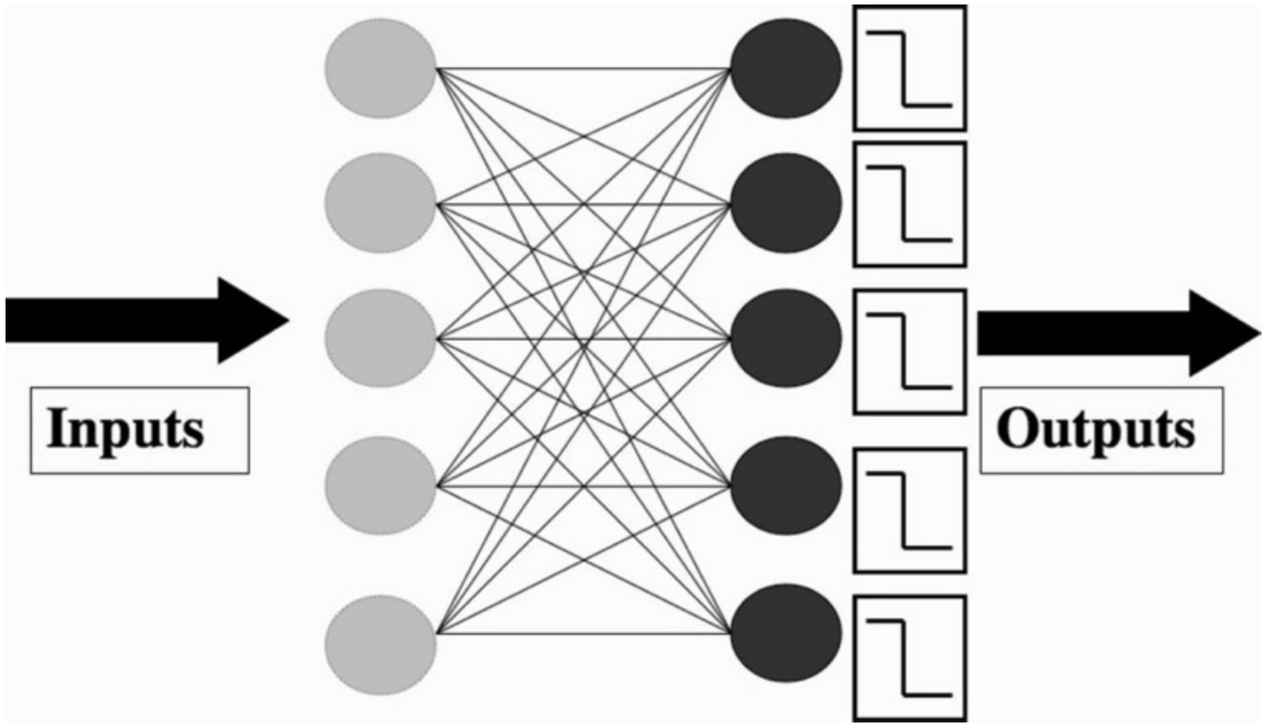
Machine Learning
Perceptron

Jian Liu

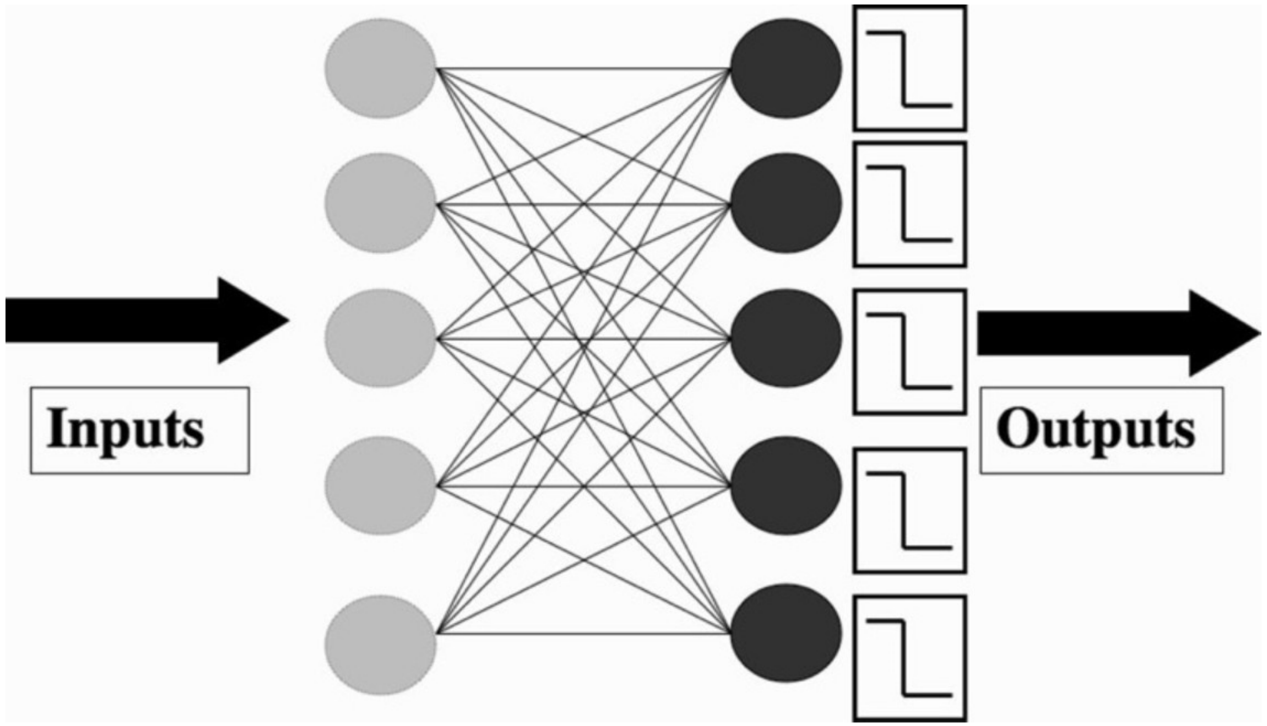


Machine Learning
Perceptron

Jian Liu

Part 1: Perceptron Algorithm

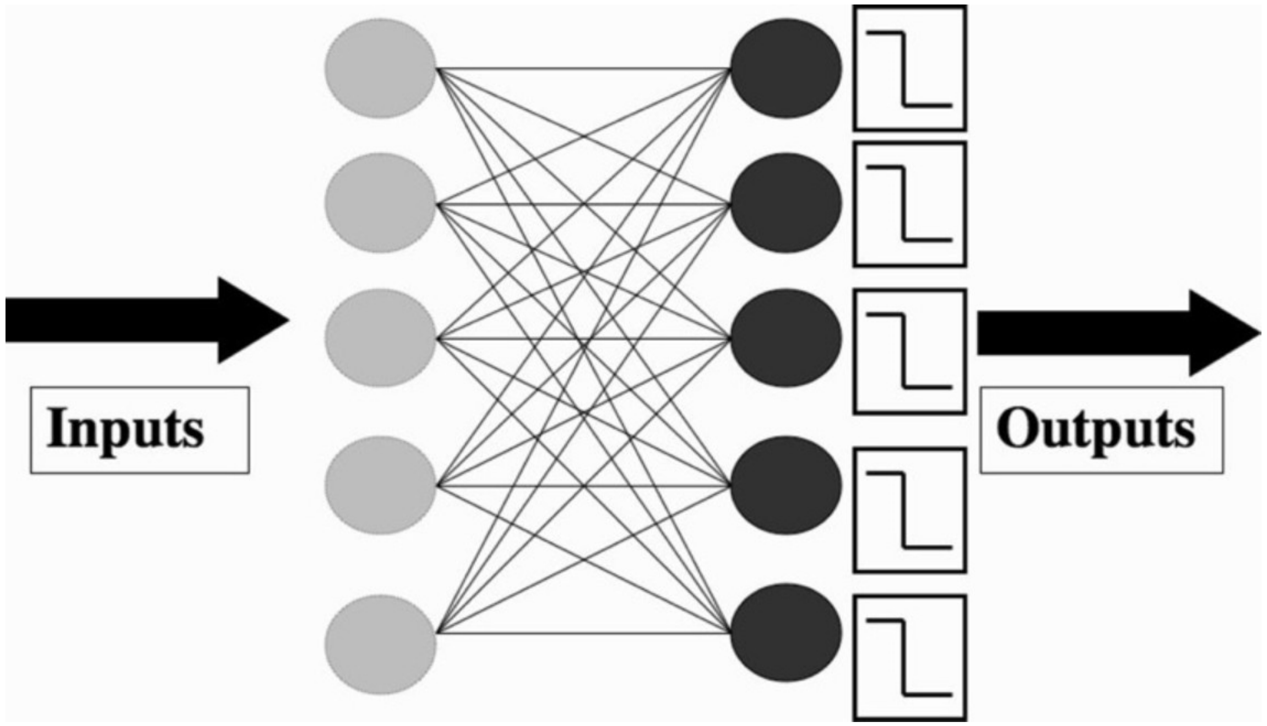
Part 2: Error Functions



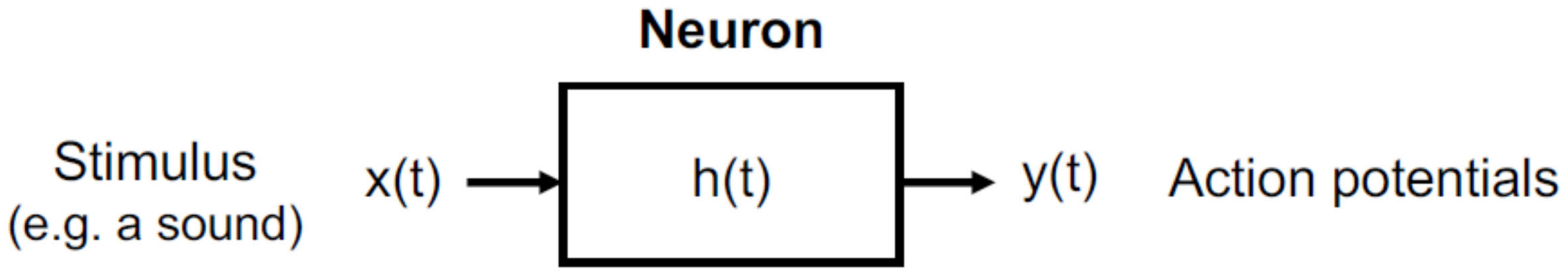
Machine Learning
Perceptron

Jian Liu

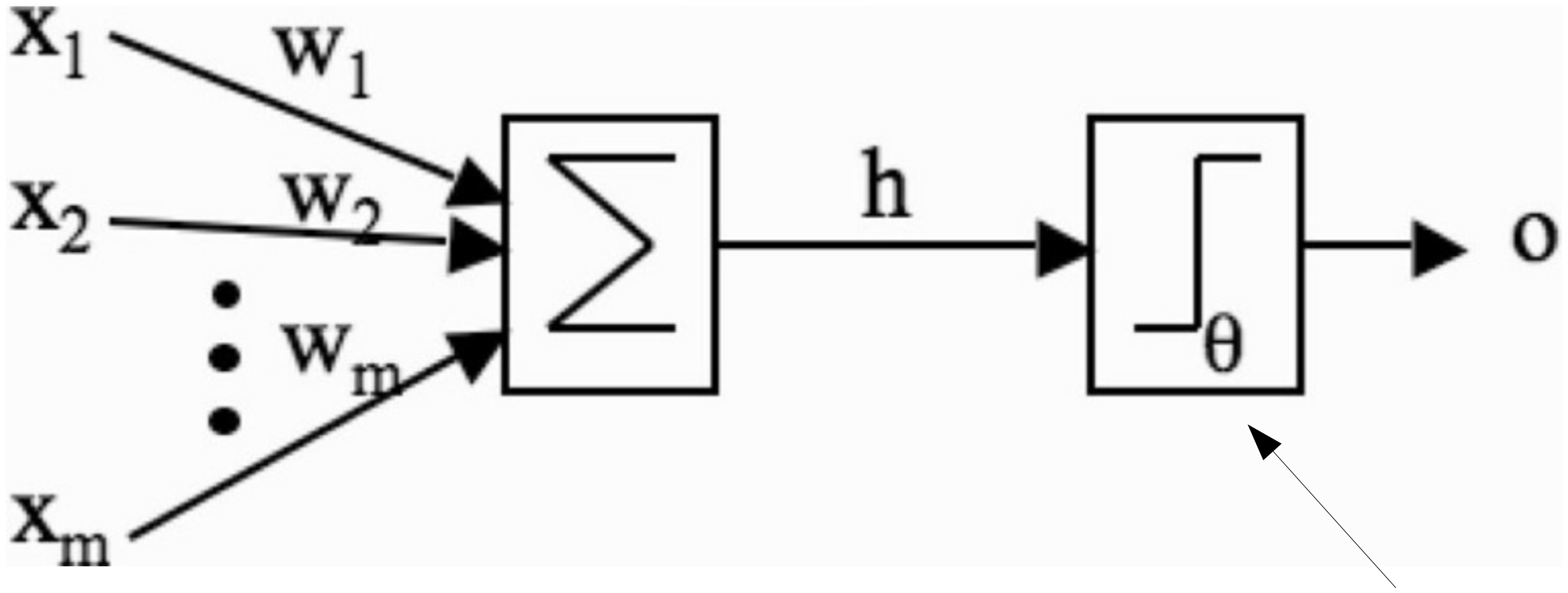
Part 1: Perceptron Algorithm



Neuron as an information processor



McCulloch and Pitts Neuron



Activation function

$$h_w(\boldsymbol{x}) = \sum_i w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$$

$$o(h_w) = \begin{cases} 1 & \text{if} \quad h_w > \theta \\ 0 & \text{if} \quad h_w \leq \theta \end{cases}$$

Linear neurons

The neuron has a real-valued output which is a weighted sum of its inputs

weight vector

$$\hat{y} = \sum_i w_i x_i = \mathbf{w}^T \mathbf{x}$$

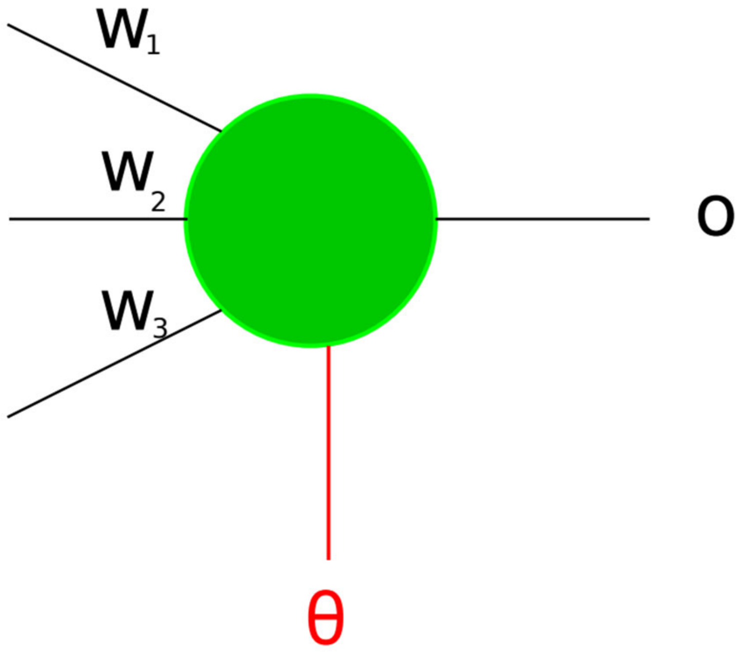
Neuron's estimate of the desired output

input vector

The aim of learning is to minimize the discrepancy between the desired output and the actual output

Perceptron

- The perceptron is a highly stylised version of a real neuron
- It has a number of inputs (N), which can be set a numerical value $x_i, i = 1, \cdots, N$
- Each input has a weight, also a numerical value w_i
- It has threshold θ



Perceptron

$$h = w_1 x_1 + w_2 x_2 + w_3 x_3 - \theta$$

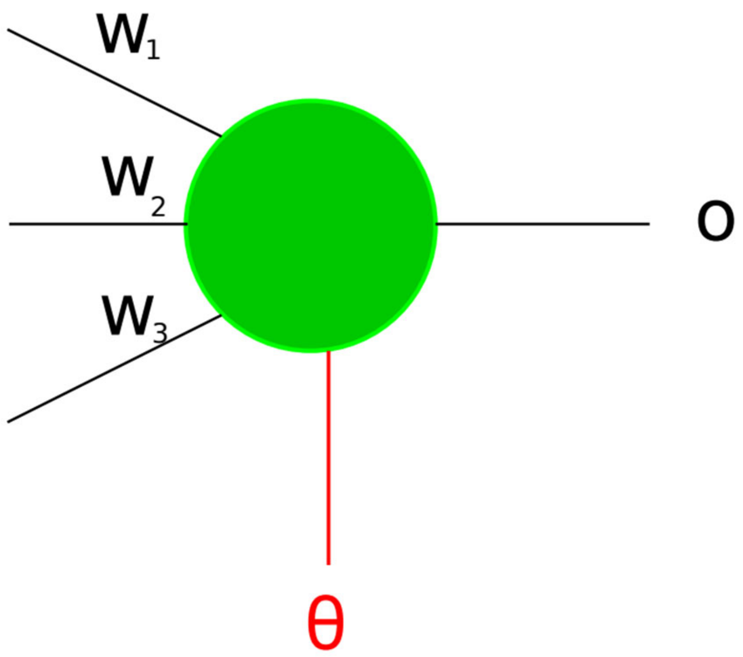
h is called the *local field* x_i are the numerical values of the input, w_i weights and θ is called threshold (or bias)

$$o = \mathcal{H}(h),$$

where:

$$\mathcal{H}(x) = \begin{cases} x < 0 : & 0 \\ x \geq 0 : & 1 \end{cases}$$

Also called the step function

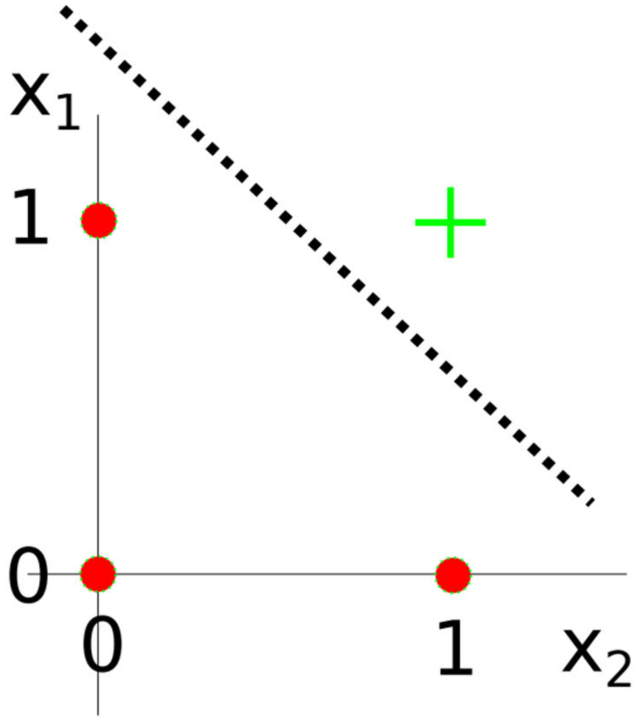


A 2D example

- **AND** gate is a simplified example
- Demonstrates the original thinking behind neural networks (McCullogh & Pitts, 1943)

x_1	x_2	x_3	o
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

Can you solve this classification problem with a perceptron?



The Need for a Training Algorithm

- The example is so simple that you may wonder why you need an algorithm at all!
- But consider having 10 variables
- This entails finding a hyperplane in a 10 dimensional space!
- We then need 10 weights and a threshold!
- Multiplying weights (and threshold) by a positive factor does not change the decisions
- Multiplying by a negative factor reverses the decision. Why?

Intuition: If classification is wrong and desired output is one, my weights may be too large. If classification is wrong and desired output is 0, my weights may be too small. Why not add (subtract) the input pattern to the weights?

Absorbing the Threshold

Simple Observation: You can consider the threshold as an extra input which is always clamped to one. If the weight of this input is minus the threshold, you will have a perceptron taking the same decision.

This follows from:

$$w_1x_1 + w_2x_2 + w_3x_3 = w_1x_1 + w_2x_2 + w_3 = w_1x_1 + w_2x_2 - \theta$$

In practice I need to add an extra column of ones to my dataset:

x_1	x_2	x_3	o
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	1

I can present the *perceptron algorithm* without treating the threshold separately. Which is good as the threshold needs to adapted just as the weights. (Why?)

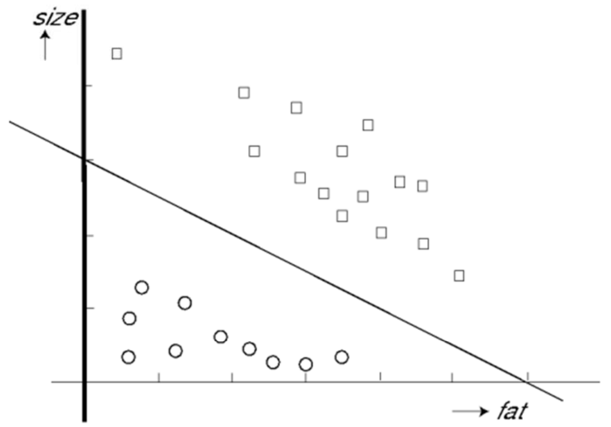
The Perceptron Algorithm

The algorithm is as follows. Start with a perceptron with $N+1$ inputs, one of which is clamped to value $+1$, and $N+1$ weights so that a threshold is unnecessary.

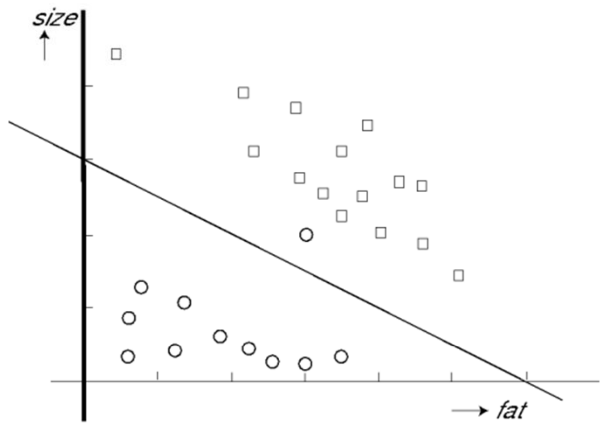
1. **Start**: Choose a random set of weights: $\boldsymbol{w}_0$. For later reference, denote the current set of weights by $\boldsymbol{w}_i$, so a the start $\boldsymbol{w}_i = \boldsymbol{w}_0$.
2. **Continue:** Pick a random data point $(\boldsymbol{x}_j, d_j), j = 1, \cdots, D$, where $\boldsymbol{x}_j$ is a point for which classification $d_j \in \{0, 1\}$ is given.
3. Evaluate $\boldsymbol{w}_i^T \boldsymbol{x}_j$. If $\begin{cases} \boldsymbol{w}^T \boldsymbol{x_j} < 0 \quad \textbf{AND} \quad d_j = 0 : \text{goto } \textbf{Continue} \\ \boldsymbol{w}^T \boldsymbol{x_j} < 0 \quad \textbf{AND} \quad d_j = 1 : \text{goto } \textbf{Add} \\ \boldsymbol{w}^T \boldsymbol{x_j} \geq 0 \quad \textbf{AND} \quad d_j = 1 : \text{goto } \textbf{Continue} \\ \boldsymbol{w}^T \boldsymbol{x_j} \geq 0 \quad \textbf{AND} \quad d_j = 0 : \text{goto } \textbf{Subtract} \end{cases}$
4. **Add:** $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \boldsymbol{x}$; Goto **Continue**
5. **Subtract:** $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \boldsymbol{x}$; Goto **Continue**

An example:
Crisps: portion size and fat content

- Box: Should be classified as a health hazard
- Circle: OK for consumption



This perceptron is fine. Give weight and threshold values.



This perceptron misclassifies the point (3,3). Apply the algorithm

The Perceptron Algorithm

- In plain English: if data is linearly separable, the perceptron algorithm will find a set of weights that separates the two classes
- Mathematically, it is formulated like this:
- Assume a single class of points exists that is on one side of a plane through the origin
- For every $\vec{x} \in \mathcal{C}$, assume there is a $\vec{w}^*$ such that $\vec{w}^* \cdot \vec{x} \geq \delta > 0$, for all patterns $\vec{x}$.
- Wait! we had two classes?

Let $\mathcal{C}$ be a set of patterns.

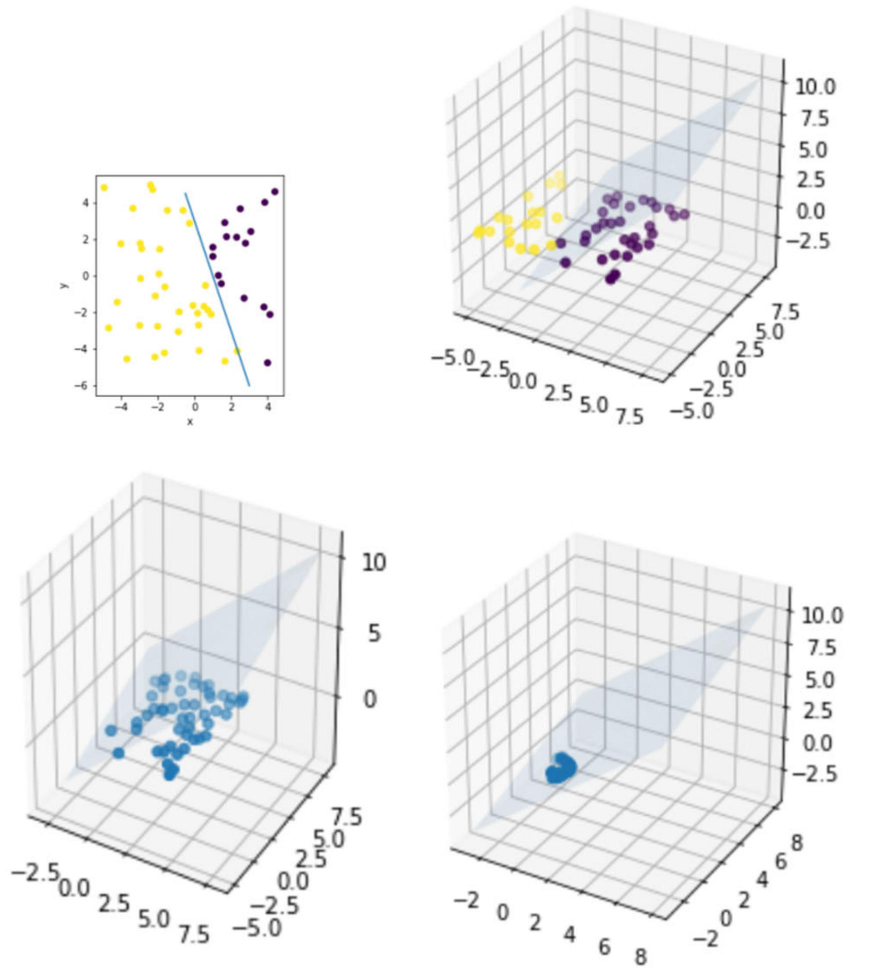
Perceptron Algorithm

If for every $\vec{x} \in \mathcal{C}$, there is a $\vec{w}^*$ such that $\vec{w}^* \cdot \vec{x} \geq \delta > 0$, for all patterns $\vec{x}$, then the perceptron algorithm will go through a finite number of steps **Add** and **Subtract**

- Implication: algorithm has settled on a set of weights that classify correctly
- Not necessarily $\vec{w}^*$

Geometrical Ideas of Proof

- Absorb threshold
- Assume there are two classes of patterns, one: $\mathcal{C}^+$ that must be classified as 1, one $\mathcal{C}^-$ that needs to be classified as 0
- If $\vec{x} \in \mathcal{C}^-$, replace by $-\vec{x} \in \mathcal{C}^+$
- For good measure: normalise $\vec{x} \rightarrow \frac{\vec{x}}{|\vec{x}|}$
- Now the conditions for the Perceptron Theorem satisfied



Proof of the Perceptron Theorem

Follow the following quantity as we step through the algorithm:

$$\frac{\vec{w^*} \cdot \vec{w_n}}{| \vec{w^*} || \vec{w_n} |} = \cos\theta$$

- Can we do this? We don't even know $\vec{w^*}$
- It turns out we can!
- Observe that this quantity is a cosine
- So must be smaller than one.
- If every update of the algorithm increases $\cos\theta$ by a finite amount it must stop
- Even if we can't calculate $\cos\theta$ itself

Proof of the Perceptron Theorem
What Happens to the Numerator?

- Imagine we already have been through *n* steps of the algorithm
- The algorithm goes through **Add**, so the weights change

$$\vec{w^*} \cdot \vec{w_{n+1}} = \vec{w^*} \cdot (\vec{w}_n + \vec{x}) < \vec{w^*} \cdot \vec{x} + \delta$$

- so the numerator increases by at least δ

Proof of the Perceptron Theorem

- Remember we normalised all input patterns $\mid \vec{x} \mid = 1$
- Again, we have already stepped n times through **Add**

$$\mid \vec{w^*} \mid\mid \vec{w}_{n+1} \mid = \mid \vec{w^*} \mid\mid \vec{w}_n + \vec{x} \mid$$

- $\vec{w^*}$ never changes, so we only have to consider:

$$(\vec{w_n} + \vec{x})^2 = (\vec{w_n}^2 + 2\vec{w_n} \cdot \vec{x} + 1)^2 < \vec{w_n}^2 + 1$$

- The latter step: $\vec{w_n} \cdot \vec{x} < 0$, otherwise the algorithm would not have done **Add**

The Perceptron Algorithm

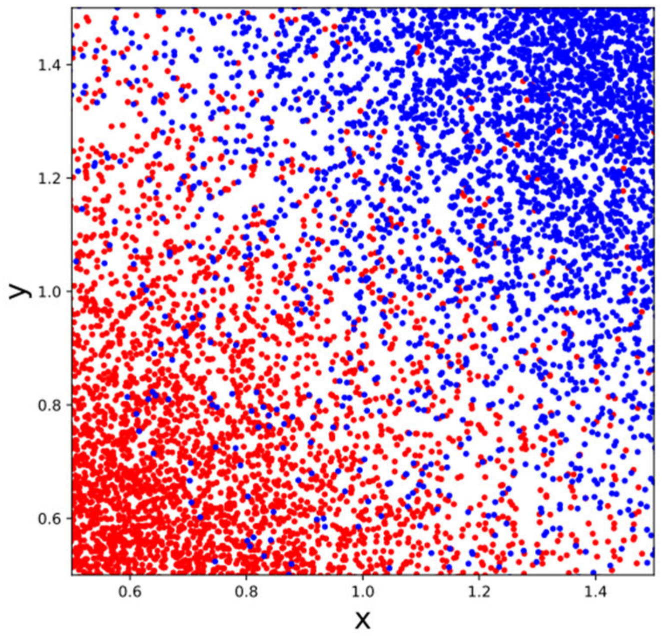
The proof of the algorithm relies on the fact that each **Add** step increases the cosine of angle between $\vec{w}^*$ and $\vec{w}_n$ by at least δ; a *finite* amount. This means the number of steps must remain finite.

- The perceptron algorithm is an iterative procedure for determining the weights (parameters) of a neural network
- We say we *train* the neural network

Limitations of the Perceptron

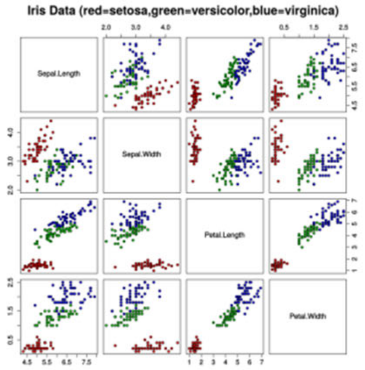
There are substantial problems with the perceptron algorithm

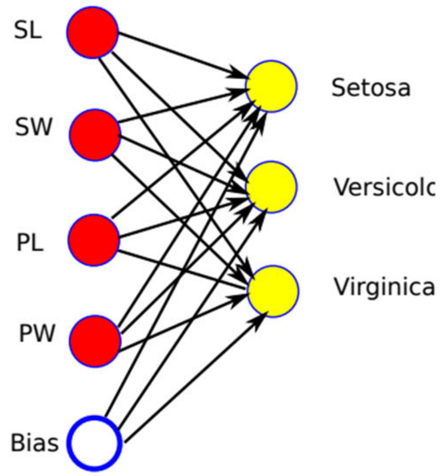
- There is no real stopping criterion
- One can be improvised
- E.g. what to with an almost linearly separable dataset?
- It seems a linear decision boundary would still be reasonable



Limitations of the Perceptron

There are substantial problems with the perceptron algorithm





- More importantly: some datasets are not linearly separable
- The perceptron theorem in this form can't handle such data

- Only two nodes in this network can be successfully trained
- Why?

Recap

- Historically, the perceptron algorithm has had a large influence on the study of neural networks
- It is not the only algorithm to train so-called linear discriminants: Fisher’s discriminant (Bishop, 2006) is probably more widely used
- The perceptron naturally leads to logistic regression
- The quantity δ plays an important role in support vector machines