Perceptron

Binary classification algorithm

Procheta Sen

UNIVERSITY OF LIVERPOOL

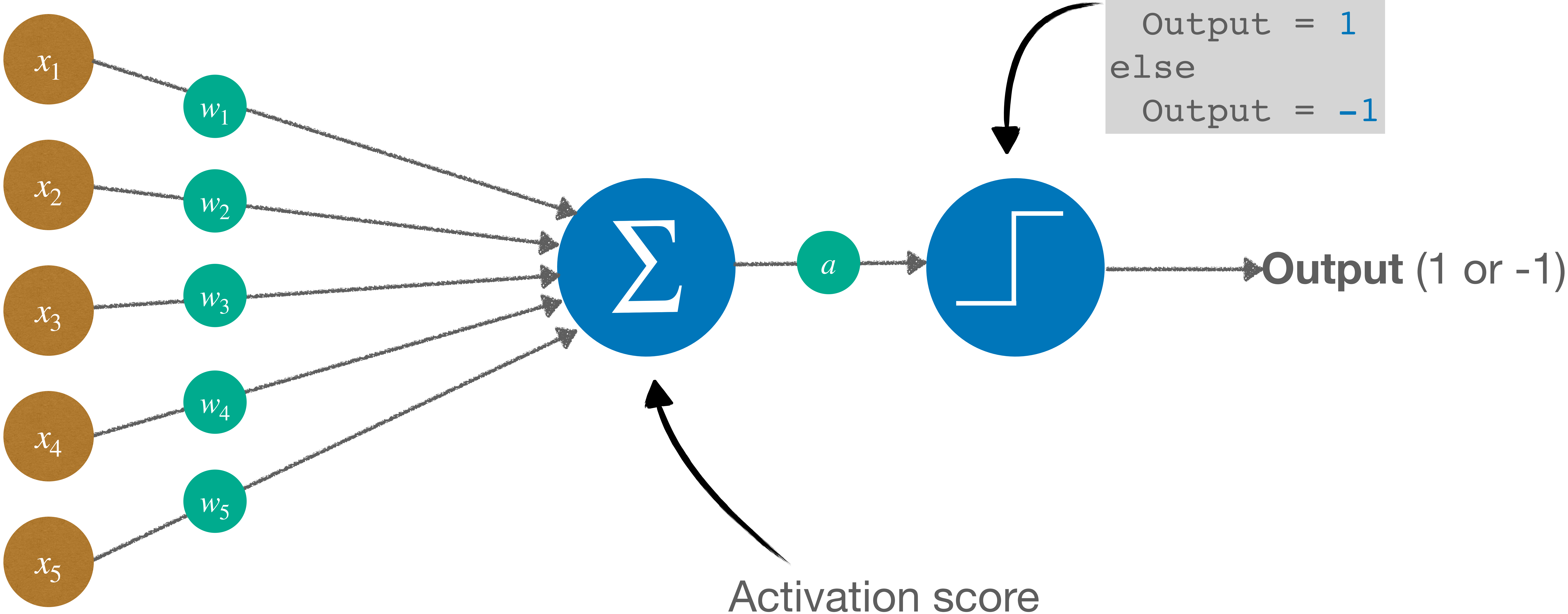
Bio-inspired model

- **Neural networks** are a model of simulation of the human nervous system
- Nervous system is composed of **nerve cells** (**neurons)**
- Neurons are connected to one another at contact points (synapses)
- Learning is done by changing the strength of **synaptic connections** between neurons
- The strength of the connections change in response to external stimuli
- **Perceptron** is a model of a **single neuron**

Perceptron

If the **score** is greater than a predefined threshold θ, then the neutron fires

```
If a > θ then
  Output = 1
else
  Output = -1
```



Activation score

$$a = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5$$

Perceptron

- The computation function at a neuron is defined by the **weights**
- The **weights** correspond to the **strengths of synaptic connections**
- The computation function is **learned** by appropriately changing the weights
- The “**external stimulus**” is provided by the **training data**
- Idea: **incrementally modify the weights** whenever **incorrect predictions** are made by the current set of weights

Mathematical notation

- Input object $\overline{X}^T = (x_1, x_2, \ldots, x_d)$
- Weights $\overline{W}^T = (w_1, w_2, \ldots, w_d)$

Activation score $a = \sum_{i=1}^{d} w_i x_i = \overline{W}^T \overline{X}$

- Output 1 if $a > \theta$, and
- Output -1 if $a \leq \theta$.

Bias

- It is convenient to make the threshold θ equal to 0
- This is achieved by introducing a bias term $b = -\theta$

$$a = b + \sum_{i=1}^{d} w_i x_i$$

- Output 1 if $a > 0$, and
- Output -1 if $a \leq 0$
- Equivalently, output sign$(\overline{W}^T X + b)$

Notational trick

- By introducing a feature x_0 that is always ON (i.e., $x_0 = 1$ for all objects), we can squeeze the bias term b into the weight vector by setting $w_0 = b$

$$a = \sum_{i=0}^{d} w_i x_i = \overline{W}^T \overline{X}$$

This is more "elegant" as we can write the activation as the inner-product between the weight vector and the feature vector. However, we should keep in mind that bias term still appears in the model.

The training algorithm

PerceptronTrain(Training data: D, MaxIter)

$w_i = 0$ for all $i = 1, \ldots, d$;

$b = 0$

for iter = 1 … MaxIter **do**

  for all $(\overline{X}, y) \in D$ **do**

    $a = \overline{W}^T \overline{X} + b$

    if $y \cdot a \leq 0$ **then**

      $w_i = w_i + y \cdot x_i$, for all $i = 1, \ldots, d$

      $b = b + y$

return $b, w_1, w_2, \ldots, w_d$

Initialize weights and bias

Compute activation score

Update weights an bias

The test algorithm

`PerceptronTest`$(b, w_1, w_2, \ldots, w_d, \overline{X})$

1: $a = \overline{W}^T \overline{X} + b$

2: **`return`** $\text{sign}(a)$

Important features of Perceptron

- **Online algorithm**: processes objects from the training data set one by one (as opposed to **batch learning** that requires access to the entire data set, e.g. k-NN)

- **Error driven**: the parameters are updated **only** when a test object is classified wrongly using the current parameters (weights and bias)

Detecting misclassification (incorrect predictions)

`PerceptronTrain`(Training data: `D`, `MaxIter`)

$w_i = 0$ `for all` $i = 1, \ldots, d$;

$b = 0$

for `iter = 1 … MaxIter` **do**

  for `all` $(\overline{X}, y) \in D$ **do**

    $a = \overline{W}^T \overline{X} + b$

    if $y \cdot a \leq 0$ **then**

      $w_i = w_i + y \cdot x_i$, `for all` $i = 1, \ldots, d$

      $b = b + y$

return $b, w_1, w_2, \ldots, w_d$

Predicted class (sign(a)) is different from the current instance class (y) if and only if $y \cdot a \leq 0$

Update rule — Intuitive Explanation

PerceptronTrain(Training data: D, MaxIter)

$w_i = 0$ `for all` $i = 1, \ldots, d$;

$b = 0$

for `iter = 1 … MaxIter` **do**

  for `all` $(\overline{X}, y) \in D$ **do**

    $a = \overline{W}^T \overline{X} + b$

    if $y \cdot a \leq 0$ **then**

      $w_i = w_i + y \cdot x_i$, `for all` $i = 1, \ldots, d$

      $b = b + y$

return $b, w_1, w_2, \ldots, w_d$

Perceptron update rule

$$\overline{W} = \overline{W} + y\overline{X}$$

$$b = b + y$$

- If we incorrectly classify a **positive** instance as **negative**
  - We should have a **higher activation** to avoid this
  - We **increase** $\overline{W}^T\overline{X}$ and b by **adding** the current instance to the weight vector and the bias
- If we incorrectly classify a negative instance as positive
  - We should have a lower **activation** to avoid this
  - We **decrease** $\overline{W}^T\overline{X}$ and b by **deducting** the current instance from the weight vector and the bias

Update rule — Math Explanation

`PerceptronTrain`(Training data: `D`, `MaxIter`)

$w_i = 0$ `for all` $i = 1, \ldots, d$;

$b = 0$

`for` `iter = 1 … MaxIter` **do**

  `for` `all` $(\overline{X}, y) \in D$ **do**

    $a = \overline{W}^T \overline{X} + b$

    `if` $y \cdot a \leq 0$ **`then`**

      $w_i = w_i + y \cdot x_i$, `for all` $i = 1, \ldots, d$

      $b = b + y$

`return` $b, w_1, w_2, \ldots, w_d$

- Current parameters: $b, w_1, w_2, \ldots, w_d$
- Incoming object from the test data: $(\overline{X}, y)$
- Suppose $y = +1$ and $a \leq 0$ (**misclassification**)
- New parameters: $b', w_1', w_2', \ldots, w_d'$

$$
\begin{aligned}
a' &= \sum_{i=1}^{d} w_i' x_i + b' \\
&= \sum_{i=1}^{d} (w_i + x_i) \cdot x_i + (b + 1) \\
&= \sum_{i=1}^{d} w_i x_i + b + \sum_{i=1}^{d} x_i x_i + 1 = a + \sum_{i=1}^{d} x_i^2 + 1 > a
\end{aligned}
$$

Remark: activation adjustment

PerceptronTrain(Training data: D, MaxIter)

1: $w_i = 0$ `for all` $i = 1, \ldots, d$;

2: $b = 0$

3: **for** `iter = 1 … MaxIter` **do**

4: **for** `all` $(\overline{X}, y) \in D$ **do**

5: $a = \overline{W}^T \overline{X} + b$

6: **if** $y \cdot a \leq 0$ **then**

7: $w_i = w_i + y \cdot x_i$, `for all` $i = 1, \ldots, d$

8: $b = b + y$

9: **return** $b, w_1, w_2, \ldots, w_d$

- There is no guarantee that we will correctly classify a misclassified instance in the next round.
- We have simply increased/decreased the activation but this adjustment might not be sufficient. We might need to do more aggressive adjustments.
- There are algorithms that enforce such requirements explicitly such as the Passive Aggressive Classifier (not discussed here)

Remark: ordering of training instances

PerceptronTrain(Training data: D, MaxIter)

$w_i = 0$ `for all` $i = 1, \ldots, d$;

$b = 0$

for `iter = 1 … MaxIter` **do**

  for `all` $(\overline{X}, y) \in D$ **do**

    $a = \overline{W}^T \overline{X} + b$

    if $y \cdot a \leq 0$ **then**

      $w_i = w_i + y \cdot x_i$, `for all` $i = 1, \ldots, d$

      $b = b + y$

return $b, w_1, w_2, \ldots, w_d$

The order of the test instances matters

- Showing only all the positives first and all the negatives next is a bad idea
- Ordering training instances randomly within each iteration produces good results in practice

Remark: hyperparameter and overfitting

`PerceptronTrain`(Training data: `D`, `MaxIter`)

$w_i = 0$ `for all` $i = 1, \ldots, d$;

$b = 0$

for `iter = 1 … MaxIter` **do**

    for `all` $(\overline{X}, y) \in D$ **do**

        $a = \overline{W}^T \overline{X} + b$

        if $y \cdot a \leq 0$ **then**

            $w_i = w_i + y \cdot x_i$, `for all` $i = 1, \ldots, d$

            $b = b + y$

return $b, w_1, w_2, \ldots, w_d$

`MaxIter` is a **hyperparameter** which has to be chosen experimentally

- If we make many passes over the training data, then the algorithm is likely to **overfit**.
- If we make few passes might lead to **underfitting**.