**Machine Learning : The Perceptron**

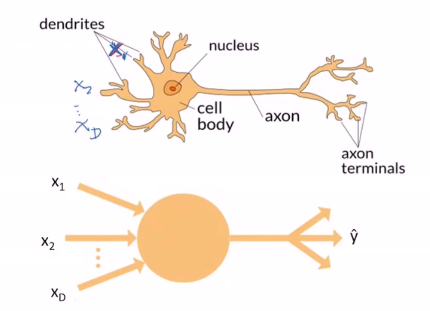
**\*Comment Notes : Perceptron’s like neural networks has a biological motivation of wanting to design a computational learning method that mimics human thinking and learning. It’s an incremental learning algorithm that tries to minimize error in its predictive performance.**

**Perceptron**

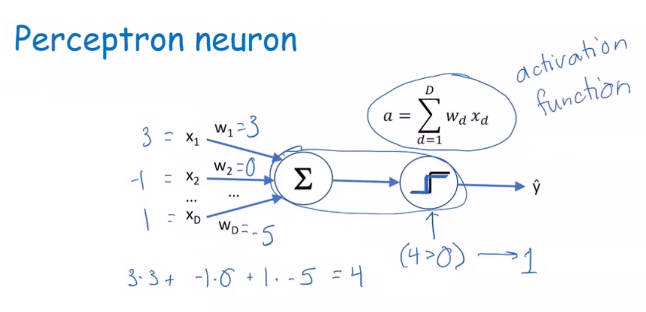
* **Supervised learning algorithm** 
  + **Can map datapoints to class labels**
* **Regression (numeric) or classification (categorical)**
* **Allows us to weight features** 
  + **Learn appropriate weights for the features and in so it learns a numeric function that maps the input to the output**

**Biological Inspiration**

* **Inspiration from a human brain**

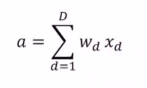


**Perceptron Neuron**



* **X is the feature, W is the weight, the neuron sums the weighted inputs and based on the sum it decides what to output**
* **Y = 1 (positive) or -1 (negative)**
* **If the weighted sum is positive we fire**
* **If the weighted sum is not positive don’t fire**
* **Activation function is the rule for firing a neuron**
  + **So, a is the weighted sum of the inputs and a will determine what the neuron outputs**

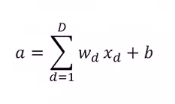
**Input to Neuron**



* **If the weight is 0, means we are ignoring the corresponding feature, because that is not considered in the sum**
* **Positive weight means there will be a positive influence and the greater the positive weight the greater the influence on the output**
* **Negative weight is the opposite of a positive weight because it causes the activation to decrease**

**Activation Function**

* **If then output 1 (positive example)**
* **Else output -1 (negative example)**
* **Use non-zero threshold**
  + **If**
  + **Can accomplish the same thing through a bias term b**



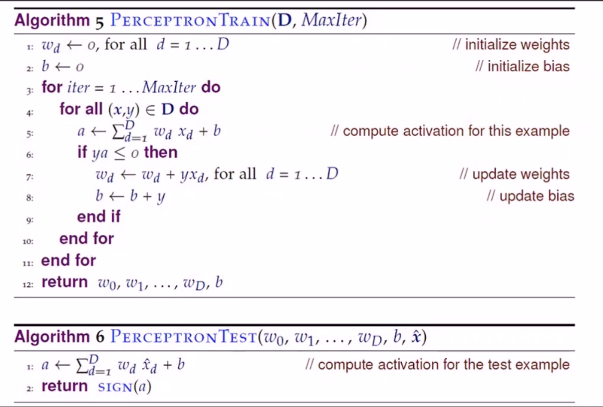
**Class Labels**

* **Binary Classifier**
* **Classes are + (positive) and – (negative)**
* **Denote by y = + 1 and y = - 1**
* **Once activation is computed, output is sign of a**

**Training a Perceptron**

* **Intuition**
  + **If output is -1 but should have output +1, need to increase weights**
    - **This is a False Negative**
    - **Nudge in the direction of**
  + **If output is +1 but should have output -1, need to decrease weights**
    - **This is a False Positive**
    - **Nudge in the direction of**
* **We can see that there is an incremental adjustment of the weight depending on what the perceptron got wrong**
* **If then the activation is wrong, so we need to adjust** 
  + **is either +1 or -1 so if then and that is wrong**
  + **is the true label and is the activation function**
  + **is corrected by nudging it in the direction of the**
  + **The bias term is also nudged in the correct direction**

**Algorithm**



#Incremental Learning Algorithms

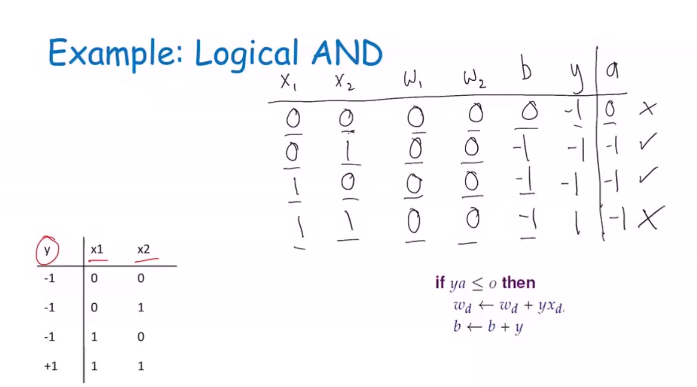
🡨 #Epochs

#Wrong

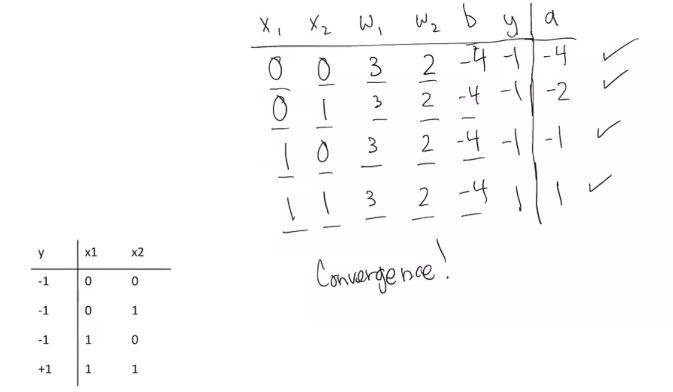
* **D is the set of training data**
* **MaxIter is the maximum number of iterations (Hyperparameter)**
* **Ordering of the datapoint is important (Hyperparameter)**
* **Amount to change the weight (Hyperparameter)**

**Logical AND Example**

* **First Epoch (Iteration)**

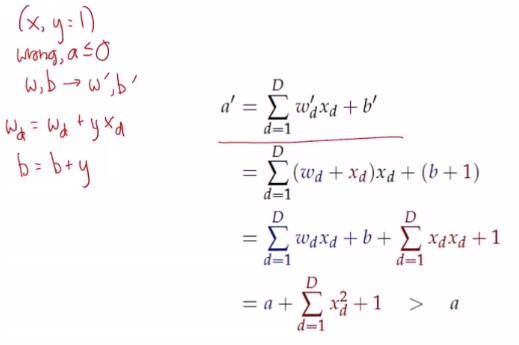


* **Final Epoch (she skipped ahead)**



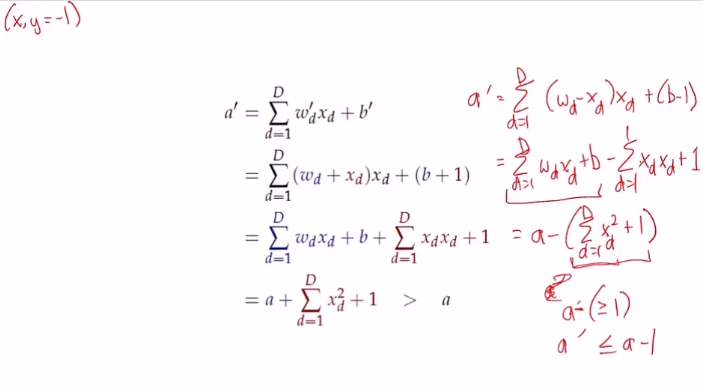
* **Notice that**
* **Convergence is a good time to stop**

Positive Example



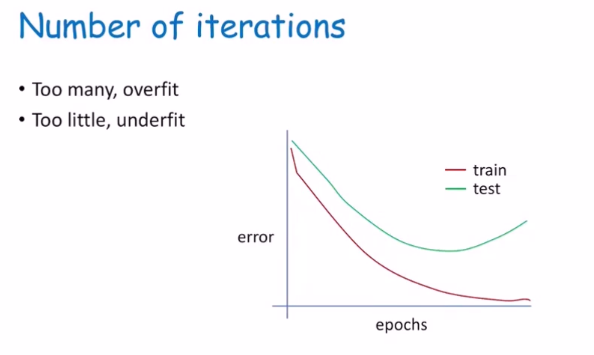
**\*Comment Notes : If it’s wrong that means that so we need to adjust the weight and bias term; New value of a’ is always at least one greater than the old a;**

**Negative Example**



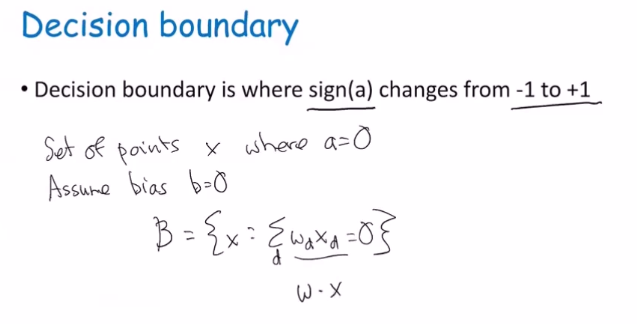
**\*Comment Notes : New value of a’ is always at least one less than the old a;**

**Number of Iterations**

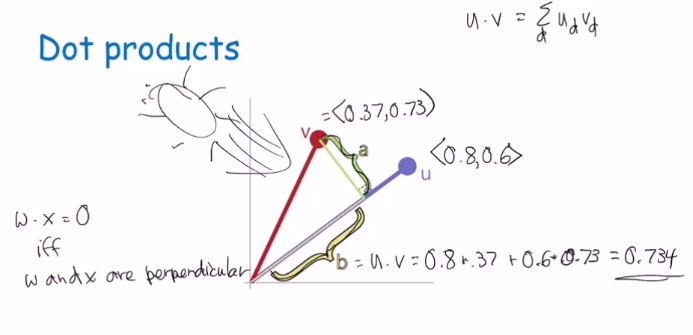


* More epochs = overfit
* Less epochs = underfit

**Decision Boundary**

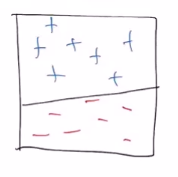


**Dot Products**

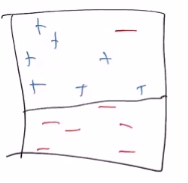


**Linearly Separable**

* If the classes can be separated by a hyperplane, then they are linearly separable
* Perceptron can learn any linearly separable function



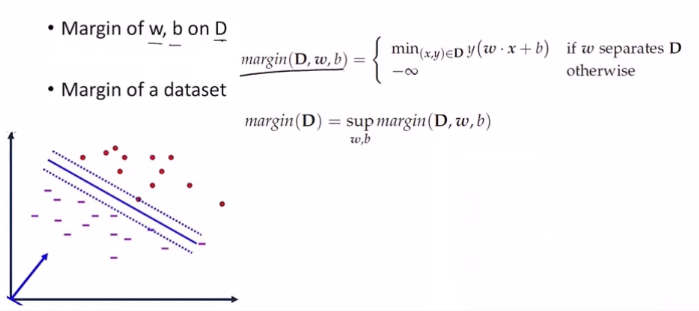
* The above picture is not a problem for the perceptron it will converge
* If the problem is not linearly separable the perceptron will never converge!



* The above picture will not converge

**Margin**

* Margin is the distance between decision boundary specifically the hyperplane and the nearest point
* Used to help us understand hard problems from easy problems
* Large Margin 🡪 easy problem; so, probably need fewer epochs, more wiggle room
* Small Margin 🡪 hard problem; so, takes more effort to find the decision boundary that perfectly separates the two classes



* Find(x, y) with the minimum value of where
  + 🡪
  + 🡪
* Sup is max but handles negative infinity differently

**Perceptron Pros and Cons**

* Pros
  + Simple
  + Classification & Regression
  + Handles numeric Data
* Cons
  + Only handles numeric data
  + Only learns linearly separable concepts
  + Computationally expensive
    - May not converge quickly
* Useful tool that lays the foundation for more complex networks
* Exclusive OR (XOR) never converges