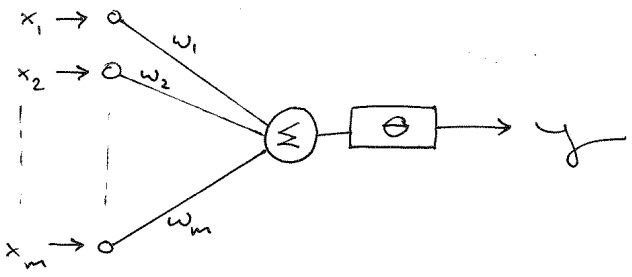
COMS 4030A

Adaptive Computation and Machine Learning

1. Perceptrons

1.1. Artificial Neurons.

An **artificial neuron** has the following structure:



The values $w_1, \ldots, w_m$ are real numbers called the **weights**.
The value of θ is a real number called the **threshold**.

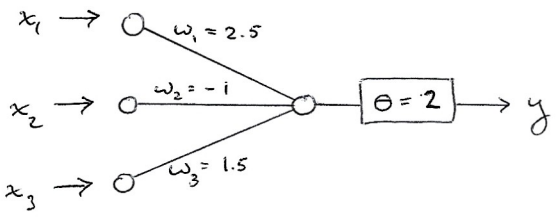
Given input values $x_1, \ldots, x_m$, where each $x_i \in \mathbb{R}$, the output y is obtained as follows:

First calculate the sum: $\quad h = \sum_{i=1}^{m} x_i w_i.$

Then, compare the sum h with the **threshold** value θ:

if $h > \theta$ then the output value is $y = 1$; (we say the neuron **fires** or **activates**)
if $h \leq \theta$ then the output value is $y = 0$.

Example: Consider the following 3-input artificial neuron:



Suppose the following values are input to neuron: $x_1 = 1$, $x_2 = 3$, $x_3 = 2$, i.e., the input vector is $\boldsymbol{x} = (1, 3, 2)$. The corresponding output is obtained as follows:

$$\sum_{i=1}^{3} x_i w_i = 1(2.5) + 3(-1) + 2(1.5) = 2.5 \quad \text{and} \quad 2.5 > 2, \quad \text{so } y = 1.$$

If the input vector is $\boldsymbol{x} = (1.5, 1, -1)$ the corresponding output is obtained as follows:

$$\sum_{i=1}^{3} x_i w_i = (1.5)(2.5) + 1(-1) + (-1)(1.5) = 1.25 \quad \text{and} \quad 1.25 \leq 2, \quad \text{so } y = 0.$$

A function made up of a single artificial neuron is called a **perceptron**.

1.2. **Linear Discriminants.**

Suppose we have a dataset D consisting of points in $\mathbb{R}^m$ such that each point in the dataset is classified into one of two possible classes. We use the labels 0 and 1 to denote the two classes.

By a **linear discriminant** for a dataset we mean a function of the form

$$f(x_1, x_2, \ldots, x_m) = a_1 x_1 + a_2 x_2 + \cdots + a_m x_m + b$$

such that for each data point $\boldsymbol{x} = (x_1, \ldots, x_m)$ in the dataset

$$f(x_1, x_2, \ldots, x_m) > 0 \text{ if, and only if, } \boldsymbol{x} \text{ has classification 1,}$$

$$f(x_1, x_2, \ldots, x_m) \leq 0 \text{ if, and only if, } \boldsymbol{x} \text{ has classification 0.}$$

Example: Suppose we have the following dataset:

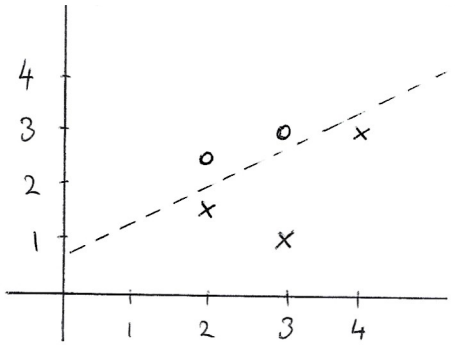
$$D = \{((3,1),1), ((2,2.5),0), ((2,1.5),1), ((4,3),1), ((3,3),0)\},$$

which we can also represent by a table:

$$X = \begin{bmatrix} 3 & 1 \\ 2 & 2.5 \\ 2 & 1.5 \\ 4 & 3 \\ 3 & 3 \end{bmatrix} \qquad T = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

The values in X are input values and values in T are the corresponding classifications.
The goal is to find a function that correctly classifies each input in the dataset.
This is an example of a **classification** problem.

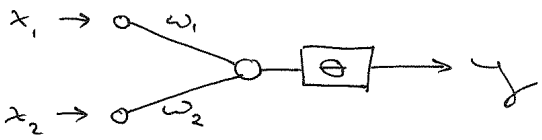
The diagram below depicts all the datapoints in D together with their classifications, where a $\circ$ indicates class 0 and a $\times$ indicates class 1.
We will try to find a straight line that separates the 1's from the 0's, i.e., a linear discriminant, as shown by the dotted line in the diagram below.



Once we have found a linear discriminant we can use it to predict the classification of other inputs. For example, we would predict that the input $(3, 2)$ should be classified as a 1.

Next, we show how an m-input perceptron is a linear discriminant in $\mathbb{R}^m$, and later we show how to train a perceptron as a linear discriminant for a given dataset.

First, consider a 2-input perceptron:



For inputs $\boldsymbol{x} = (x_1, x_2)$ the output y is calculated as follows:

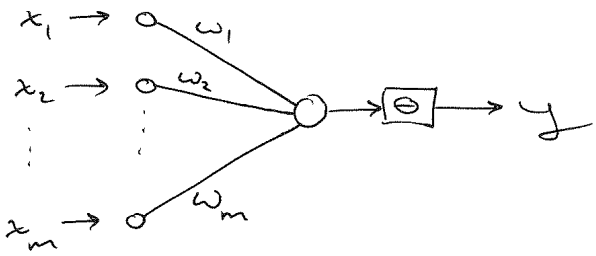
if $x_1w_1 + x_2w_2 > \theta$, then $y = 1$;
if $x_1w_1 + x_2w_2 \leq \theta$, then $y = 0$.

Another way to write this is:
if $x_1w_1 + x_2w_2 - \theta > 0$, then $y = 1$;
if $x_1w_1 + x_2w_2 - \theta \leq 0$, then $y = 0$.

Thus, the straight line with equation: $x_1w_1+x_2w_2-\theta = 0$ (where w_1, w_2 and θ are constants) separates inputs with output $y = 1$ from those with output $y = 0$.
The function $f(x_1, x_2) = x_1w_1 + x_2w_2 - \theta$ is the linear discriminant.

Thus, a 2-input perceptron gives a straight line, which is a linear discriminant in $\mathbb{R}^2$.
Similarly, a 3-input perceptron gives a plane, which is a linear discriminant in $\mathbb{R}^3$.

In general, an m-input perceptron gives an m-**hyperplane**, a linear discriminant in $\mathbb{R}^m$.
Suppose we have a neuron with m inputs as follows:



Then the neuron fires, i.e., $y = 1$, if $\sum_{i=1}^{m} x_i w_i > \theta$.
The linear discriminant associated with this perceptron is $f(x_1, \ldots, x_m) = \sum_{i=1}^{m} x_i w_i - \theta$.

1.3. **Training a Perceptron.**

Suppose we are given a dataset D containing N data points of the form $(\boldsymbol{x}, t)$, where each $\boldsymbol{x}$ is in $\mathbb{R}^m$, and each t is either 1 or 0.

The following method is used to train the weights and threshold of a perceptron so that is correctly predicts the output for a given input value.

Perceptron Training Algorithm

Create an m-input perceptron by randomly choosing weights $w_1, \ldots, w_m$
and randomly choosing a threshold value θ.
(Usually, the random values are chosen from a fixed range such as $[-1, 1]$.)

Choose a **learning rate** η, where $0 < \eta < 1$.
(Usually, we choose η between 0.001 and 0.1.)

while (stopping condition is not satisfied)
  for each data point $(\boldsymbol{x}, t)$ in the set D, do the following:
    feed $\boldsymbol{x}$ into the perceptron, to get output y
    update each weight w_i using the rule: $w_i \leftarrow w_i + \eta(t - y)x_i$
    update the threshold θ using the rule: $\theta \leftarrow \theta - \eta(t - y)$

The final weights obtained by the PERCEPTRON TRAINING ALGORITHM give us the trained perceptron. It should be the case that for 'most of' the training data, the output from the perceptron matches the target output, i.e., if $(\boldsymbol{x}, t)$ is in the dataset and $\boldsymbol{x}$ is input into the perceptron, then the output y should be the same as the target t.

One complete execution of the above **for** loop (over all data points in D) is called an **epoch**. During an epoch, the order in which the data points are input can make a difference for learning the weights since the last data points have more impact than the first ones.
A good idea is to randomize the order in which the data is input at the start of each epoch.

1.4. **Stopping Conditions.**

We can stop the algorithm if there were no changes to any weights during an epoch. Alternately, since this may not happen, we can stop after some maximum number of epochs or after some specified time has elapsed.

Another way to determine if the perceptron has trained for long enough is to calculate the **loss** (or **error**) of the perceptron on the dataset.
Suppose we have a data set $D = \{(\boldsymbol{x}_k, t_k) \mid 1 \leq k \leq N\}$, where each $\boldsymbol{x}_k \in \mathbb{R}^m$ and each $t_k \in \{0, 1\}$, and we have trained a perceptron on this dataset. For each data point $(\boldsymbol{x}_k, t_k)$ in D, feed the input $\boldsymbol{x}_k$ into the perceptron to get the output y_k and calculate the loss L as follows:

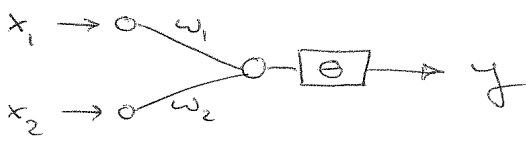
$$L = \sum_{k=1}^{N} |t_k - y_k|.$$

The above sum counts the number of times the perceptron misclassifies an input. If L gets to 0, then the perceptron gives the correct output for every input so you can stop training. Since this may not happen, you may need to stop when L gets below some fixed value. Since even this is not guaranteed, one can stop when the change in the error between epochs, i.e. ΔL, is close to 0. This means that the perceptron has stopped improving itself.

It is useful to keep a record of the L values and plot the value of L against the number of epochs to see how it changes with learning. Note that calculating L requires computing time, so on a large dataset one could calculate it only after a fixed number of epochs.

1.5. **Where does the perceptron learning rule come from?**

To understand this, consider a 2-input perceptron:



Recall that for any input $\boldsymbol{x} = (x_1, x_2)$, we first calculate $h = x_1 w_1 + x_2 w_2$.
Then, the output is $y = 1$ if $h > \theta$, and $y = 0$ if $h \leq \theta$.

Observe that h is the dot-product of the vectors $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{w} = (w_1, w_2)$,
that is, $h = x_1 w_1 + x_2 w_2 = \boldsymbol{x} \cdot \boldsymbol{w}$.

Thus, $y = 1$ if $\boldsymbol{x} \cdot \boldsymbol{w} > \theta$, and $y = 0$ if $\boldsymbol{x} \cdot \boldsymbol{w} \leq \theta$.

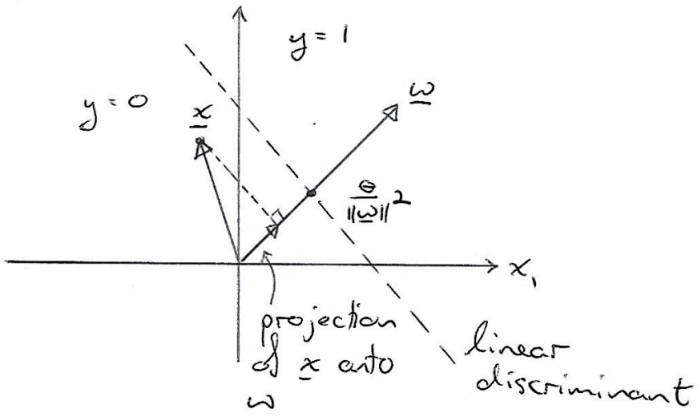
Plot vectors $\boldsymbol{x}$ and $\boldsymbol{w}$ in $\mathbb{R}^2$, and calculate the **projection** of $\boldsymbol{x}$ onto $\boldsymbol{w}$, which is the vector

$$\frac{\boldsymbol{x} \cdot \boldsymbol{w}}{||\boldsymbol{w}||^2}\, \boldsymbol{w}, \quad \text{where } ||\boldsymbol{w}|| = \sqrt{w_1^2 + w_2^2}.$$

Note that $\boldsymbol{x} \cdot \boldsymbol{w} > \theta$ iff $\dfrac{\boldsymbol{x} \cdot \boldsymbol{w}}{||\boldsymbol{w}||^2} > \dfrac{\theta}{||\boldsymbol{w}||^2}$,

so $y = 1$ if the projection of $\boldsymbol{x}$ onto $\boldsymbol{w}$ lies above the point $\dfrac{\theta}{||\boldsymbol{w}||^2}$ along $\boldsymbol{w}$,
and $y = 0$ if the projection lies below that point.

Thus, the linear discriminant given by the perceptron is the line perpendicular to $\boldsymbol{w}$
at the point $\dfrac{\theta}{||\boldsymbol{w}||^2}$ along $\boldsymbol{w}$, as shown in the diagram below.



To return to the perceptron training rule, suppose we have a dataset D of points $(\boldsymbol{x}, t)$, where $\boldsymbol{x} \in \mathbb{R}^2$ and $t \in \{0, 1\}$. Suppose we have some existing values for w_1, w_2 and θ and we want to train these weights to match the data point $(\boldsymbol{x}, t)$.

Suppose $\boldsymbol{x}$ has target $t = 1$, but when fed through the perceptron it outputs $y = 0$.

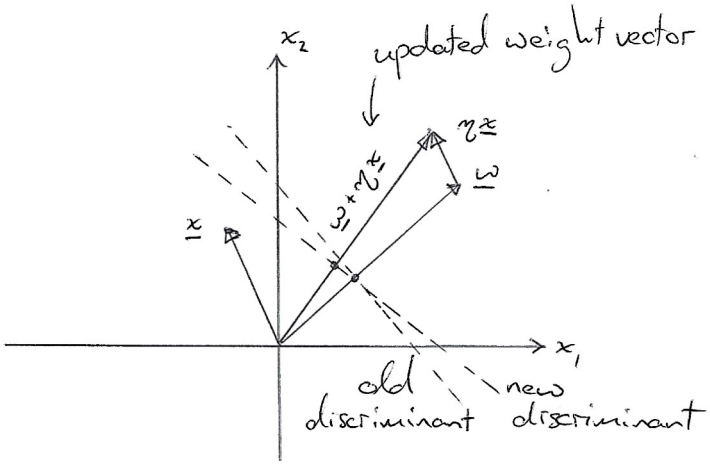
This means that $\dfrac{\boldsymbol{x} \cdot \boldsymbol{w}}{||\boldsymbol{w}||^2} \leq \dfrac{\theta}{||\boldsymbol{w}||^2}$.

We want to change the weights w_i so that $\dfrac{\boldsymbol{x} \cdot \boldsymbol{w}}{||\boldsymbol{w}||^2} > \dfrac{\theta}{||\boldsymbol{w}||^2}$.

We do this by moving the vector $\boldsymbol{w}$ incrementally in the direction of $\boldsymbol{x}$, i.e.,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{x}.$$

This is illustrated in the diagram below. (Recall that $0 < \eta < 1$.)



Since we only want to change the weights if $\boldsymbol{x}$ is misclassified, we add the term $(t - y)$, which gives 0 if the target t equals the output y:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(t - y)\boldsymbol{x}.$$

If $t = 1$ and $y = 0$, then $\boldsymbol{w}$ will move towards $\boldsymbol{x}$.
If $t = 0$ and $y = 1$, then we get $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{x}$, i.e., the weight vector moves away from $\boldsymbol{x}$.

The learning rule for w_1 and w_2 is given by:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \leftarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \eta(t - y) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

which we can write as

$$w_i \leftarrow w_i + \eta(t - y)x_i, \quad \text{for each } i.$$

Next, consider the case of θ.
Suppose that input $\boldsymbol{x}$ has target $t = 1$ but the perceptron gives output $y = 0$.
The value of θ is changed using the update rule:

$$\theta \leftarrow \theta - \eta.$$

By making this change, the point $\frac{\theta}{||\boldsymbol{w}||^2}$ along $\boldsymbol{w}$ that determines the linear discriminant moves downwards to try include $\boldsymbol{x}$ on the correct side of the line.

In the case where $t = 0$ and $y = 1$ we use the update rule:

$$\theta \leftarrow \theta + \eta.$$

Then the point $\frac{\theta}{||\boldsymbol{w}||^2}$ along $\boldsymbol{w}$ that determines the linear discriminant moves upwards to try include $\boldsymbol{x}$ on the correct side of the line.

We can describe both rules using one rule as follows: $\theta \leftarrow \theta - \eta(t - y)$.

EXERCISES

(1) Consider a 3-input perceptron with weights $w_1 = 2.5$, $w_2 = -3$, $w_3 = 1.5$ and threshold $\theta = 2$.
Find the output of the perceptron for the following input vectors:
(a) $\boldsymbol{x} = (-1, 2, 4)$
(b) $\boldsymbol{x} = (2, -1, -2)$

(2) Using the perceptron in exercise (1), do one update of the weights w_1, w_2, w_3 and threshold θ as follows: Suppose the input vector $\boldsymbol{x} = (1, 1, 2)$ from your dataset has target $t = 0$. When $\boldsymbol{x}$ is fed into the perceptron the output we get is $y = 1$, which differs from the target t. Update all the weights and threshold using the rules in the PERCEPTRON TRAINING ALGORITHM. Use $\eta = 0.1$.