The flow of information from one neuron to another is typically based on the rate of spiking. The higher the rate, the more active the neuron.

In computer science, we typically abstract this rate into a single number, as if transmitted instantaneously.

We’ve already mentioned that signals can be abstracted into a single real number.

We sometimes further abstract into a two-valued set, such as {−1, +1} “bipolar”, or {0, 1} “unipolar” depending on the intended application.

The connection from one neuron to another has an associated efficiency, typically called the “synaptic strength” or simply “weight”.

Weights are usually represented as real numbers and can be positive or negative.

Although weights should be ascribed to connections, it is common to depict them as associated with the input side of the neuron.

Early efforts at modeling neural networks used threshold logic, which is still valid for some types of applications.

– The input values are discrete, say {0, 1}.

– The weighted sum minus the threshold is called the “activation” or “net” value.

– The neuron fires if activation value is above a threshold value associated with the neuron.

– Output is 1 if the neuron fires, 0 otherwise.
Threshold Logic Behavior

Say the threshold happens to be -0.2.

Inputs

\[ +3 \]
\[ -7 \]
\[ +1.5 \]

\[ 0 \]

activation = \[ 1 \times 3 + 1 \times (-7) + 0 \times 1.5 - (-0.2) = -0.2 \]
activation < 0 so the neuron does not fire (output = 0)

Threshold Logic Behavior

Again, suppose the threshold happens to be -0.2.

Inputs

\[ 1 \]
\[ -7 \]
\[ 0 \]
\[ +1.5 \]

activation = \[ 1 \times 3 + 0 \times (-7) + 0 \times 1.5 - (-0.2) = 0.5 \]
activation > 0 so the neuron does fire (output = 1)

Rosenblatt’s Perceptron, 1957

A perceptron in simplest form is a linear threshold gate.

More generally, it could have reals as inputs rather than just \{0, 1\}. But it still has \{0, 1\} as output.

Introduced the idea of training for determining weights to achieve a given function

Application of Perceptrons

Classification problems:

– Given a pre-specified set of input vectors, each with a desired response
  \( (1 = \text{in the set}, 0 = \text{not in the set}) \), determine the weights so that a perceptron gives the desired response (and generalizes in an appropriate way to unseen inputs)

– An input vector could be a retinal image, for example.

Retinal Image Classification Model

Given a classification problem, try to find a perceptron to fit.

Find a vector of weights \( \{w_i\} \) and a threshold \( \theta \), such that:

\[
\text{output} = \begin{cases} 
1 & \text{if } \sum w_i x_i > \theta \\
0 & \text{otherwise} \\
1 & \text{if } (x_i) \text{ represents a vector in the set} \\
0 & \text{otherwise (not in the set)}
\end{cases}
\]

vector of \( m \times n \) bits, RGB, or gray-scale levels.
### Issues

**Existence**: Given a set of input vectors, does there exist a perceptron that correctly classifies the given inputs?

**Solving**: Can the weights be found, e.g. analytically or numerically?

**Training**: Can the weights be found simply by presenting examples?

### Table

<table>
<thead>
<tr>
<th>$n$</th>
<th>NUMBER OF THRESHOLD FUNCTIONS $a_n$</th>
<th>TOTAL NUMBER OF BOOLEAN FUNCTIONS $2^{a_n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>104</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>1,382</td>
<td>65,536</td>
</tr>
<tr>
<td>5</td>
<td>94,372</td>
<td>4.3x10^6</td>
</tr>
<tr>
<td>6</td>
<td>15,025,134</td>
<td>1.4x10^11</td>
</tr>
<tr>
<td>7</td>
<td>8,278,070,864</td>
<td>3.4x10^17</td>
</tr>
<tr>
<td>8</td>
<td>17,061,239,552,946</td>
<td>1.16x10^22</td>
</tr>
</tbody>
</table>

### Existence

In terms of switching logic, a perceptron is a “linear threshold gate” (LTG).

As the number of inputs increases, the likelihood of realizing a function as an LTG diminishes.

### 1-dimensional version

1 input, 1 weight, 1 threshold

output = \[\begin{cases} 1 & \text{if } wx > \theta \\ 0 & \text{otherwise} \end{cases}\]

$x = \text{output is 1}$

$0 = \text{output is 0}$

Which sets below are solvable by Perceptron? i.e. what are suitable $w$ and $\theta$?

a. $\circ \circ \circ \dots$ input

b. $\circ \circ \circ \circ \circ \circ \circ \circ$ input

c. $\circ \circ \circ \circ \circ \circ \circ \circ$ input

### 2-dimensional version

2 inputs, 2 weights, 1 threshold

output = \[\begin{cases} 1 & \text{if } \sum_{i=1}^{2} w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}\]

$x = \text{output is 1}$

$0 = \text{output is 0}$

Which sets below are solvable by Perceptron? What are suitable $w$ and $\theta$?

- $\circ \circ \circ \circ \circ \circ \circ \circ$ input

### General Line Equation

Output is

- 1 if $w_1 x_1 + w_2 x_2 > \theta$
- 0 otherwise

When will $(w_i)$ and $\theta$ exist?

Exactly when a straight line can be drawn that separates the points ("linearly separable")

$w_1 x_1 + w_2 x_2 = \theta$ is the equation of such a line.
Generalizing Linear Separability to $n$ dimensions

Given function $d: \mathbb{R}^n \to \{0, 1\}$ (d for desired), to find $q, w_i \in \mathbb{R}$ such that

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = \theta$$

separates the points:

- $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$ when $d(x_1, x_2, \ldots, x_n) = 1$
- $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \leq \theta$ when $d(x_1, x_2, \ldots, x_n) = 0$

General Linear Separability, $n$ dimensions

The equation

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = \theta$$

defines a hyperplane in $n$-space

If such a hyperplane exists for a classification problem, the problem is called linearly-separable.

Geometry

The vector of weights $(w_1, w_2, \ldots, w_n)$ is normal (perpendicular) to the hyperplane.

Threshold is proportional to the distance of the hyperplane from the origin.

Perceptron Summary

A perceptron can solve a classification problem iff the problem is linearly-separable.

There are problems a single perceptron cannot solve.

Perhaps the simplest unsolvable one is the XOR problem:

1 output: $\{0, 1\}$,
0 output: $\{0, 0\}, \{1, 1\}$
**Perceptron Learning**

**General Perceptron Training**

Assuming weights exist for a problem (linear separability), how to find them?

- Analytic? (technically not “training”)
  - A conventional perceptron doesn’t lend well to analytic solution, due to the non-linear nature of the threshold function (a step function).
- Successive approximations?

**A Somewhat General Approach**

Use a set of training samples:
- Input vectors, each paired with desired output
Choose a network structure.
Initialize the weights and thresholds arbitrarily.
Repeat:
  - Choose a sample
  - Test the network against the sample
  - If answer is incorrect, adjust weights until all samples test correctly.

**Sample Choice**

Method 1: Simply cycle through all of the samples in a fixed order.
Method 2: Choose samples randomly, but with some assurance that each will be checked before stopping.

**Perceptron Learning Rule**

The Perceptron learning rule (Rosenblatt):
- If the perceptron gives the correct answer, do nothing.
- If the perceptron gives the wrong answer, nudge the weights and threshold “in the right direction”, so that it eventually gives the right answer.
Inner-Product Interpretation

\[ w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \] is the inner product \( wx \) of two vectors, \( w \) and \( x \).

The more closely-aligned the directions of these vectors are, the higher the value.

In fact, \( wx = |w| |x| \cos \theta \) where \( \theta \) is the angle between \( w \) and \( x \) and \( |w| \) is the length of \( w \).

Values of \( wx \)

Adjusting \( w \) by vector addition

Let \( \eta \) be a small scalar value.

If an increase of \( wx \) is desired, add \( \eta x \):

If an decrease of \( wx \) is desired, subtract \( \eta x \):

Perceptron Learning Rule

If \( wx \) is negative, but it should be positive, add \( \eta x \) to \( w \).

If \( wx \) is positive, but it should be negative, subtract \( \eta x \) from \( w \).

If \( wx \) is as desired, do nothing.
Perceptron Training Semi-Algorithm

Keep applying the learning rule, until all examples are correctly classified.

Effectively this rotates the weight vector, and thus the hyperplane to which it is a normal, into an orientation where the examples are correctly classified.

This process only terminates if the points are linearly separable. Therefore, it is best to impose an artificial limit on the number of steps just in case.

Training for Threshold

In addition to rotating the hyperplane, we also need to adjust its distance from the origin.

We can handle this along with adjusting other weights by making the dimension 1 higher and treating the threshold as another weight, against a constant input of -1.

We’ll provide details in a moment.

Error Value

We can capture correct vs. incorrect answers succinctly by introducing an error value \( \varepsilon \):

\[
\varepsilon = d(x_1, x_2, \ldots, x_n) - a(x_1, x_2, \ldots, x_n) = \text{desired output} - \text{actual output}
\]

So that

- \( \varepsilon = 0 \) when the correct answer is given
- \( \varepsilon = 1 \) when \( w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < \theta \) but \( d(x_1, x_2, \ldots, x_n) = 1 \)
- \( \varepsilon = -1 \) when \( w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta \) but \( d(x_1, x_2, \ldots, x_n) = 0 \)

Error vs. Threshold

Instead of subtracting the threshold, we could add a “bias”, in which case the phantom input would be 1 rather than -1.

The actual phantom value \( \{1, -1, \ldots\} \) doesn’t really matter, as long as it is constant and not 0.

The network will learn to accommodate the given phantom value.

Desired Output vs. Actual Output a

Define actual output

\[
a(x_1, x_2 \ldots, x_n) = \\
1 \text{ if } w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta \\
0 \text{ otherwise.}
\]

Note that a is an implied function of the weights and threshold.

Simplification: Threshold conversion

The threshold can be treated as if one of the weights by introducing a “phantom” input of -1:

\[
w_1 x_1 + w_2 x_2 + \ldots + w_n x_n - \theta > 0
\]

iff

\[
w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta
\]

iff

\[
w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0
\]

where \( w_0 \) is defined to be \( \theta \) and \( x_0 = -1 \) unchangingly.
Perceptron training (continued)

Now we characterize answers by the error value $\varepsilon$:

$\varepsilon = 1$: (Wrong answer of the first type):

$$w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < 0$$

when $d(x_1, x_2, \ldots, x_n) = 1$

i.e., $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$

is too low.

To correct for this, we need to make the sum higher.

$\varepsilon = -1$: (Wrong answer of the second type):

$$w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$$

when $d(x_1, x_2, \ldots, x_n) = 0$

i.e., $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$

is too high.

To correct for this, we need to make the sum lower.

Perceptron training (continued)

Make $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$

lower or higher by adjusting weights.

- $\varepsilon = 1$: Make each contribution $w_i x_i$ higher
- $\varepsilon = -1$: Make each contribution $w_i x_i$ lower

In either case, can add some multiple $\eta$ (called the "learning rate") of $\varepsilon x_i$ to $w_i$ to get the desired effect.

Add $\eta x_i$ to $w_i$ to get the desired effect:

$$w_i x_i = (w_i x_i + \eta x_i)^2$$

if $\varepsilon > 0$

$$w_i x_i = w_i x_i$$

if $\varepsilon < 0$

($\varepsilon = \text{desired} - \text{actual}$, so increase actual to decrease $\varepsilon$, decrease actual to increase $\varepsilon$)

Note that if $\varepsilon = 0$, nothing will be added.

Learning Rate $\eta$

$\eta$ governs the rate at which the training rule converges toward the correct solution.

Typically $\eta \leq 1$.

Too small an $\eta$ produces slow convergence.

Too large of an $\eta$ can cause oscillations in the process.

Example

Train a perceptron to classify according to:

- $(4, 6)$ 1
- $(6, 1)$ 1
- $(4, 1)$ 0
- $(1, 2)$ 0

There will be three weights ($w_0, w_1, w_2$) where is the $w_0$

threshold, corresponding to phantom input -1.

Start with "random" weights, say (0, +1, -1)

Choose $\eta = 1$. 

### Perceptron Training Example: One Pass over Data Samples

Fill out this table sequentially:

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 1, -1)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 6, 1)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 4, 1)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 1, 2)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Perceptron Training Example: Second Pass over Data Samples

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0, 1)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 6, 1)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 4, 1)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 1, 2)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Perceptron Training Example: 3rd Pass over Data Samples

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2, 1, -1)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 6, 1)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 4, 1)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 1, 2)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Perceptron Training Example: 4th Pass over Data Samples

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3, 0, 1)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 6, 1)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 4, 1)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1, 1, 2)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Perceptron Training Example: 5th Pass over Data Samples

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4, 1, -1)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(3, 5, 4)</td>
</tr>
<tr>
<td>(3, 5, 4)</td>
<td>(-1, 6, 1)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>(3, 5, 4)</td>
<td>(-1, 4, 1)</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>(4, 1, 3)</td>
</tr>
<tr>
<td>(4, 1, 3)</td>
<td>(-1, 1, 2)</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>(5, 0, 1)</td>
</tr>
</tbody>
</table>

### Perceptron Training Example: 6th Pass over Data Samples

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5, 0, 1)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(4, 4, 6)</td>
</tr>
<tr>
<td>(4, 4, 6)</td>
<td>(-1, 6, 1)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>(4, 4, 6)</td>
<td>(-1, 4, 1)</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>(5, 0, 5)</td>
</tr>
<tr>
<td>(5, 0, 5)</td>
<td>(-1, 1, 2)</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>(6, -1, 3)</td>
</tr>
</tbody>
</table>

### Perceptron Training Example: 7th Pass over Data Samples

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(6, -1, 3)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>(6, -1, 3)</td>
<td>(-1, 6, 1)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(5, 5, 4)</td>
</tr>
<tr>
<td>(5, 5, 4)</td>
<td>(-1, 4, 1)</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>(6, 1, 3)</td>
</tr>
<tr>
<td>(6, 1, 3)</td>
<td>(-1, 1, 2)</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>(7, 0, 1)</td>
</tr>
</tbody>
</table>

### Perceptron Training Example: 8th Pass over Data Samples

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(7, 0, 1)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(6, 4, 6)</td>
</tr>
<tr>
<td>(6, 4, 6)</td>
<td>(-1, 6, 1)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>(6, 4, 6)</td>
<td>(-1, 4, 1)</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>(7, 0, 5)</td>
</tr>
<tr>
<td>(7, 0, 5)</td>
<td>(-1, 1, 2)</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>(8, -1, 3)</td>
</tr>
</tbody>
</table>

### Perceptron Training Example: 9th Pass over Data Samples

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(8, 1, 3)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>(8, -1, 3)</td>
<td>(-1, 6, 1)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(7, 5, 2)</td>
</tr>
<tr>
<td>(7, 5, 2)</td>
<td>(-1, 4, 1)</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>(8, 1, 3)</td>
</tr>
<tr>
<td>(8, 1, 3)</td>
<td>(-1, 1, 2)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(8, 1, 3)</td>
</tr>
</tbody>
</table>

### Perceptron Training Example: 10th Pass over Data Samples

<table>
<thead>
<tr>
<th>weights</th>
<th>input</th>
<th>desired</th>
<th>actual</th>
<th>error</th>
<th>new weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>(8, 1, 3)</td>
<td>(-1, 4, 5)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>(8, 1, 3)</td>
<td>(-1, 6, 1)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>(8, 1, 3)</td>
<td>(-1, 4, 1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>(8, 1, 3)</td>
<td>(-1, 1, 2)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>no change</td>
</tr>
</tbody>
</table>
Perceptron Training Example: 12th Pass over Data Samples

A perceptron with weights (8, 1, 3) correctly classifies all inputs.

The "yes" criterion is therefore:

\[-8 \cdot x_1 + 3 \cdot x_2 > 0\]  \[\text{[i.e. } x_1 + 3 \cdot x_2 > 8]\]

Check:

<table>
<thead>
<tr>
<th>input x1, x2</th>
<th>desired</th>
<th>actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4, 5)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(6, 1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(4, 1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Perceptron Training Algorithm (1)

Inputs:
- A list of training samples, each of the form
  • [d(x1, x2, ..., xn), -1, x1, x2, ..., xn]
  • (d is the desired output, -1 the phantom input)
- An initial weight vector [w0, w1, w2, ..., wn]
  • (w0 is the threshold)
- A learning rate \(\eta\)

Perceptron Training Algorithm (2)

Outputs:
- If the set of samples is linearly separable, a vector of weights 
  \([w_0, w_1, w_2, ..., w_n]\) such that with these weights the perception properly separates the training samples.

- If the set of samples is not linearly separable, then the algorithm diverges.

Perceptron Training Algorithm (3)

Operation:
- Set \([w_0, w_1, w_2, ..., w_n]\) = initial weights;
- while (there is a sample not correctly classified)
  • Let \([d(x_1, x_2, ..., x_n), -1, x_1, x_2, ..., x_n]\) be an incorrectly classified sample.
  • Let \(\varepsilon = d(x_1, x_2, ..., x_n) - a(x_1, x_2, ..., x_n)\), where \(a(x_1, x_2, ..., x_n) = 1\) if \(\sum w_i x_i > 0\), 0 otherwise.
  • Vector-add to \([w_0, w_1, w_2, ..., w_n]\) the vector \(\Delta w = \varepsilon \eta [-1, x_1, x_2, ..., x_n]\)

Perceptron Training Algorithm Modifications for practical usage

- Put a limit on the number of iterations, so that the algorithm will terminate (without perfect classification) even if the sample set is not linearly separable.
- Include an error bound as an extra input. The algorithm can stop as soon as the portion of mis-classified samples is less than this bound (as opposed to requiring perfect classification, which would be an error bound of 0).
- Generate the initial weights randomly, so that the user does not have to specify them. Or just start with all 0's (but this won't work in more advanced models).

Correctness of the Perceptron Training Algorithm assuming linear separability
Some Simplifications

All training vectors \( x \) (including the phantom -1) can be normalized, by dividing by the length \( |x| \), so that \( |x| = 1 \).

This is because only the sign of \( wx \) matters in classification, and the sign of \( wx \) is the same as that of \( wx/|x| \).

The weight vector \( w \) can also be normalized, so that \( |w| = 1 \), by the same rationale.

All training samples can be assumed to have positive desired value.

If the desired value were to be negative, it can be made positive just by complementing all of the components of the sample.

Visualization

For normalized \( w \) and \( x \), the value \( wx \) is the cosine of the angle \( \alpha \) between \( w \) and \( x \).

We want to find a weight vector which has a positive value of \( \cos \alpha \) with each \( x \).

A training step consists of adding vector \( \eta x \) (proportional to \( x \)) to \( w \), to push \( w \) away if \( \cos \alpha < 0 \).

We can assume the learning rate is \( \eta = 1 \) without loss of generality.

We can assume the weights are all initially 0, without loss of generality.

Learning Based on Mis-Classified Samples

(from before) An increase of \( wx \) is desired:

\[
\begin{align*}
\text{w} + \eta x & \quad \text{is closer to } x \\
\eta x & \quad \text{than } w \text{ was.}
\end{align*}
\]

\( \eta \leq 1 \)
Convergence Proof

Assume all samples are positive and normalized.
Assume the weights are normalized.

Claim: If a weight vector $w^*$ exists that correctly classifies all samples, one will be found by the perceptron training algorithm.

Without loss of generality, assume the learning rate $\eta$ is 1.
WLOG, assume classification is strict, i.e. $w^* x > 0$ for all $x$.
Also may assume $|w^*| < 1$, as its length does not matter.

Proof (following Rojas, pp 88-89)

Assume $w^*$ exists.
($w^*$ classifies all samples correctly.)

Let $w_t$ be the weight vector after $t$ steps of the algorithm.

Consider step $t+1$, which must have resulted from some vector $x_i$ being *misclassified* by $w_t$.
Thus $w_{t+1} = w_t + x_i$ (because the error is 1 and $\eta = 1$).


Proof continues

Consider angle $\rho$ where $\cos \rho = w^* (w_{t+1} / |w_{t+1}|)$ (inner product) (RHS must be $\leq 1$).

Factoring the numerator.

$w^* w_{t+1} = w^* (w_t + x_i)$

$= w^* w_t + w^* x_i$

$\geq w^* w_t + \delta$,

where $\delta = \min\{w^* x_j | \text{ samples } x_j\} > 0$, due to strict classification.

By inductive substitution, for all $t$

$w^* w_{t+1} \geq w^* w_t + (t+1)$, where $w_0$ is the initial weight vector.

So we have linear lower bound on $w^* w_{t+1}$, which translates to a quadratic lower bound on $|w_{t+1}|^2$.

Again, $w^* w_{t+1} / |w_{t+1}| \leq 1$.

Squaring the denominator.

$|w_{t+1}|^2 = (w_t + x_i) (w_t + x_i)$

$= |w_t|^2 + 2w_t x_i + |x_i|^2$

But $w_t x_i < 0$, because $x_i$ was misclassified, so

$|w_{t+1}|^2 \leq |w_t|^2 + |x_i|^2$

$\leq |w_t|^2 + 1$, since $x_i$ are normalized.

By inductive substitution,

$|w_{t+1}|^2 \leq |w_0|^2 + (t + 1)$

So we have a linear upper bound on $|w_t|^2$

Proof concludes

From two inequalities derived on the previous two slides:

$1 \geq \frac{(w^* w_0 + x(t+1))}{\sqrt{w_0^2 + (t+1)}}$

The RHS as a function of $t$ is bounded above.

Hence there is a maximum step $t$ for which classification can be incorrect.

Thus the algorithm terminates, because there are no incorrect classifications beyond step $t$.

Biological Plausibility

Discuss whether or not the perceptron algorithm is a plausible explanation for learning in real neurons.