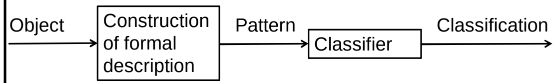
Statistical Pattern Recognition



The formal description consists of relevant numerical features, e.g., size, brightness, or curvature.

The pattern (or pattern vector, feature vector) is a vector of all chosen features for a given object, e.g. $(x_1, x_2, \ldots, x_n)$.

The description of any object corresponds to one point in the pattern space X.

Classification Principles

A statistical classifier is a device with n inputs and 1 output.

The input is a feature vector $(x_1, x_2, \ldots, x_n)$.

For an R-class classifier, its output is one of R symbols $w_1, w_2, \ldots, w_R$, which are the class identifiers.

The function $d(\mathbf{x}) = w_r$ is the classifier's decision rule.

It divides the feature space X into R disjoint subsets K_r, $r = 1, \ldots, R$.

Classification Principles

The function $d(\mathbf{x})$ can be defined using R scalar discrimination functions $g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_R(\mathbf{x})$.

For all $\mathbf{x} \in K_r$ and any $s \in \{1, \ldots, R\}$, $s \neq r$, the discrimination functions must be defined such that

$g_r(\mathbf{x}) \geq g_s(\mathbf{x})$.

This way, the discrimination hyper-surface between regions K_r and K_s is defined by

$g_r(\mathbf{x}) - g_s(\mathbf{x}) = 0$.

Classification Principles

The decision rule then becomes

$d(\mathbf{x}) = w_r \Leftrightarrow g_r(\mathbf{x}) = \max_{s=1, \ldots, R} g_s(\mathbf{x})$.

Often, linear classification functions are used:

$g_r(\mathbf{x}) = q_{r0} + q_{r1}x_1 + \ldots + q_{rn}x_n$.

In that case, the discrimination hyper-surfaces become discrimination hyper-planes.

Neural Networks

- The "building blocks" of neural networks are the **neurons**.
- In technical systems, we also refer to them as **units** or **nodes**.
- Basically, each neuron
  - receives **input** from many other neurons,
  - changes its internal state (**activation**) based on the current input,
  - sends **one output signal** to many other neurons, possibly including its input neurons (recurrent network)

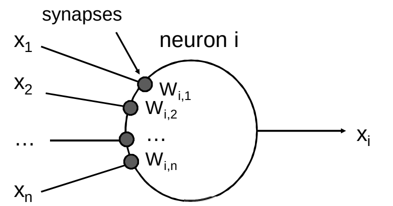
How do Neural Networks Work?

- Information is transmitted as a series of electric impulses, so-called **spikes**.
- The **frequency** and **phase** of these spikes encodes the information.
- In biological systems, one neuron can be connected to as many as **10,000** other neurons.
- Usually, a neuron receives its information from other neurons in a confined area, its so-called **receptive field**.

How do Neural Networks Learn?

- NNs are able to **learn** by **adapting their connectivity patterns** so that the organism improves its behavior in terms of reaching certain (evolutionary) goals.
- The strength of a connection, or whether it is excitatory or inhibitory, depends on the state of a receiving neuron's **synapses**.
- The NN achieves **learning** by appropriately adapting the states of its synapses.

An Artificial Neuron



net input signal $\text{net}_i(t) = \sum_{j=1}^{n} w_{i,j}(t) x_j(t)$

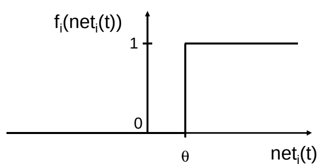
output $x_i(t) = f_i(net_i(t))$

The Activation Function

One possible choice is a **threshold function**:

$$f_i(\text{net}_i(t)) = 1, \quad \text{if} \quad \text{net}_i(t) \geq \theta$$
$$= 0, \quad \text{otherwise}$$

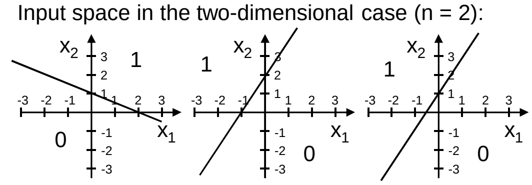
The graph of this function looks like this:



Linear Separability

$$f(x_1, x_2, ..., x_n) = 1, \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i \geq \theta$$
$$= 0, \quad \text{otherwise}$$

Input space in the two-dimensional case (n = 2):



$w_1 = 1$, $w_2 = 2$, $\theta = 2$ | $w_1 = -2$, $w_2 = 1$, $\theta = 2$ | $w_1 = -2$, $w_2 = 1$, $\theta = 1$

Linear Separability

So by varying the weights and the threshold, we can realize **any linear separation** of the input space into a region that yields output 1, and another region that yields output 0.

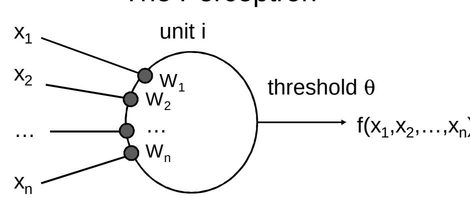
As we have seen, a **two-dimensional** input space can be divided by any straight line.

A **three-dimensional** input space can be divided by any two-dimensional plane.

In general, an **n-dimensional** input space can be divided by an (n-1)-dimensional plane or hyperplane.

Of course, for n > 3 this is hard to visualize.

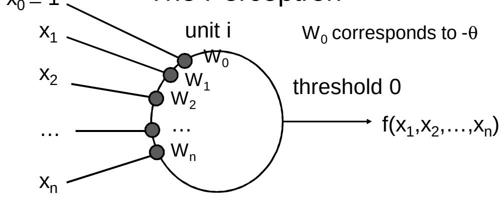
The Perceptron



net input signal $\text{net} = \sum_{i=1}^{n} w_i x_i$

output $f(\text{net}) = 1, \quad \text{if} \quad \text{net} \geq \theta$
$= -1, \quad \text{otherwise}$

The Perceptron



net input signal $\quad \text{net} = \sum_{i=0}^{n} w_i x_i$

output $\quad f(\text{net}) = 1, \quad \text{if} \quad \text{net} > 0$

$\qquad = -1, \quad \text{otherwise}$

Here, only the weight vector is adaptable, but not the threshold.

Perceptron Computation

A perceptron divides its n-dimensional input space by an (n-1)-dimensional hyperplane defined by the equation:

$w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n = 0$

For $w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0$, its output is 1, and

for $w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n \leq 0$, its output is -1.

With the right weight vector $(w_0, \dots, w_n)^T$, a single perceptron can compute any linearly separable function.

We are now going to look at an algorithm that determines such a weight vector for a given function.

Perceptron Training Algorithm

Algorithm Perceptron;

Start with a randomly chosen weight vector $\mathbf{w}_0$;

Let k = 1;

while there exist input vectors that are misclassified by $\mathbf{w}_{k-1}$, **do**

Let $\mathbf{i}_j$ be a misclassified input vector;

Let $\mathbf{x}_k = \text{class}(\mathbf{i}_j)\cdot\mathbf{i}_j$, implying that $\mathbf{w}_{k-1}\cdot\mathbf{x}_k < 0$;

Update the weight vector to $\mathbf{w}_k = \mathbf{w}_{k-1} + \eta\mathbf{x}_k$;

Increment k;

end-while;

Perceptron Training Algorithm

For example, for some input $\mathbf{i}$ with class($\mathbf{i}$) = -1,

If $\mathbf{w}\cdot\mathbf{i} > 0$, then we have a misclassification.

Then the weight vector needs to be modified to $\mathbf{w} + \Delta\mathbf{w}$

with $(\mathbf{w} + \Delta\mathbf{w})\cdot\mathbf{i} < \mathbf{w}\cdot\mathbf{i}$ to possibly improve classification.

We can choose $\Delta\mathbf{w} = -\eta\mathbf{i}$, because

$(\mathbf{w} + \Delta\mathbf{w})\cdot\mathbf{i} = (\mathbf{w} - \eta\mathbf{i})\cdot\mathbf{i} = \mathbf{w}\cdot\mathbf{i} - \eta\mathbf{i}\cdot\mathbf{i} < \mathbf{w}\cdot\mathbf{i}$,

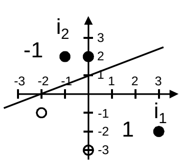
and $\mathbf{i}\cdot\mathbf{i}$ is the square of the length of vector $\mathbf{i}$ and is thus positive.

If class($\mathbf{i}$) = 1, things are the same but with opposite signs; we introduce $\mathbf{x}$ to unify these two cases.

Perceptron Learning Example

We would like our perceptron to correctly classify the five 2-dimensional data points below.

Let the random initial weight vector $\mathbf{w}_0 = (2, 1, -2)^T$.



Then the dividing line crosses at $(0, 1)^T$ and $(-2, 0)^T$.

Let us pick the misclassified point $(-2, -1)^T$ for learning:

$\mathbf{i} = (1, -2, -1)^T$ (include offset 1)

$\mathbf{x}_1 = (-1)\cdot(1, -2, -1)^T$ ($\mathbf{i}$ is in class -1)

$\mathbf{x}_1 = (-1, 2, 1)^T$

○ class -1

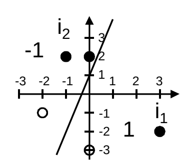
● class 1

Perceptron Learning Example

$\mathbf{w}_1 = \mathbf{w}_0 + \mathbf{x}_1$ (let us set $\eta = 1$ for simplicity)

$\mathbf{w}_1 = (2, 1, -2)^T + (-1, 2, 1)^T = (1, 3, -1)^T$

The new dividing line crosses at $(0, 1)^T$ and $(-1/3, 0)^T$.



Let us pick the next misclassified point $(0, 2)^T$ for learning:

$\mathbf{i} = (1, 0, 2)^T$ (include offset 1)

$\mathbf{x}_2 = (1, 0, 2)^T$ ($\mathbf{i}$ is in class 1)

○ class -1

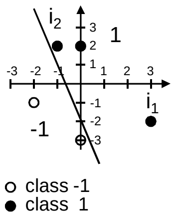
● class 1

Perceptron Learning Example

$\mathbf{w}_2 = \mathbf{w}_1 + \mathbf{x}_2$

$\mathbf{w}_2 = (1, 3, -1)^T + (1, 0, 2)^T = (2, 3, 1)^T$

Now the line crosses at $(0, -2)^T$ and $(-2/3, 0)^T$.



With this weight vector, the perceptron achieves perfect classification!

The learning process terminates.

In most cases, many more iterations are necessary than in this example.

Learning Rate and Termination

- Terminate when all samples are correctly classified.
- If the number of misclassified samples has not changed in a large number of steps, the problem could be the choice of learning rate η:
- If η is too large, classification may just be swinging back and forth and take a long time to reach the solution;
- On the other hand, if η is too small, changes in classification can be extremely slow.
- If changing η does not help, the samples may not be linearly separable, and training should terminate.
- If it is known that there will be a minimum number of misclassifications, train until that number is reached.

Guarantee of Success

Novikoff (1963) proved the following theorem:

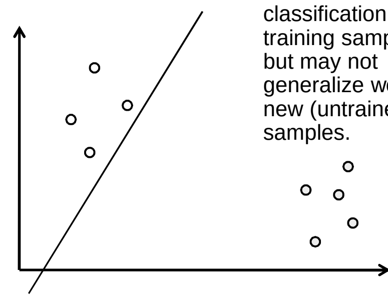
Given training samples from two linearly separable classes, the perceptron training algorithm terminates after a finite number of steps, and correctly classifies all elements of the training set, irrespective of the initial random non-zero weight vector $\mathbf{w}_0$.

But are those solutions optimal?

One of the reasons why we are interested in neural networks is that they are able to generalize, i.e., give plausible output for new (untrained) inputs.

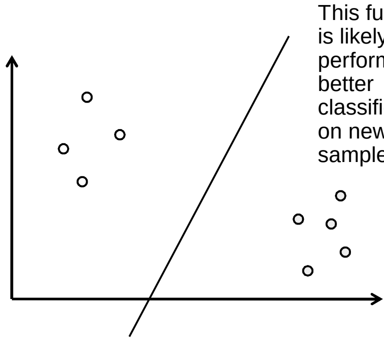
How well does a perceptron deal with new inputs?

Perceptron Learning Results



Perfect classification of training samples, but may not generalize well to new (untrained) samples.

Perceptron Learning Results



This function is likely to perform better classification on new samples.

Perceptron Learning Results

Perceptrons do not “care” about whether they found an optimal solution with regard to new samples.

They stop learning once perfect classification of the training samples has been achieved.

Therefore, results are often suboptimal for novel samples.

This is one of the reasons why perceptrons are rarely used in current applications.

However, their learning algorithm is simple and illustrates the general idea underlying neural classification approaches.