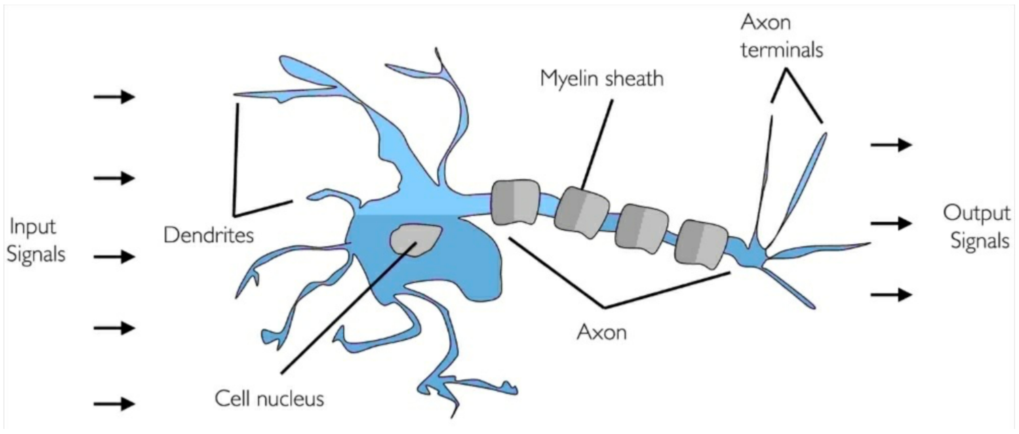
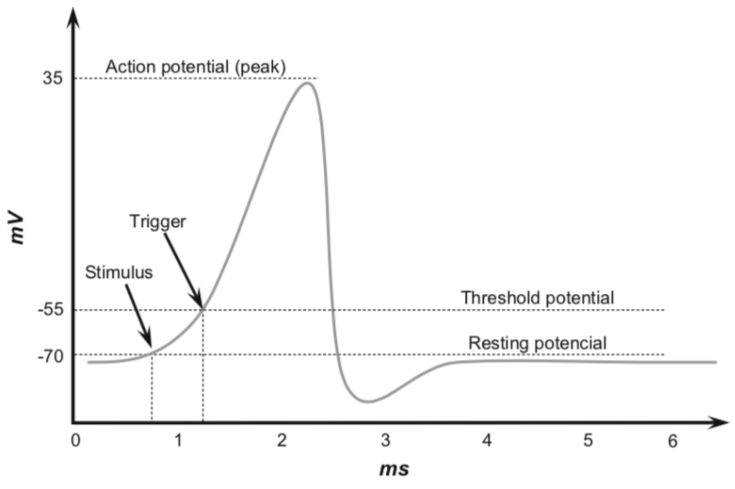
Perceptron

The Perceptron network is one of the simplest ANN architectures, invented in 1957 by Frank Rosenblatt. The Perceptron network is inspired by the biological neuron shown below. Neurons are interconnected nerve cells in the brain that are involved in the processing and transmitting of chemical and electrical signals, which is illustrated in the following figure:

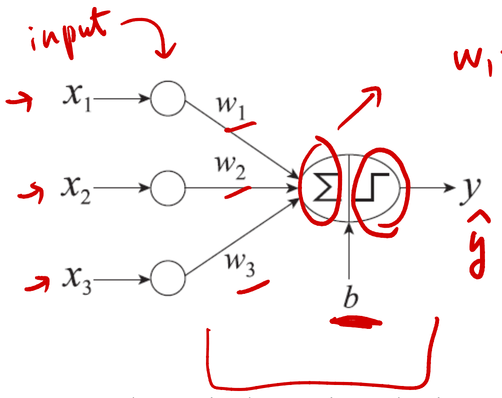


When the nervous cell is stimulated with an impulse higher than its activation threshold (−55 mV), caused by the variation of internal concentrations of sodium (Na+) and potassium (K+) ions, it triggers an electrical impulse which will propagate throughout its axon with a maximum amplitude of 35 mV. The stages related to variations of the action voltage within a neuron during its excitation are shown in the Figure below.



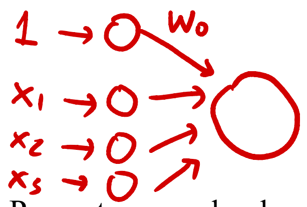
2. Perceptron: The Basics

The Perceptron network consists of one artificial neuron. Figure below illustrates a Perceptron network composed of *3* input signals, representing the problem being analyzed, and just one output.



$w_1x_1 + w_2x_2 + w_3x_3 + \underline{b} \mapsto$ activate or not

$y = f(x)$



In mathematical notation, the inner processing performed by the Perceptron can be described by the following expressions:

$$\begin{cases} \underline{z} = w_1x_1 + w_2x_2 + w_3x_3 + b \\ y = \underline{\theta}(z) \end{cases}$$

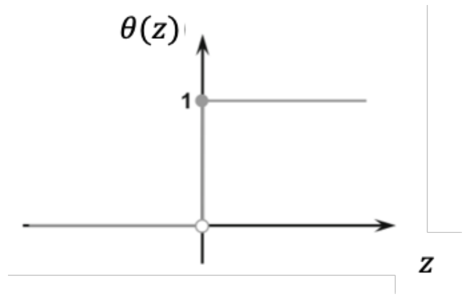
where x_i is a network input, w_i is the weight associated with the i^{th} input, b is the activation threshold (or bias), $\theta(.)$ is the activation function and z is the activation potential or sometimes called input signal.

The most common activation functions used in Perceptron are the step and bipolar step functions:

Step Function

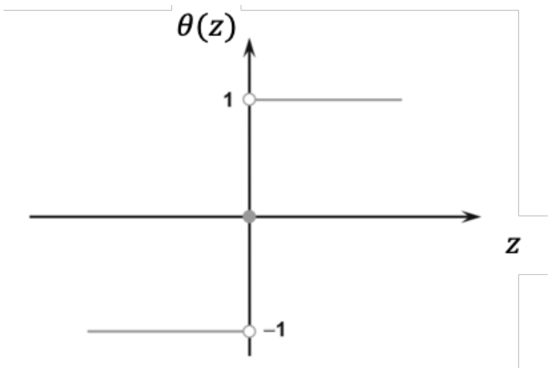
hard ↘ soft

$$\theta(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$



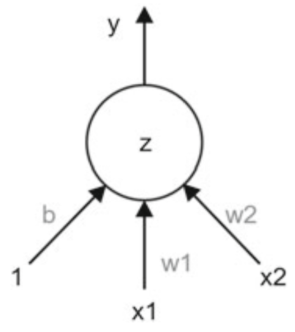
<u>Bipolar Step Function</u>

$$\theta(z) = \begin{cases} -1 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$



Q: What are the possible output values possibly produced by the Perceptron?
A: Only two values can be produced by the network output, that is, 0 or 1 for the step function, and −1 or 1 if the bipolar step function is used.

Remark on Bias Absorption: We note that it is possible to absorb the bias as one of the weights, so that we only need **a weight update rule**. This is displayed in the figure below: to absorb the bias as a weight, one needs to add an input x_0 with value 1 and **the bias is its weight**.



$$\begin{cases} z = b + \sum_{i=1}^{d} w_i x_i = \sum_{i=0}^{d} w_i x_i = \mathbf{w}^{\mathrm{T}}\mathbf{x} \\ y = \theta(z) \end{cases}$$

in which $\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$ and $\mathbf{x} = [1, x_1, \ldots, x_d]^T$ are column vectors.

Fun Time: Can perceptron do (1) linear classification (2) linear regression (3) nonlinear classification (4) nonlinear regression?

The Perceptron network can be used for simple linear binary classification. It computes a linear combination of the inputs and if the result exceeds a threshold, it outputs the positive class or else outputs the negative class. Training a Perceptron network means finding the right values for the weights $\mathbf{w} = [w_0, w_1, \dots, w_d]^T$.

3. Training Process of the Perceptron

The adjustment of Perceptron's weights $\mathbf{w}$, in order to classify patterns that belong to one of the two possible classes, is performed by the use of Gradient Decent. For the bipolar step function, the Perceptron can be written in a vector form as:

$$\hat{y} = h(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

in which $\mathbf{w} = [w_0, w_1, \dots, w_d]^T$ and $\mathbf{x} = [1, x_1, \dots, x_d]^T$ are column vectors. Given a training set, we are interested in learning parameters $\mathbf{w}$ such that $\hat{y}$ closely resembles the true y of training instances. This is achieved by using the perceptron learning algorithm given below:

```
1: Let $D = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \dots, N\}$ be the set of training examples.
2: Initialize the weight vector with random values, $\mathbf{w}^{(0)}$        $\cancel{w_0}$
3: repeat
4:   for each training example $(\mathbf{x}_i, y_i) \in D$ do
5:     Compute the predicted output $\hat{y}_i^{(k)}$
6:     for each weight $w_j$ do
7:       Update the weight, $w_j^{(k+1)} = w_j^{(k)} + \lambda\big(y_i - \hat{y}_i^{(k)}\big)x_{ij}$.
8:     end for
9:   end for
10: until stopping condition is met
```

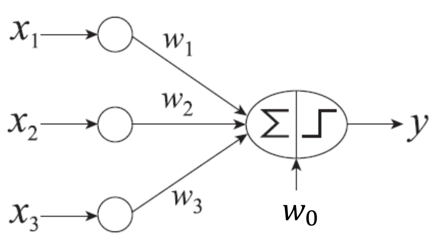
The key computation for this algorithm is the weight update formula given in Step 7 of the algorithm:

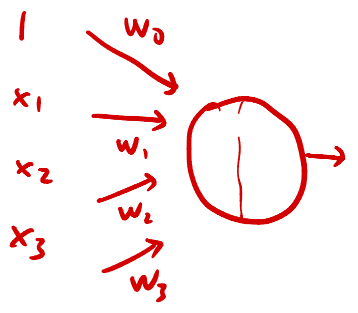
$$w_j^{(k+1)} = w_j^{(k)} + \lambda\left(y_i - \hat{y}_i^{(k)}\right)x_{ij}$$

where $w_j^{(k)}$ is the weight parameter associated with the j^{th} attribute after the k^{th} iteration, λ is a parameter known as **the learning rate**, and x_{ij} is the value of the j^{th} attribute of the training example x_i.

Example

Consider a perceptron with a bipolar step function:





$$\theta(z) = \begin{cases} -1 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

The table below shows a dataset containing eight training examples with three Boolean variables (x_1, x_2, x_3) and an output variable, y. The output takes on the value -1 if at least two of the three inputs are zero, and $+1$ if at least two of the inputs are greater than zero.

X$_1$	X$_2$	X$_3$	y
1	0	0	–1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	–1
0	1	0	–1
0	1	1	1
0	0	0	–1

$i=3 \rightarrow$

$\uparrow$ $j=2$

$\rightarrow \hat{y} = 1$

$w_0^{(1)} = w_0^{(0)} + 0.1 \times (-2) = -0.2$

$w_1^{(1)} = w_1^{(0)} + 0.1 \times (-2) \cdot 1 = -0.2$

$$w_j^{(k+1)} = w_j^{(k)} + \lambda \left(y_i - \hat{y}_i \right) x_{ij}$$

Let us train the network by sequence with each training example. We will call it an epoch when we go through all the examples. Note that there can be multiple decision boundaries that can separate the two classes, and **the perceptron arbitrarily learns one of these boundaries depending on the random initial values of parameters**. (SVM takes care of the selection of the optimal decision boundary)

Let us start from $\mathbf{w}^{\text{init}} = [w_0 = 0, w_1 = 0, w_2 = 0, w_3 = 0]^T$ with learning rate $\lambda = 0.1$

Example 1, Epoch 1

Fun Time: What are the updated values for w_0 and w_1? (1) $[0, 0]$ (2) $[1, 1]$ (3) $[-0.2, 0.2]$ (4) $[-0.2, -0.2]$ (5) $[-0.2, 0.0]$?

We summarize weight updates after the first epoch and weight updates after a few epochs.

$\lambda = 0.1$ Epoch = 1

	w_0	w_1	w_2	w_3
	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.2
5	-0.4	0.4	0.4	0.2
6	-0.4	0.4	0.4	0.2
7	-0.4	0.4	0.4	0.2
8	-0.4	0.4	0.4	0.2
9	-0.4	0.4	0.4	0.2
10	-0.4	0.4	0.4	0.2

As we mentioned, **the perceptron arbitrarily learns one of these boundaries depending on the random initial values of parameters**. Let us try different initial values:

Let us start from $\mathbf{w}^{\text{init}} = [w_0 = 0.5, w_1 = 0.5, w_2 = 0.5, w_3 = 0.5]^T$ with learning rate $\lambda = 0.1$

Again, we summarize weight updates after a few epochs.

Epoch	w_0	w_1	w_2	w_3
0	0.5	0.5	0.5	0.5
1	-0.1	0.3	0.3	0.3

2	-0.3	0.1	0.3	0.3
3	-0.3	0.1	0.3	0.3
4	-0.3	0.1	0.3	0.3
5	-0.3	0.1	0.3	0.3
6	-0.3	0.1	0.3	0.3

Remarks:

1. For $\mathbf{w}^{\text{init}} = [w_0 = 0, w_1 = 0, w_2 = 0, w_3 = 0]^T$ with learning rate $\lambda = 0.1$, the final weights for the perceptron model are $\mathbf{w} = [w_0 = -0.4, w_1 = 0.4, w_2 = 0.4, w_3 = 0.2]^T$
2. For $\mathbf{w}^{\text{init}} = [w_0 = 0.5, w_1 = 0.5, w_2 = 0.5, w_3 = 0.5]^T$ with learning rate $\lambda = 0.1$, the final weights for the perceptron model are $\mathbf{w} = [w_0 = -0.3, w_1 = 0.1, w_2 = 0.3, w_3 = 0.3]^T$
3. The way this optimization algorithm works is that each training instance is shown to the model one at a time. The model makes a prediction for a training instance, the error is calculated and the model is updated in order to reduce the error for the next prediction. This process is repeated for a fixed number of iterations. This is called Gradient Descent. If we feed the training instance randomly to the model, we called the optimization algorithm **Stochastic Gradient Descent**.

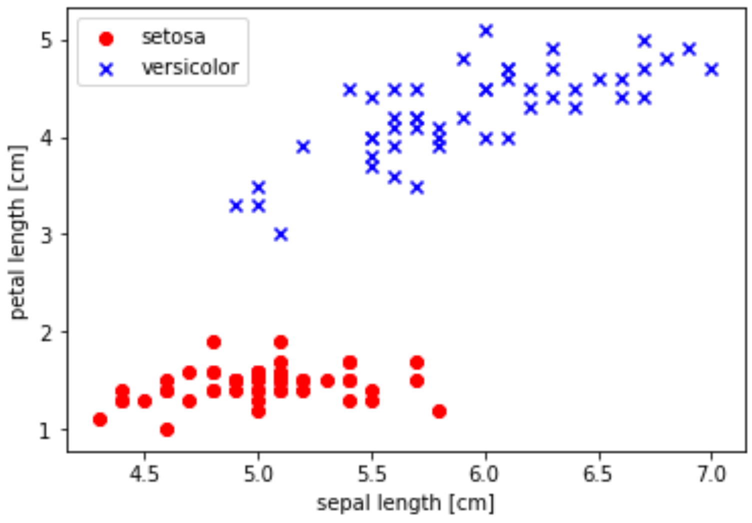
Python Example: The `scikit-learn` has `Perceptron` class implemented in the `linear_model` module. Let us use parts of `Iris` dataset to demonstrate its simple usage for binary classification. Our goal is to use `sepal` and `petal` lengths to classify the flowers for `Setosa` or `Versicolor`:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split

iris = load_iris()
X = iris.data[0:100, (0, 2)] # sepal length, petal length
y = iris.target[0:100] # Setosa or Versicolor
```

```
# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x', label='versicolor')

plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
```



```
per_clf = Perceptron(random_state=42) #default learning rate = 1.0
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)
per_clf.fit(X_train,y_train)

y_pred = per_clf.predict(X_test)
print('Misclassified samples: %d' % (y_test != y_pred).sum())
```

```
Misclassified samples: 0
```

The threshold and weights can be found via:

```
print ("threshold: ", per_clf.intercept_)
print ("w1: ", per_clf.coef_[0,0])
print ("w2: ", per_clf.coef_[0,1])
```

```
threshold:  [-0.01]
w1:  -0.027
w2:  0.052
```

You can download the above Python codes `Perceptron.ipynb` from the course website.