

Lecture 6. Perceptron

COMP90051 Statistical Machine Learning

Semester 2, 2018

Lecturer: Ben Rubinstein



THE UNIVERSITY OF
MELBOURNE

Copyright: University of Melbourne

This lecture

- Perceptron
 - * Introduction to Artificial Neural Networks
 - * The perceptron model
 - * Stochastic gradient descent

The Perceptron Model

A building block for artificial neural networks, yet another linear classifier

Biological inspiration

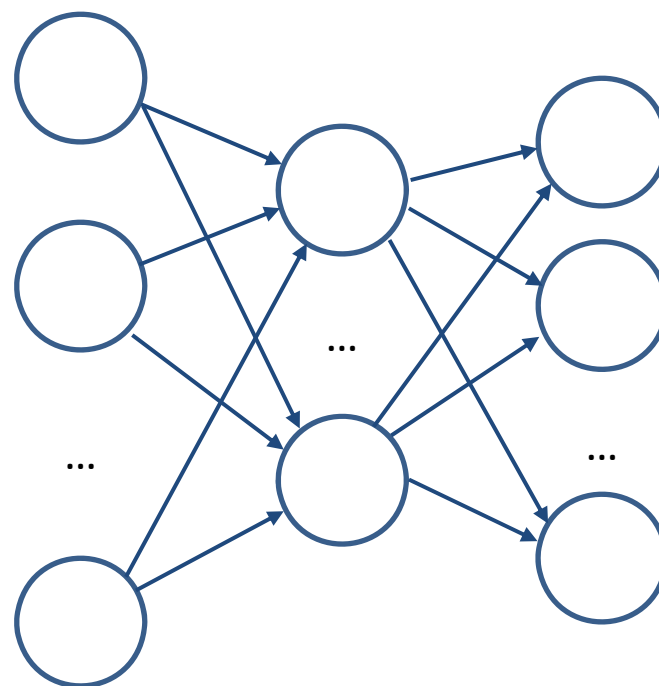
- Humans perform well at many tasks that matter
- Originally neural networks were an attempt to mimic the human brain

photo: Alvesgaspar,
Wikimedia Commons, CC3



Artificial neural network

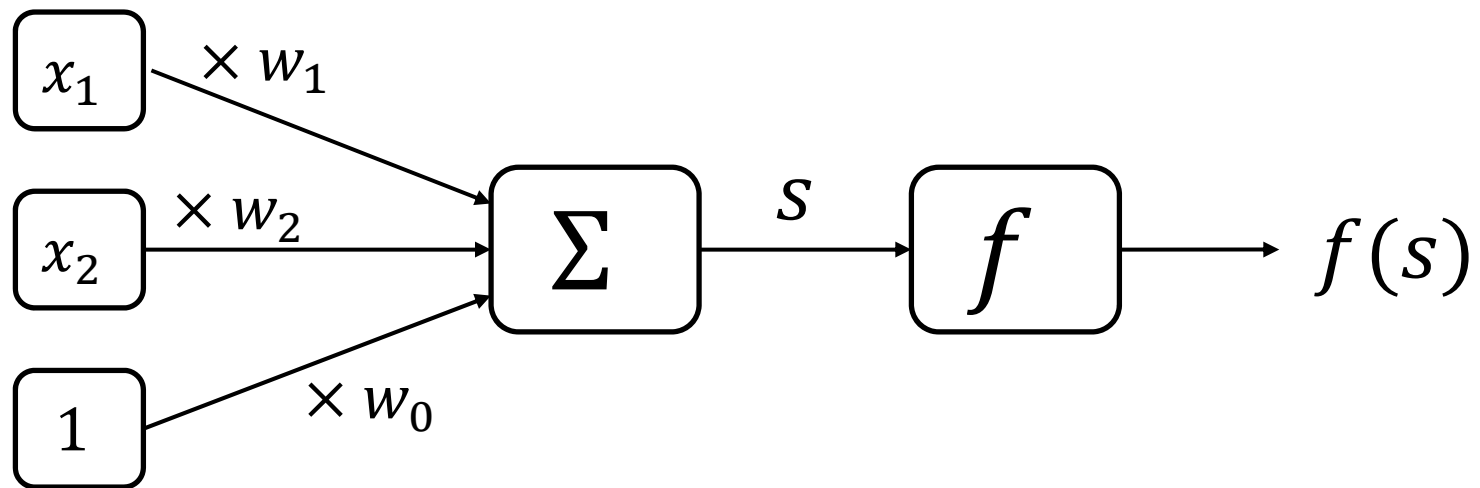
- As a *crude approximation*, the human brain can be thought as a mesh of interconnected processing nodes (neurons) that relay electrical signals
- **Artificial neural network** is a network of processing elements
- Each element converts inputs to output
- The output is a function (called **activation function**) of a weighted sum of inputs



Outline

- In order to use an ANN we need (a) to design network topology and (b) adjust weights to given data
 - * In this subject, we will exclusively focus on task (b) for a particular class of networks called **feed forward** networks
- Training an ANN means adjusting **weights** for training data given a pre-defined network **topology**
- We will come back to ANNs and discuss ANN training in the next lecture
- Right now we will turn our attention to an individual network element because it is an interesting model in itself

Perceptron model



Compare this
model to logistic
regression

- x_1, x_2 – inputs
- w_1, w_2 – synaptic weights
- w_0 – bias weight
- f – activation function

Perceptron is a linear binary classifier

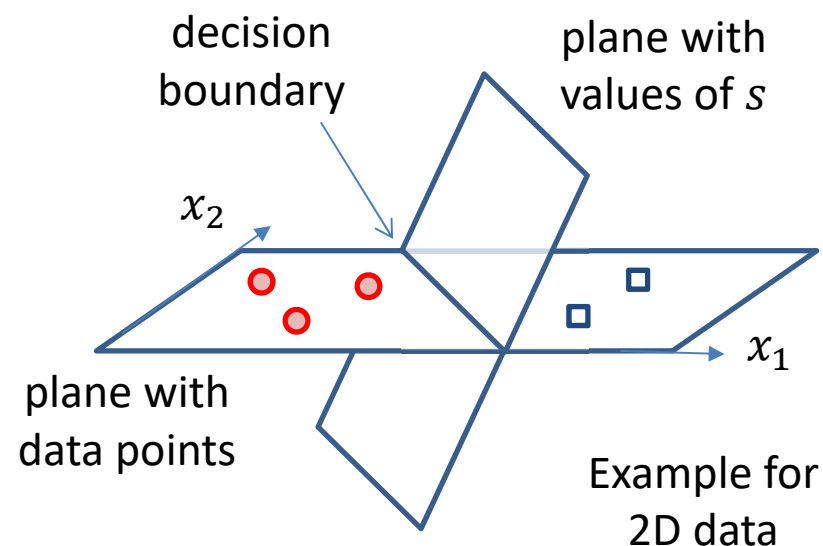
Perceptron is a
binary classifier:

Predict class A if $s \geq 0$

Predict class B if $s < 0$

where $s = \sum_{i=0}^m x_i w_i$

Perceptron is a linear classifier: s
is a linear function of inputs, and
the decision boundary is linear



Exercise: find weights of a perceptron capable of perfect classification of the following dataset

x_1	x_2	y
0	0	Class B
0	1	Class B
1	0	Class B
1	1	Class A



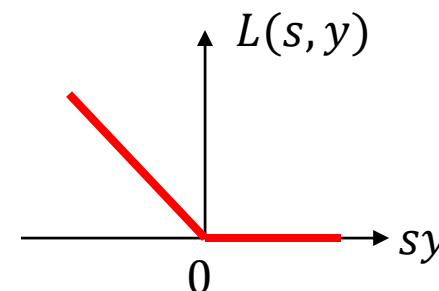
art: OpenClipartVectors
at pixabay.com (CC0)

Loss function for perceptron

- “Training”: finds weights to minimise some loss. Which?
- Our task is binary classification. Let’s arbitrarily encode one class as $+1$ and the other as -1 . So each training example is now $\{\mathbf{x}, y\}$, where y is either $+1$ or -1
- Recall that, in a perceptron, $s = \sum_{i=0}^m x_i w_i$, and the sign of s determines the predicted class: $+1$ if $s > 0$, and -1 if $s < 0$
- Consider a single training example. If y and s have **same sign** then the example is classified correctly. If y and s have **different signs**, the example is misclassified


Loss function for perceptron

- Consider a single training example. If y and s have the same sign then the example is classified correctly. If y and s have different signs, the example is misclassified
- The perceptron uses a loss function in which there is no penalty for correctly classified examples, while the penalty (loss) is equal to s for misclassified examples*
- Formally:
 - * $L(s, y) = 0$ if both s, y have the same sign
 - * $L(s, y) = |s|$ if both s, y have different signs
- This can be re-written as $L(s, y) = \max(0, -sy)$



* This is similar, but not identical to another widely used loss function called **hinge loss**

Stochastic gradient descent

- Split all training examples in B **batches**
- Choose initial $\theta^{(1)}$
- For i from 1 to T 

Iterations over the entire dataset are called epochs
- For j from 1 to B
- Do gradient descent update using data from batch j
- Advantage of such an approach: computational feasibility for large datasets

Perceptron training algorithm

Choose initial guess $\mathbf{w}^{(0)}$, $k = 0$

For i from 1 to T (epochs)

For j from 1 to N (training examples)

Consider example $\{\mathbf{x}_j, y_j\}$

Update*: $\mathbf{w}^{(k++)} = \mathbf{w}^{(k)} - \eta \nabla L(\mathbf{w}^{(k)})$

$$L(\mathbf{w}) = \max(0, -sy)$$

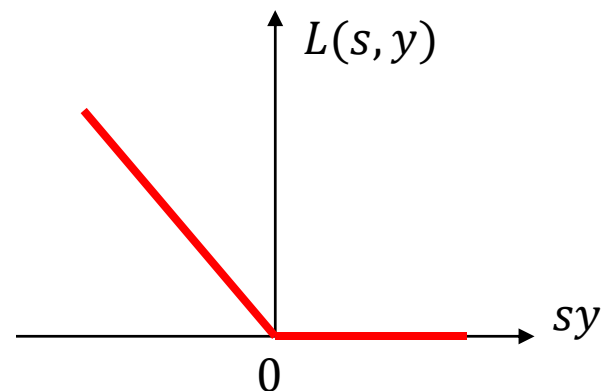
$$s = \sum_{i=0}^m x_i w_i$$

η is learning rate

*There is no derivative when $s = 0$, but this case is handled explicitly in the algorithm, see next slides

Perceptron training rule

- We have $\frac{\partial L}{\partial w_i} = 0$ when $sy > 0$
 - * We don't need to do update when an example is correctly classified
- We have $\frac{\partial L}{\partial w_i} = -x_i$ when $y = 1$ and $s < 0$
- We have $\frac{\partial L}{\partial w_i} = x_i$ when $y = -1$ and $s > 0$
- $s = \sum_{i=0}^m x_i w_i$



Perceptron training algorithm

When classified correctly, weights are unchanged

When misclassified: $\mathbf{w}^{(k+1)} = -\eta(\pm \mathbf{x})$
($\eta > 0$ is called *learning rate*)

If $y = 1$, but $s < 0$

$$w_i \leftarrow w_i + \eta x_i$$

$$w_0 \leftarrow w_0 + \eta$$

If $y = -1$, but $s \geq 0$

$$w_i \leftarrow w_i - \eta x_i$$

$$w_0 \leftarrow w_0 - \eta$$

Convergence Theorem: if the training data is linearly separable, the algorithm is guaranteed to converge to a solution. That is, there exist a finite K such that $L(\mathbf{w}^K) = 0$

Perceptron convergence theorem

- Assumptions

- * Linear separability: There exists \mathbf{w}^* so that $y_i(\mathbf{w}^*)' \mathbf{x}_i \geq \gamma$ for all training data $i = 1, \dots, N$ and some positive γ .
- * Bounded data: $\|\mathbf{x}_i\| \leq R$ for $i = 1, \dots, N$ and some finite R .

- Proof sketch

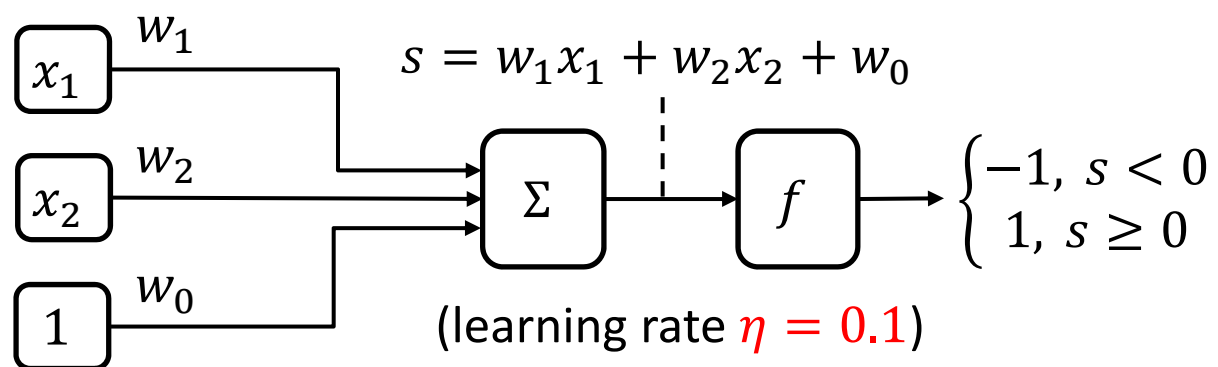
- * Establish that $(\mathbf{w}^*)' \mathbf{w}^{(k)} \geq (\mathbf{w}^*)' \mathbf{w}^{(k-1)} + \gamma$
- * It then follows that $(\mathbf{w}^*)' \mathbf{w}^{(k)} \geq k\gamma$
- * Establish that $\|\mathbf{w}^{(k)}\|^2 \leq kR^2$
- * Note that $\cos(\mathbf{w}^*, \mathbf{w}^{(k)}) = \frac{(\mathbf{w}^*)' \mathbf{w}^{(k)}}{\|\mathbf{w}^*\| \|\mathbf{w}^{(k)}\|}$
- * Use the fact that $\cos(\dots) \leq 1$
- * Arrive at $k \leq \frac{R^2 \|\mathbf{w}^*\|^2}{\gamma}$

Pros and cons of perceptron learning

- If the data is linearly separable, the perceptron training algorithm will converge to a correct solution
 - * There is a formal proof \leftarrow good!
 - * It will converge to some solution (separating boundary), one of infinitely many possible \leftarrow bad!
- However, if the data is not linearly separable, the training will fail completely rather than give some approximate solution
 - * Ugly 😞

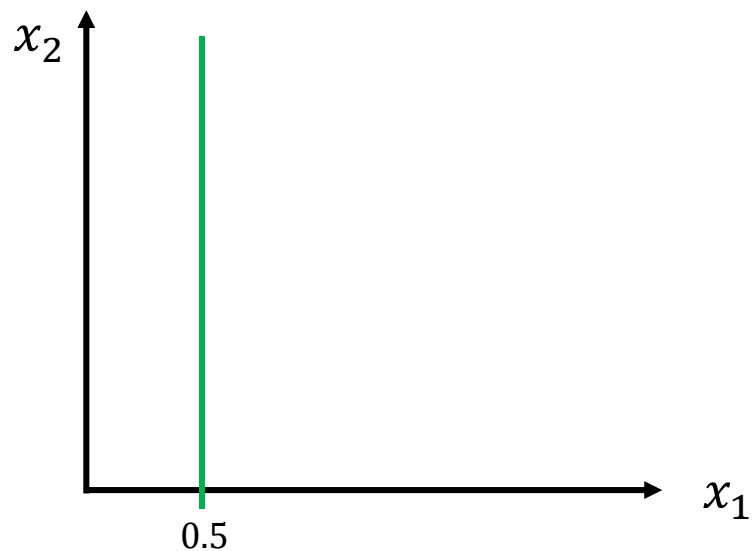
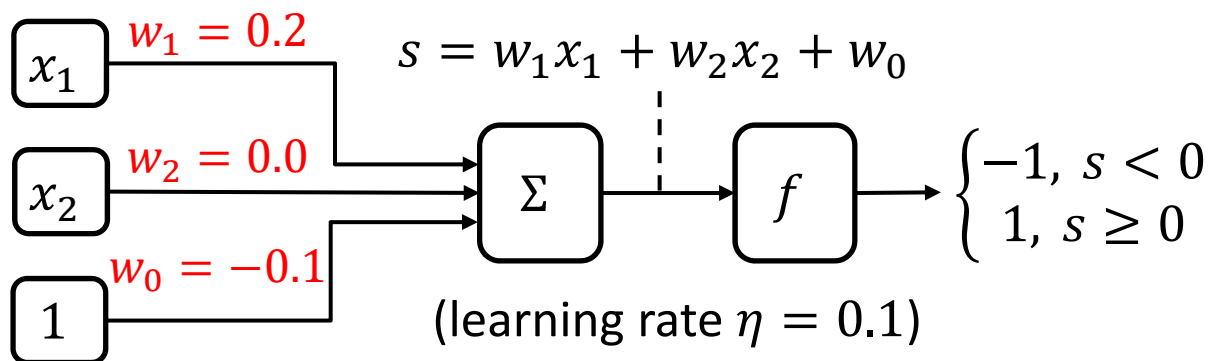
Perceptron learning example

Basic setup



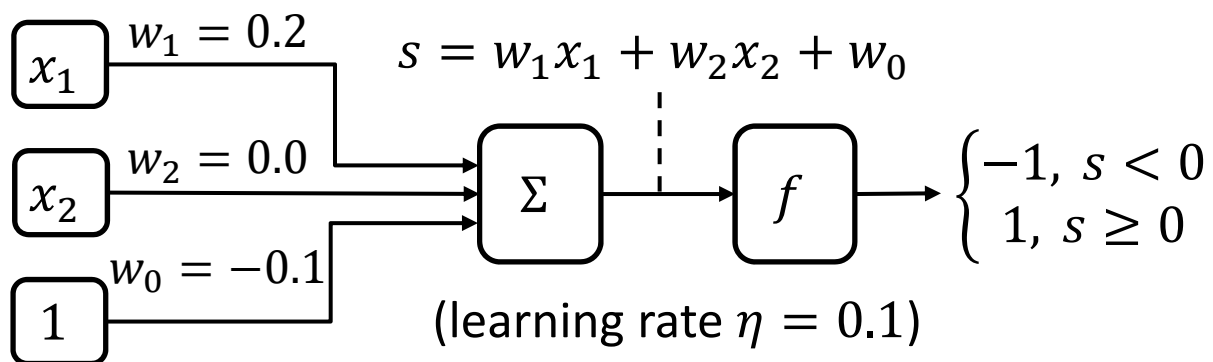
Perceptron learning example

Start with random weights



Perceptron learning example

Consider training example 1



$$\underline{0.2x_1 + 0.0x_2 - 0.1 > 0}$$

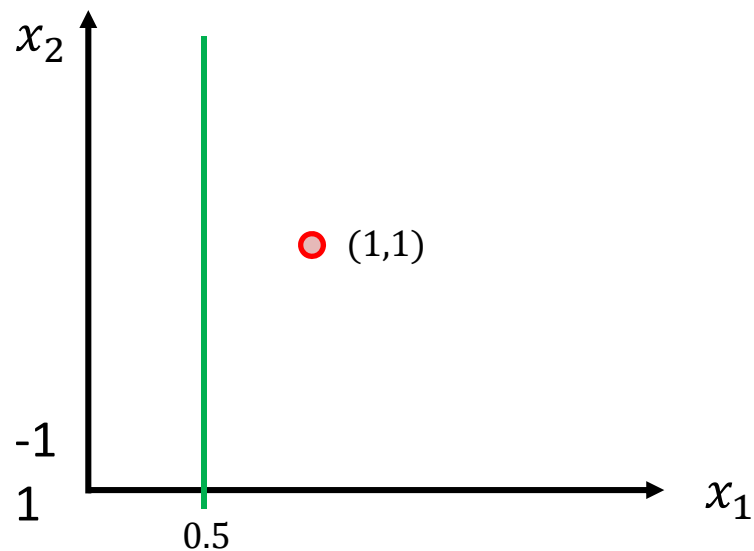
$$w_1 \leftarrow w_1 - \eta x_1 = 0.1$$

$$w_2 \leftarrow w_2 - \eta x_2 = -0.1$$

$$w_0 \leftarrow w_0 - \eta = -0.2$$

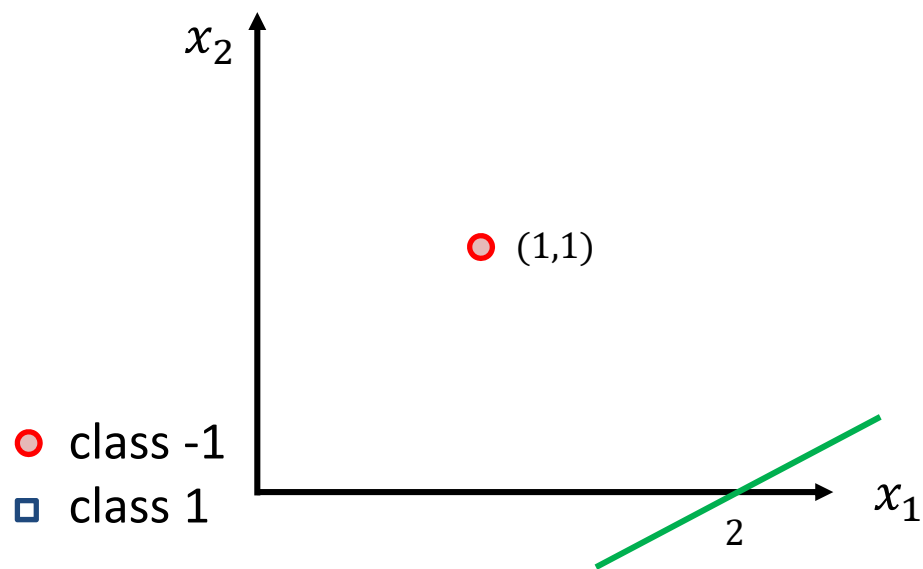
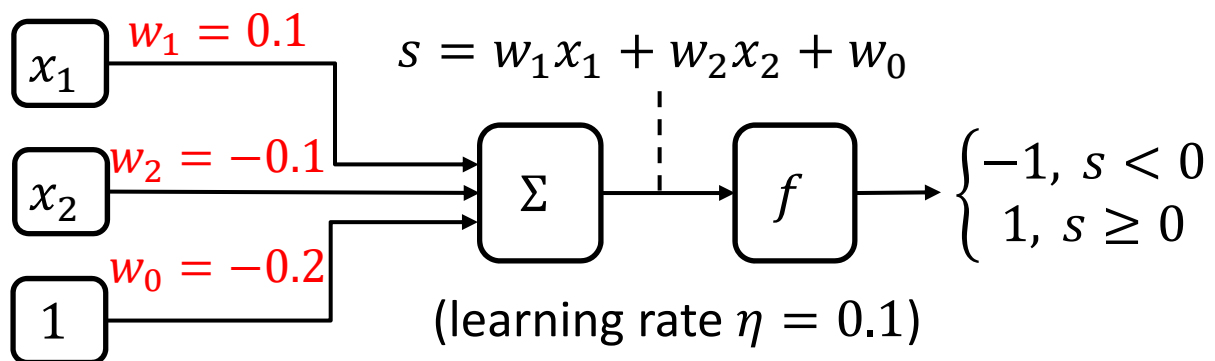
● class -1

■ class 1



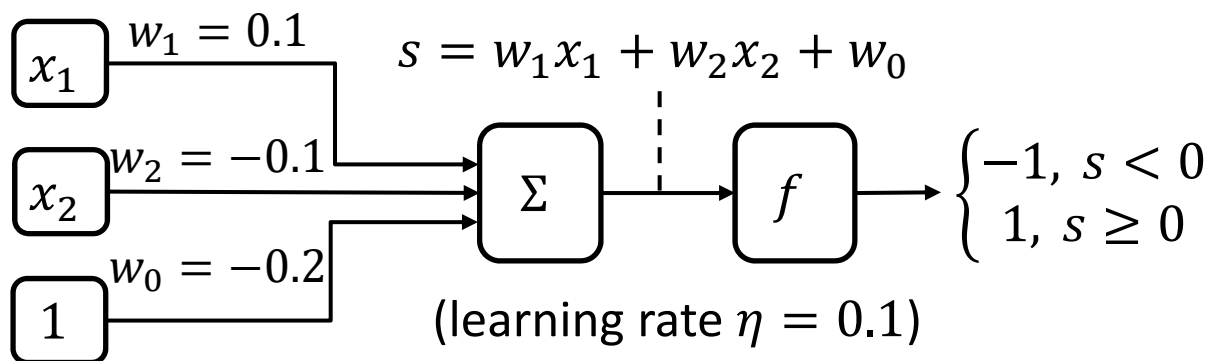
Perceptron learning example

Update weights



Perceptron learning example

Consider training example 2



$$\underline{0.1x_1 - 0.1x_2 - 0.2 < 0}$$

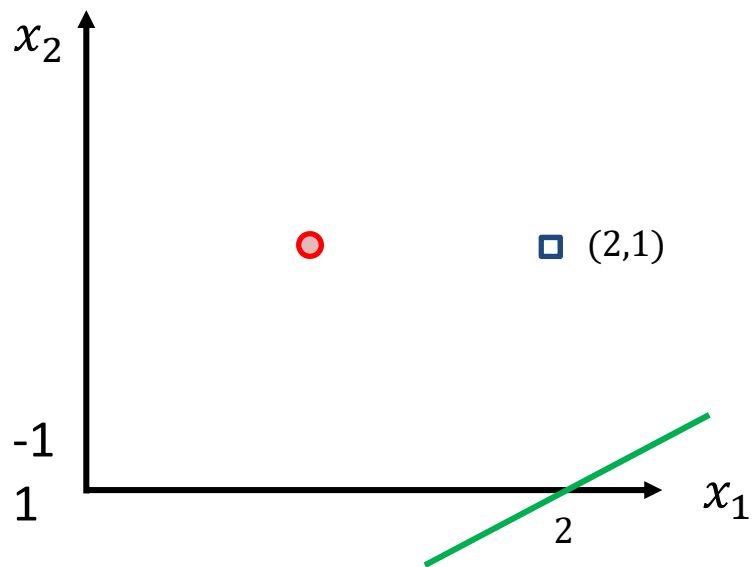
$$w_1 \leftarrow w_1 + \eta x_1 = 0.3$$

$$w_2 \leftarrow w_2 + \eta x_2 = 0.0$$

$$w_0 \leftarrow w_0 + \eta = -0.1$$

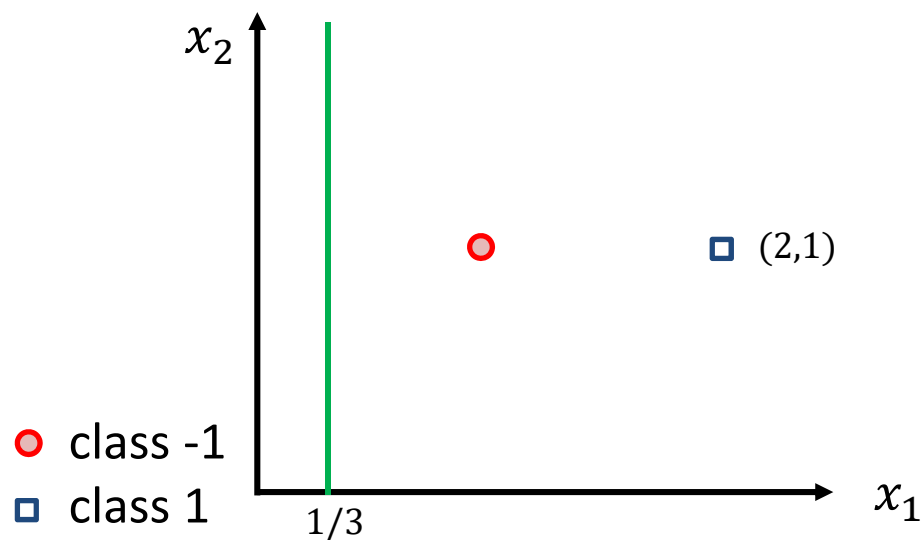
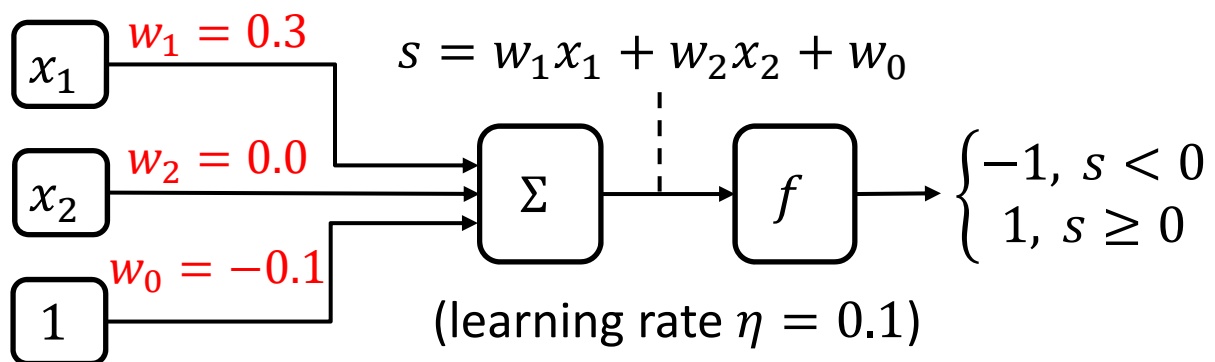
● class -1

■ class 1



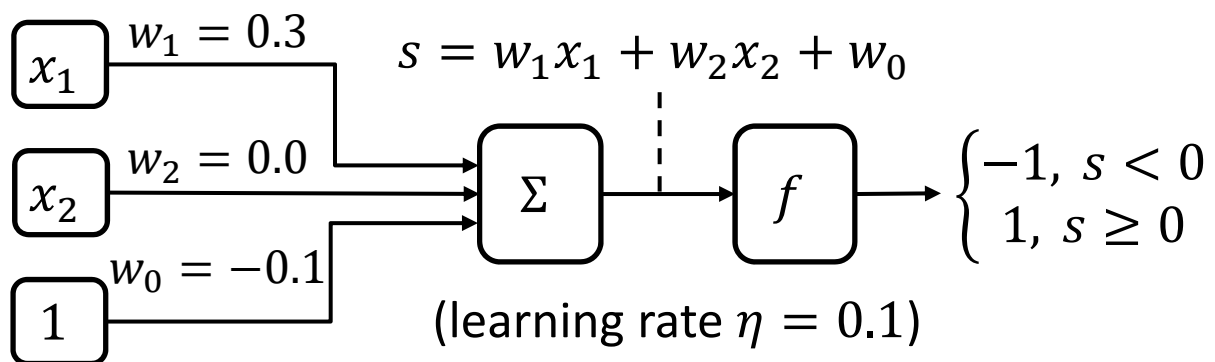
Perceptron learning example

Update weights

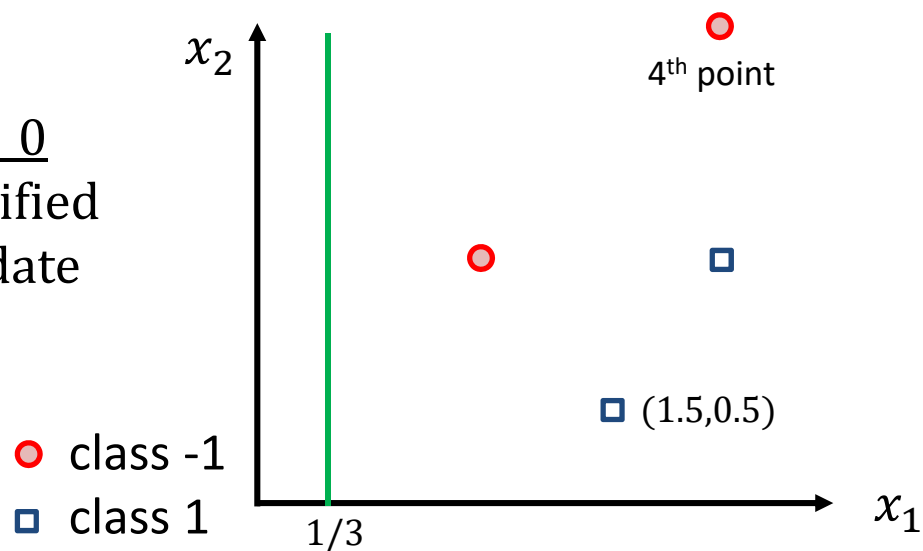


Perceptron learning example

Further examples

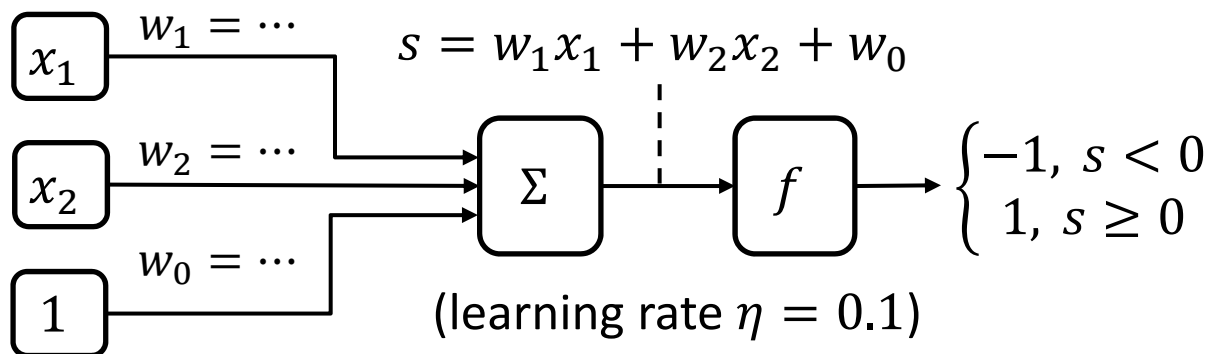


$\underline{0.3x_1 - 0.0x_2 - 0.1 > 0}$
 3rd point: correctly classified
 4th point: incorrect, update
 etc.

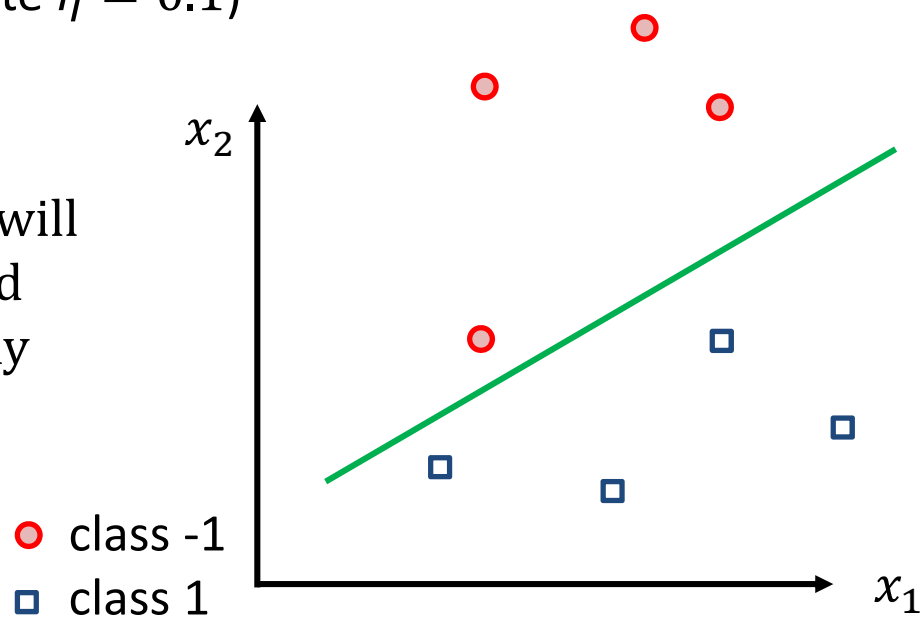


Perceptron learning example

Further examples



Eventually, all the data will
be correctly classified
(provided it is linearly
separable)



Summary

- Perceptron
 - * Introduction to Artificial Neural Networks
 - * The perceptron model
 - * Training algorithm
- **Workshop Week #4:** Exploring the perceptron
- Next lecture: Multiple layers, Backprop training