feature matrix.
52     - y (np.ndarray): Training labels.
53     """
54     self.weights = np.zeros(X.shape[1] + 1) # Initialize
55     ↪ weights with bias
56
57     for epoch in range(self.epochs):
58         total_error = 0
59         for i in range(len(X)):
60             x_i = np.insert(X[i], 0, 1) # Add bias term
61             weighted_sum = np.dot(self.weights, x_i)
62             prediction = self.__activation(weighted_sum)
63             error = y[i] - prediction
64             total_error += abs(error)
65             self.weights += self.alpha * error * x_i
66
67         iteration_error = total_error / len(X)
68         print(f"Epoch {epoch + 1}/{self.epochs}, Iteration
69         ↪ Error: {iteration_error:.4f}")
70         if iteration_error < self.gamma:
71             print(f"Converged at epoch {epoch + 1} with
72             ↪ iteration error {iteration_error:.4f}")
73             break

```

Some further (unorganized) remarks and notes

Linear separability is a requirement when utilizing single-layer perceptrons (SLP). Furthermore, due to this constraint an SLP cannot learn non-linear decision boundaries (however, we could use multilayer perceptrons (MLPs) for that), a common problem hence is the "XOR problem". Above, we have only introduced the

Heaviside step function as the activation function. Note, however, that, there are multiple options for the activation function:

By reading through these learning notes, you will hopefully see that AI is a complex field, where even in a simple model like the perceptron, there are a lot of knobs that you can turn. For example, you can use different activation functions, different learning algorithms, etc.

Further learning

- The original paper by F. Rosenblatt: <https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf>.
- The Wikipedia page of the perceptron for a general overview: <https://en.wikipedia.org/wiki/Perceptron>.
- More reading on linear separability (also for other ML algorithms): <https://www.sciencedirect.com/topics/computer-science/linear-separability>.
- Some inspiration for further research: <https://www.sciencedirect.com/science/article/abs/pii/S0959438821000544>.