**Training program for ANN perceptron**

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: \{((0, 0), 1), ((0, 1), 1), ((1, 0), 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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | | | Initial  weights | | | Output | | | | | Error | Correction | Final weights | | |
| Sensor values | | | Desired output | Per sensor | | | Sum | Network |
| x_0 | x_1 | x_2 | z | w_0 | w_1 | w_2 | c_0 | c_1 | c_2 | s | n | e | d | w_0 | w_1 | w_2 |
|  |  |  |  |  |  |  | x_0*w_0 | x_1*w_1 | x_2*w_2 | c_0+c_1+c_2 | if s>tthen 1, else 0 | z-n | r * e | \Delta(x_0*d) | \Delta(x_1*d) | \Delta(x_2*d) |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | +0.1 | 0.1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0.1 | 0 | 0 | 0.1 | 0 | 0 | 0.1 | 0 | 1 | +0.1 | 0.2 | 0 | 0.1 |
| 1 | 1 | 0 | 1 | 0.2 | 0 | 0.1 | 0.2 | 0 | 0 | 0.2 | 0 | 1 | +0.1 | 0.3 | 0.1 | 0.1 |
| 1 | 1 | 1 | 0 | 0.3 | 0.1 | 0.1 | 0.3 | 0.1 | 0.1 | 0.5 | 0 | 0 | 0 | 0.3 | 0.1 | 0.1 |
| 1 | 0 | 0 | 1 | 0.3 | 0.1 | 0.1 | 0.3 | 0 | 0 | 0.3 | 0 | 1 | +0.1 | 0.4 | 0.1 | 0.1 |
| 1 | 0 | 1 | 1 | 0.4 | 0.1 | 0.1 | 0.4 | 0 | 0.1 | 0.5 | 0 | 1 | +0.1 | 0.5 | 0.1 | 0.2 |
| 1 | 1 | 0 | 1 | 0.5 | 0.1 | 0.2 | 0.5 | 0.1 | 0 | 0.6 | 1 | 0 | 0 | 0.5 | 0.1 | 0.2 |
| 1 | 1 | 1 | 0 | 0.5 | 0.1 | 0.2 | 0.5 | 0.1 | 0.2 | 0.8 | 1 | -1 | -0.1 | 0.4 | 0 | 0.1 |
| 1 | 0 | 0 | 1 | 0.4 | 0 | 0.1 | 0.4 | 0 | 0 | 0.4 | 0 | 1 | +0.1 | 0.5 | 0 | 0.1 |
| 1 | 0 | 1 | 1 | 0.5 | 0 | 0.1 | 0.5 | 0 | 0.1 | 0.6 | 1 | 0 | 0 | 0.5 | 0 | 0.1 |
| 1 | 1 | 0 | 1 | 0.5 | 0 | 0.1 | 0.5 | 0 | 0 | 0.5 | 0 | 1 | +0.1 | 0.6 | 0.1 | 0.1 |
| 1 | 1 | 1 | 0 | 0.6 | 0.1 | 0.1 | 0.6 | 0.1 | 0.1 | 0.8 | 1 | -1 | -0.1 | 0.5 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0.5 | 0 | 0 | 0.5 | 0 | 0 | 0.5 | 0 | 1 | +0.1 | 0.6 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0.6 | 0 | 0 | 0.6 | 0 | 0 | 0.6 | 1 | 0 | 0 | 0.6 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0.6 | 0 | 0 | 0.6 | 0 | 0 | 0.6 | 1 | 0 | 0 | 0.6 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0.6 | 0 | 0 | 0.6 | 0 | 0 | 0.6 | 1 | -1 | -0.1 | 0.5 | -0.1 | -0.1 |
| 1 | 0 | 0 | 1 | 0.5 | -0.1 | -0.1 | 0.5 | 0 | 0 | 0.5 | 0 | 1 | +0.1 | 0.6 | -0.1 | -0.1 |
| 1 | 0 | 1 | 1 | 0.6 | -0.1 | -0.1 | 0.6 | 0 | -0.1 | 0.5 | 0 | 1 | +0.1 | 0.7 | -0.1 | 0 |
| 1 | 1 | 0 | 1 | 0.7 | -0.1 | 0 | 0.7 | -0.1 | 0 | 0.6 | 1 | 0 | 0 | 0.7 | -0.1 | 0 |
| 1 | 1 | 1 | 0 | 0.7 | -0.1 | 0 | 0.7 | -0.1 | 0 | 0.6 | 1 | -1 | -0.1 | 0.6 | -0.2 | -0.1 |
| 1 | 0 | 0 | 1 | 0.6 | -0.2 | -0.1 | 0.6 | 0 | 0 | 0.6 | 1 | 0 | 0 | 0.6 | -0.2 | -0.1 |
| 1 | 0 | 1 | 1 | 0.6 | -0.2 | -0.1 | 0.6 | 0 | -0.1 | 0.5 | 0 | 1 | +0.1 | 0.7 | -0.2 | 0 |
| 1 | 1 | 0 | 1 | 0.7 | -0.2 | 0 | 0.7 | -0.2 | 0 | 0.5 | 0 | 1 | +0.1 | 0.8 | -0.1 | 0 |
| 1 | 1 | 1 | 0 | 0.8 | -0.1 | 0 | 0.8 | -0.1 | 0 | 0.7 | 1 | -1 | -0.1 | 0.7 | -0.2 | -0.1 |
| 1 | 0 | 0 | 1 | 0.7 | -0.2 | -0.1 | 0.7 | 0 | 0 | 0.7 | 1 | 0 | 0 | 0.7 | -0.2 | -0.1 |
| 1 | 0 | 1 | 1 | 0.7 | -0.2 | -0.1 | 0.7 | 0 | -0.1 | 0.6 | 1 | 0 | 0 | 0.7 | -0.2 | -0.1 |
| 1 | 1 | 0 | 1 | 0.7 | -0.2 | -0.1 | 0.7 | -0.2 | 0 | 0.5 | 0 | 1 | +0.1 | 0.8 | -0.1 | -0.1 |
| 1 | 1 | 1 | 0 | 0.8 | -0.1 | -0.1 | 0.8 | -0.1 | -0.1 | 0.6 | 1 | -1 | -0.1 | 0.7 | -0.2 | -0.2 |
| 1 | 0 | 0 | 1 | 0.7 | -0.2 | -0.2 | 0.7 | 0 | 0 | 0.7 | 1 | 0 | 0 | 0.7 | -0.2 | -0.2 |
| 1 | 0 | 1 | 1 | 0.7 | -0.2 | -0.2 | 0.7 | 0 | -0.2 | 0.5 | 0 | 1 | +0.1 | 0.8 | -0.2 | -0.1 |
| **1** | **1** | **0** | **1** | **0.8** | **-0.2** | **-0.1** | **0.8** | **-0.2** | **0** | **0.6** | **1** | **0** | **0** | **0.8** | **-0.2** | **-0.1** |
| **1** | **1** | **1** | **0** | **0.8** | **-0.2** | **-0.1** | **0.8** | **-0.2** | **-0.1** | **0.5** | **0** | **0** | **0** | **0.8** | **-0.2** | **-0.1** |
| **1** | **0** | **0** | **1** | **0.8** | **-0.2** | **-0.1** | **0.8** | **0** | **0** | **0.8** | **1** | **0** | **0** | **0.8** | **-0.2** | **-0.1** |
| **1** | **0** | **1** | **1** | **0.8** | **-0.2** | **-0.1** | **0.8** | **0** | **-0.1** | **0.7** | **1** | **0** | **0** | **0.8** | **-0.2** | **-0.1** |

This example can be implemented in the following [Python](http://en.wikipedia.org/wiki/Python_%28programming_language%29) 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