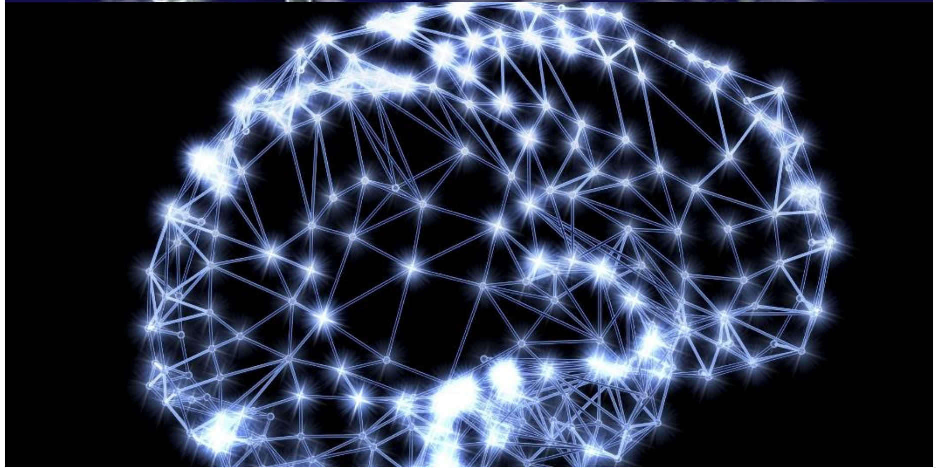
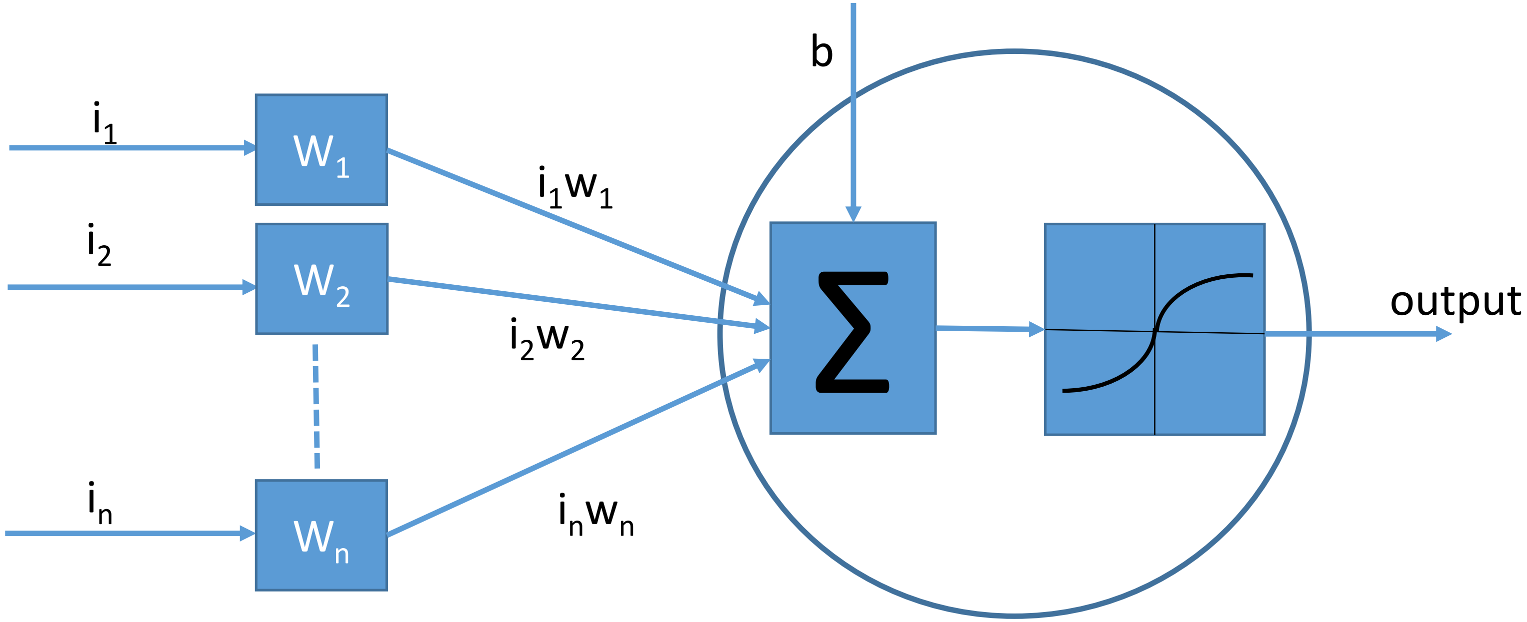
Artificial Neural Networks (ANN)

- ANNs inspired by biological neural systems.
- Human brain consists of nerve cells called neurons
- According to neurologists the human brain learns by changing the strength of the synaptic connection between neurons upon *repeated stimulation by the same impulse*.

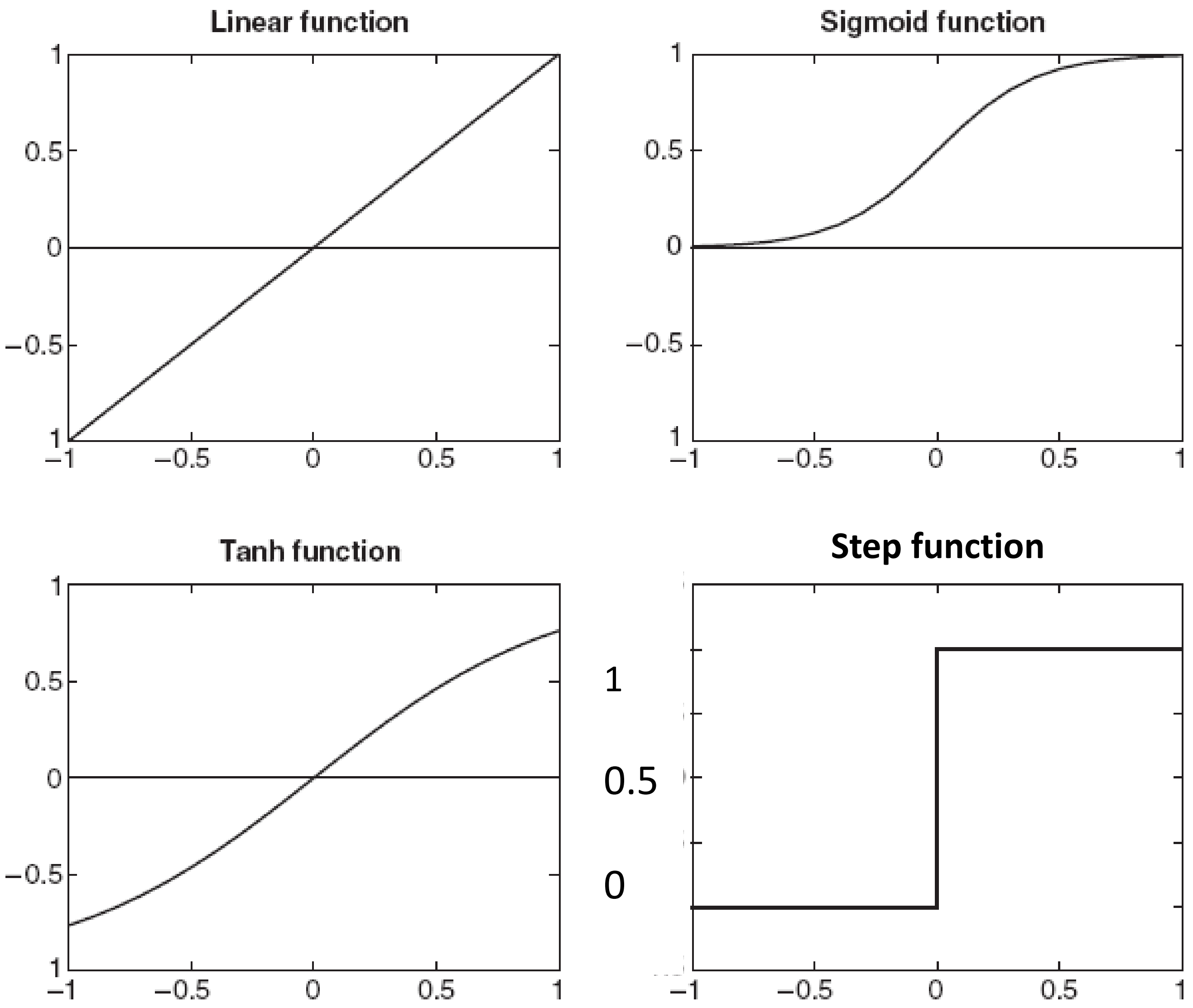


A Look at a Processing Node

- Sums up weighted inputs into it, adds bias and uses activation function to compute result.



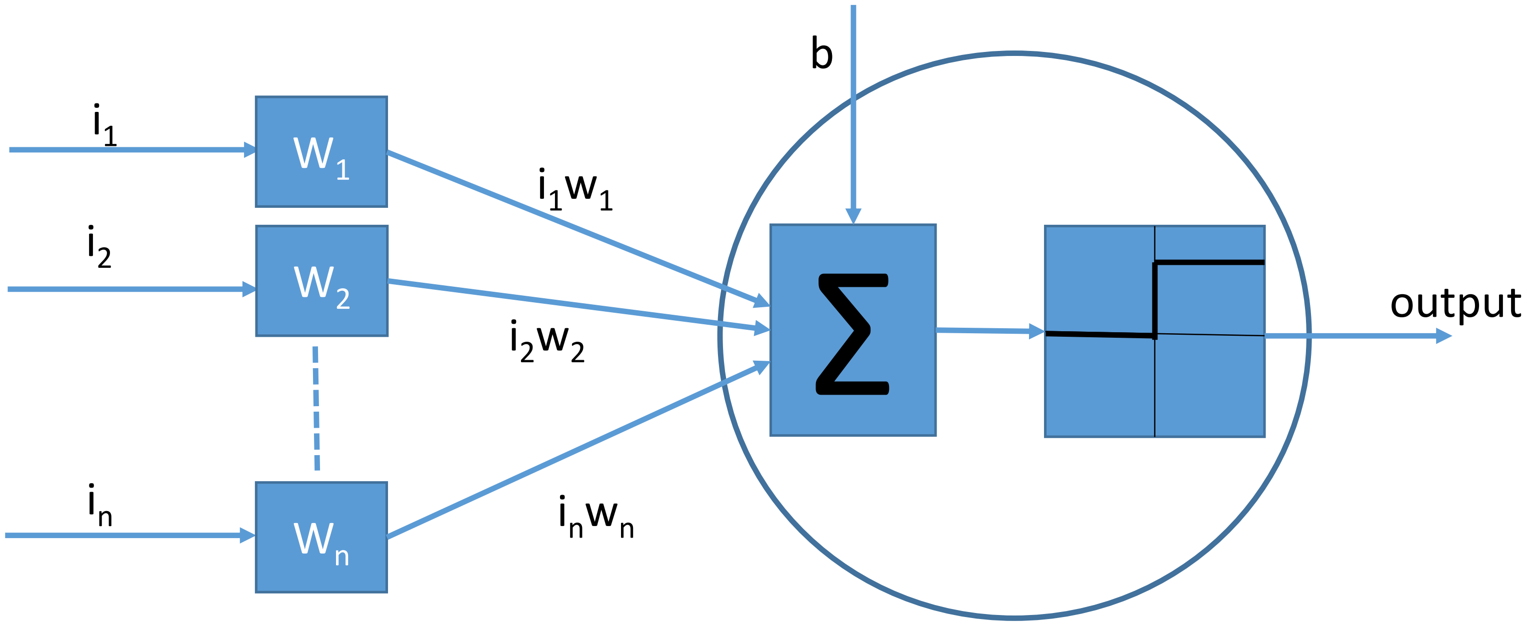
Activation Functions; Examples



Function	Expression
Linear	σ(z)=kz
Sigmoid	$\sigma(z)=\frac{1}{1+e^{-z}}$
Hyperbolic tan (tanh)	$\sigma(z)=\frac{e^{-z}-e^{z}}{e^{-z}+e^{z}}$
Gaussian/Radial Basis function	$\sigma(z)=e^{-\frac{z^2}{2}}$
Step function	$\sigma(z)=\begin{cases}0, & z \le \theta \\ 1, & z > \theta\end{cases}$

Simple Perceptron

- A one-layer feed forward network

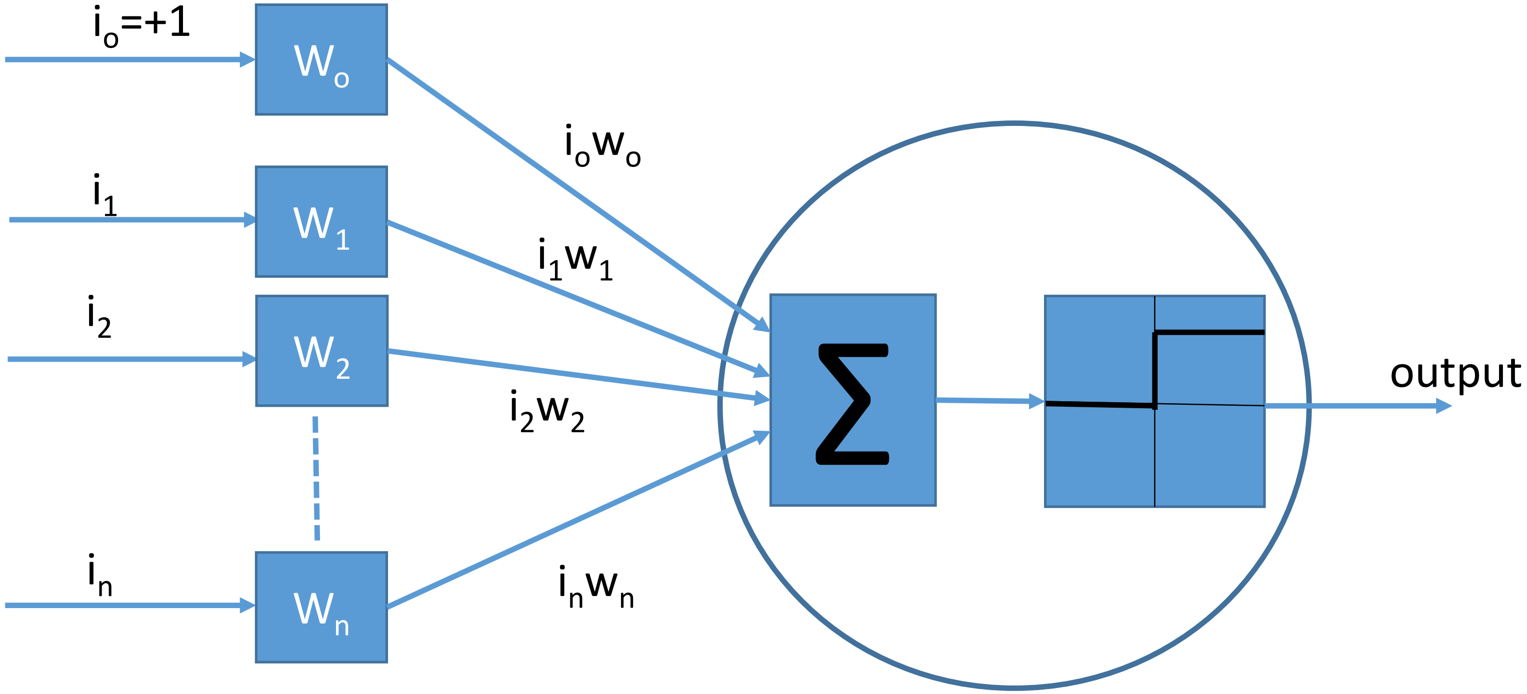


Input to activation function:

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

Simple Perceptron

- Often, bias term represented as a weight with unit input.



Input to activation function:

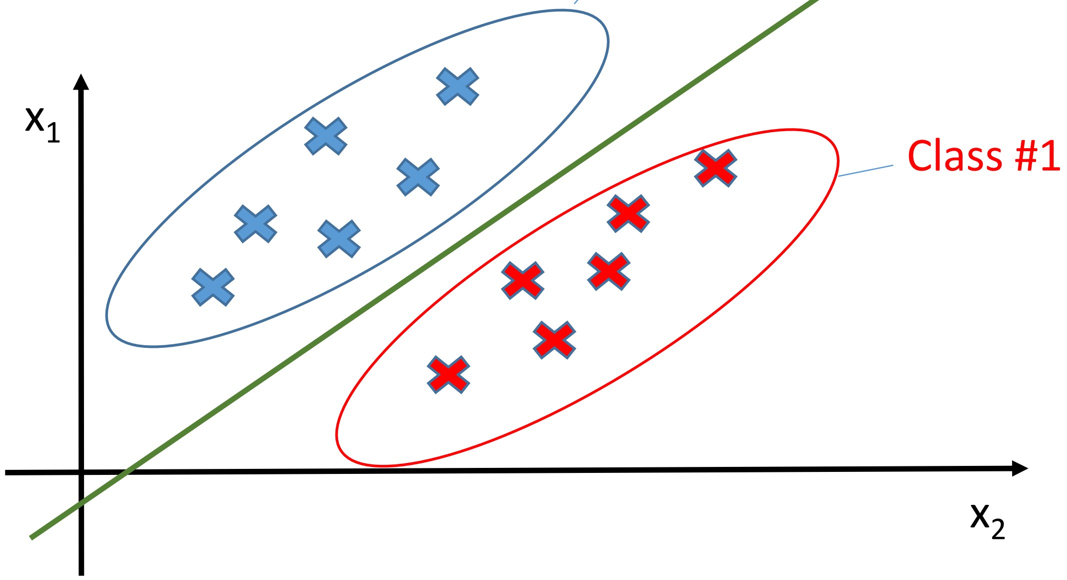
$$\sum_{i=0}^{n} w_i x_i$$

Simple Perceptron

- Used to separate linearly separable patterns.
  - E.g., In 2D, the boundary line is always a straight line…

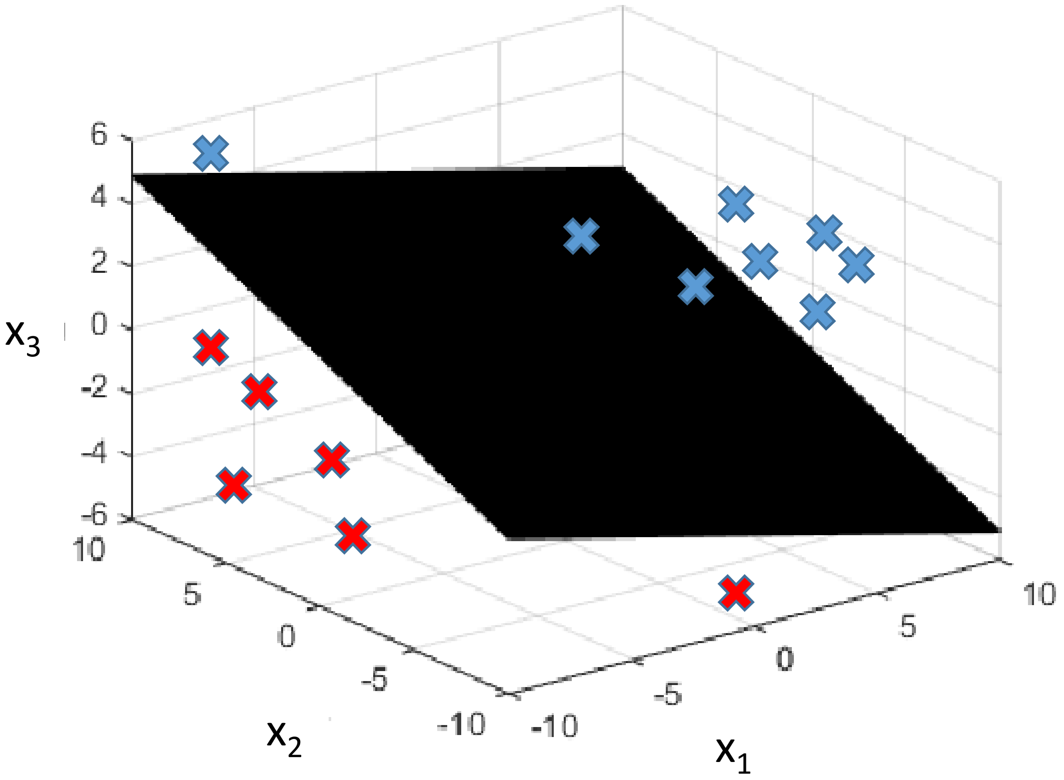
$\sum_{i=0}^{2} \boldsymbol{w_i x_i}=0$

$\Rightarrow \boldsymbol{w_1 x_1 + w_2 x_2} + b = 0$



  - Weights determine the slope of the line. Bias determines the y-intercept.

Simple Perceptron



- In 3D (i.e., 3 features, x_1, x_2 and x_3), discrimination boundary is a 2D plane:

  $\boldsymbol{w_1 x_1}$+$\boldsymbol{w_2 x_2}$+$\boldsymbol{w_3 x_3}$+b=0
- In general, when we have n features, our decision boundary is n-1 dimensional.
- A **hyperplane** is a subspace of one dimension less than its ambient space
- Learning process of a perceptron classifier can be said to amount to learning a hyperplane classifier

Perceptron Learning Algorithm

- Learning Process is basically to find the vector $\boldsymbol{w}$ of weights $\boldsymbol{w_o}$ through $\boldsymbol{w_n}$ that define a hyperplane separating the classes.
- Algorithm begins with some initial weights vector $\mathbf{w}^{\mathrm{i}}$
- Algorithm cycles through the training set, a training sample at a time and makes decisions on adjustment of weights with each input sample.
- It's a mistake-driven algorithm
  - Only updates **w** when it makes a mistake; i.e., when it wrongly predicts the label of the current training example
  - Doesn't update **w** when it correctly predicts the label of the current training example

Perceptron Learning Algorithm

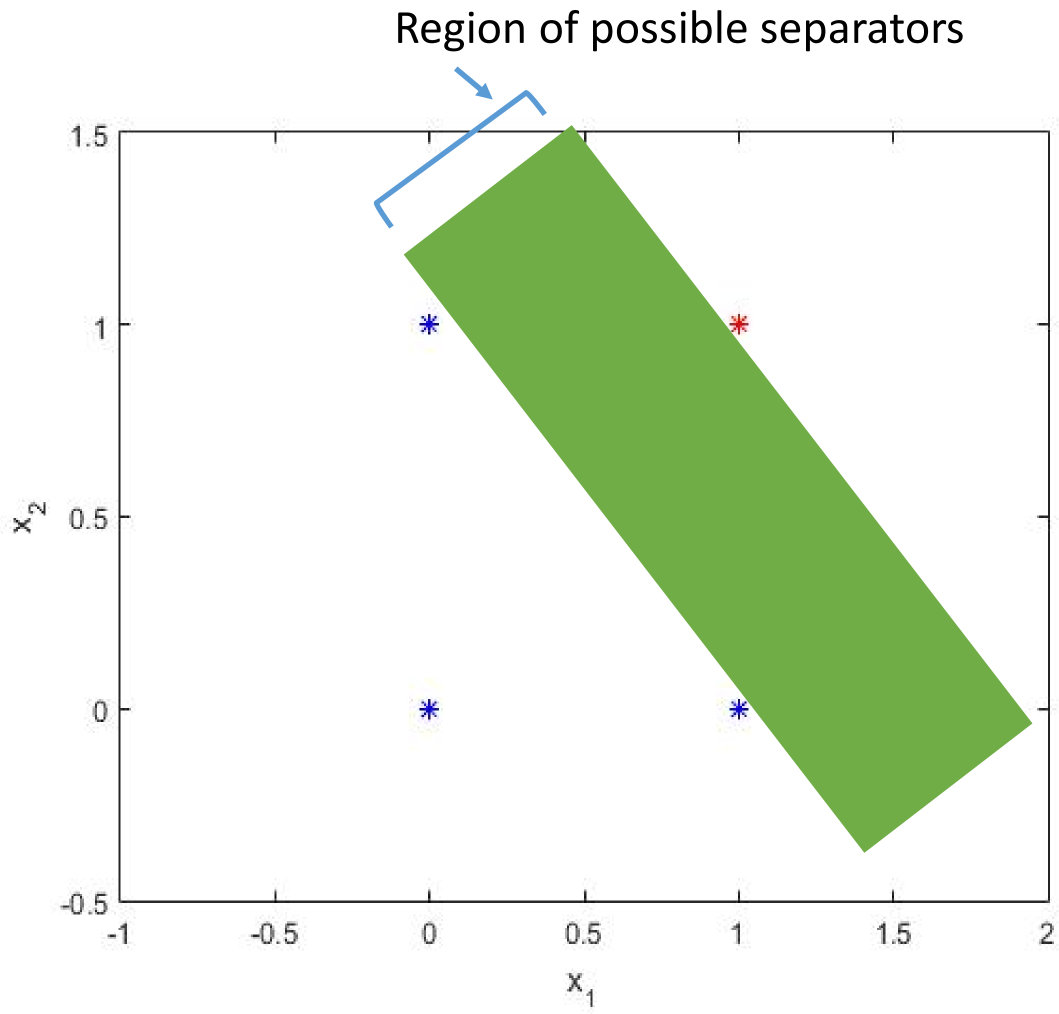
1. Initialize weight vector
2. Select random sample from training set as input
   - ✓ Lets denote each sample as (x_{train},y_{target}) where x is the feature vector and y_{target} is the class label of the sample.
3. Compute the output $y_{classifier}$
4. If $y_{classifier}$=y_{target}, do nothing.
5. Otherwise compute the error $\delta = y_{target} - y_{classifier}$ and the change in weight $\Delta w = \eta \times \delta \times x_{train}$ and update the weight vector using $w_{new} = w_{old} + \Delta w$
6. Repeat this until the entire training set is classified correctly.

Perceptron Learning Example

- Perceptron that implements a logical AND

Input #1	Input #2	Output
0	0	0
0	1	0
1	0	0
1	1	1

- Initial weight vector: [-2 3 1];
  - First component of vector is bias term
- Learning rate: 0.4
- Activation function: $\sigma(z)=\begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$



Perceptron Learning Example

- Notation: $[w_o\ w_1\ w_2]$= [-2 3 1] where w_o is weight of bias term; Features represented as $[x_o\ x_1\ x_2]$, where x_o is the bias term (recall this equals 1), and $x_1\ x_2$ are the input features for our training example. Denote, $y_t=y_{target}$; $y_c=y_{classifier}$
- Epoch #1

$w=[w_o\ \ w_1\ \ w_2]$			$x=[x_o\ x_1\ \ x_2\ ]$			y_t	Input, z to Activation function	$\sigma(z)=y_c$	$\delta= y_t\text{-}y_c$	$\Delta w=[\Delta w_o\ \ \Delta w_1\ \Delta w_2]$		
-2	3	1	1	0	0	0	**-2**	**0**	**0**	**0**	**0**	**0**
-2	**3**	**1**	1	0	1	0	**-1**	**0**	**0**	**0**	**0**	**0**
-2	**3**	**1**	1	1	0	0	**1**	**1**	**-1**	**-0.4**	**-0.4**	**0**
-2.4	**2.6**	**1**	1	1	1	1	**1.2**	**1**	**0**	**0**	**0**	**0**
-2.4	**2.6**	**1**										

Perceptron Learning Example

Epoch #2

$w=[w_o \quad w_1 \quad w_2]$			$x=[x_o \; x_1 \;\; x_2\,]$			y_t	z	$\sigma(z)=y_c$	$\delta= y_t\text{-}y_c$	$\Delta w=[\Delta w_o \quad \Delta w_1 \; \Delta w_2]$		
-2.4	**2.6**	**1**	1	0	0	0	**-2.4**	**0**	**0**	**0**	**0**	**0**
-2.4	**2.6**	**1**	1	0	1	0	**-1.4**	**0**	**0**	**0**	**0**	**0**
-2.4	**2.6**	**1**	1	1	0	0	**0.2**	**1**	**-1**	**-0.4**	**-0.4**	**0**
-2.8	**2.2**	**1**	1	1	1	1	**0.4**	**1**	**0**	**0**	**0**	**0**
-2.8	**2.2**	**1**										

Perceptron Learning Example

Epoch #3

$w=[w_o \ \ w_1 \ \ w_2]$			$x=[x_o \ x_1 \ \ x_2 \ ]$			y_t	z	$\sigma(z)=y_c$	$\delta= y_t-y_c$	$\Delta w=[\Delta w_o \ \ \Delta w_1 \ \Delta w_2]$		
-2.8	2.2	1	1	0	0	0	-2.8	0	0	0	0	0
-2.8	2.2	1	1	0	1	0	-1.8	0	0	0	0	0
-2.8	2.2	1	1	1	0	0	-0.6	0	0	0	0	0
-2.8	2.2	1	1	1	1	1	0.4	1	0	0	0	0
-2.8	2.2	1										

Final Weight Vector, w =[-2.8 2.2 1]

Perceptron Convergence Theorem

- Given data with linearly separable classes, the perceptron learning rule is guaranteed to find the separating hyperplane in a finite number of iterations
  - Requirement: Learning rate sufficiently small