Supervised Learning

Chapter VII: Neural Networks 1

Johannes Jurgovsky

Outline

Neural Networks 1

1. Overview

2. The Perceptron

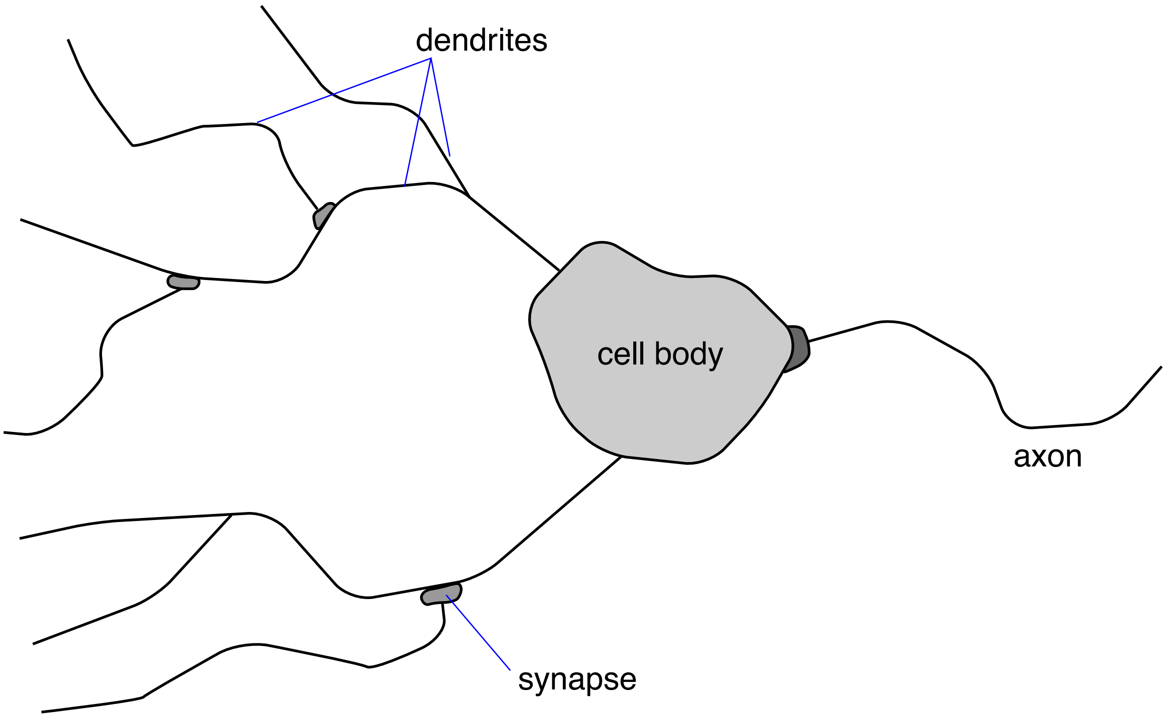
3. Logistic Regression

1. Overview

Overview: Biological Model

The Biological Model

Simplified model of a neuron:



Overview: Biological Model

The Biological Model (continued)

Neuron characteristics:

- The numerous dendrites of a neuron serve as its input channels for electrical signals.
- At particular contact points between the dendrites, the so-called synapses, electrical signals can be initiated.
- A synapse can initiate signals of different strengths, where the strength is encoded by the frequency of a pulse train.
- The cell body of a neuron accumulates the incoming signals.
- If a particular stimulus threshold is exceeded, the cell body generates a signal, which is output via the axon.
- The processing of the signals is unidirectional. (from left to right in the figure)

Overview: Biological Model

The Biological Model (continued)

Some facts about the human brain:

- The human brain is comprised of 10^{11} neurons
- The lengths of a dendrite or an axon is about 100 micron. A micron = $10^{-6}m = 10^{-3}mm$.
- The dendrites of a neuron are connected to 100.000 - 200.000 other neurons.
- An axon is connected to up to 10.000 synapses of other neurons.
- The human contains about 10^{12} synapses.
- The switching of a neuron is not faster than 10^{-3}s

Overview: History

History

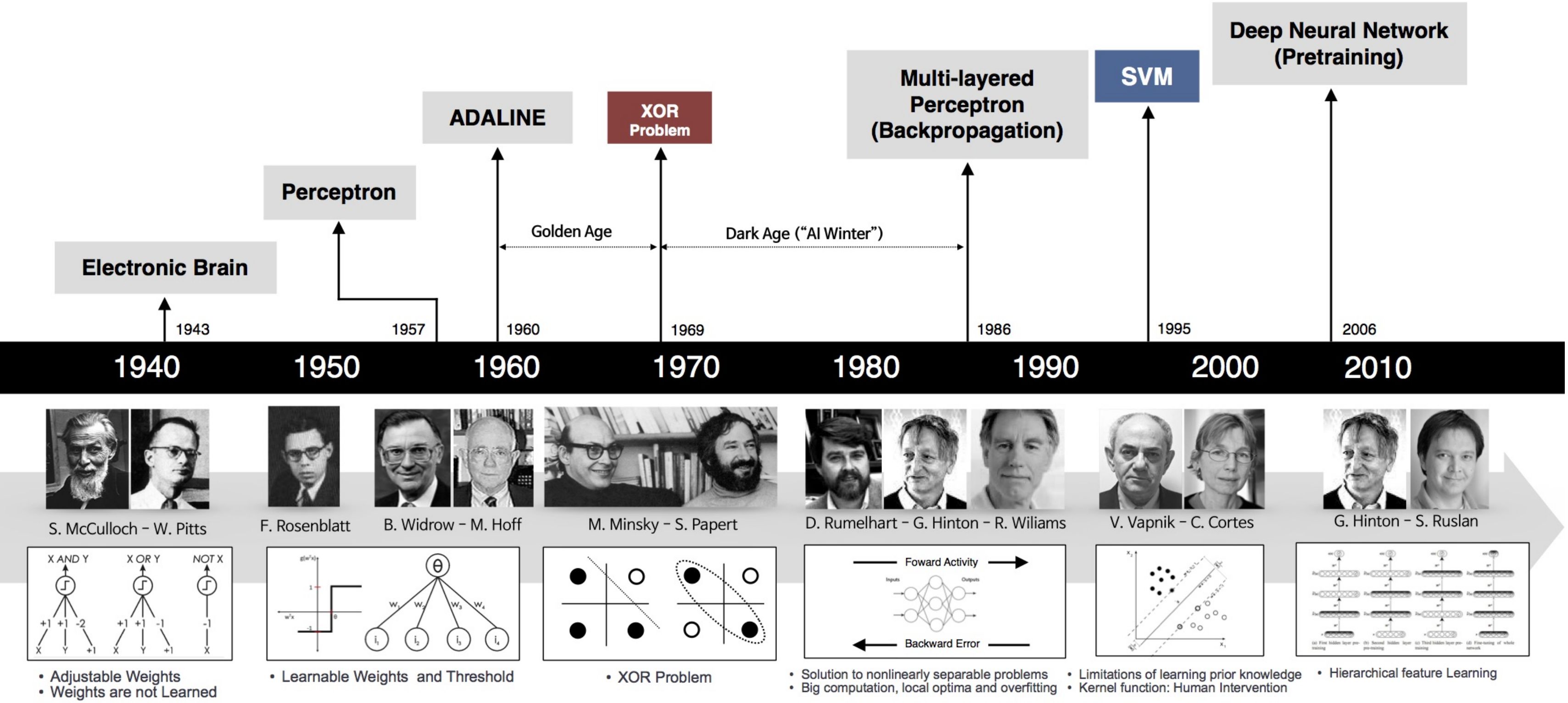


Image Source: [Bea17]

Overview: History

Breakthroughs since 2012

Today, Neural Networks (i.e. Deep Learning) enable incredible applications (e.g. Speech Interfaces, Autonomous Driving, Automatic Translation). Several reasons led to that:

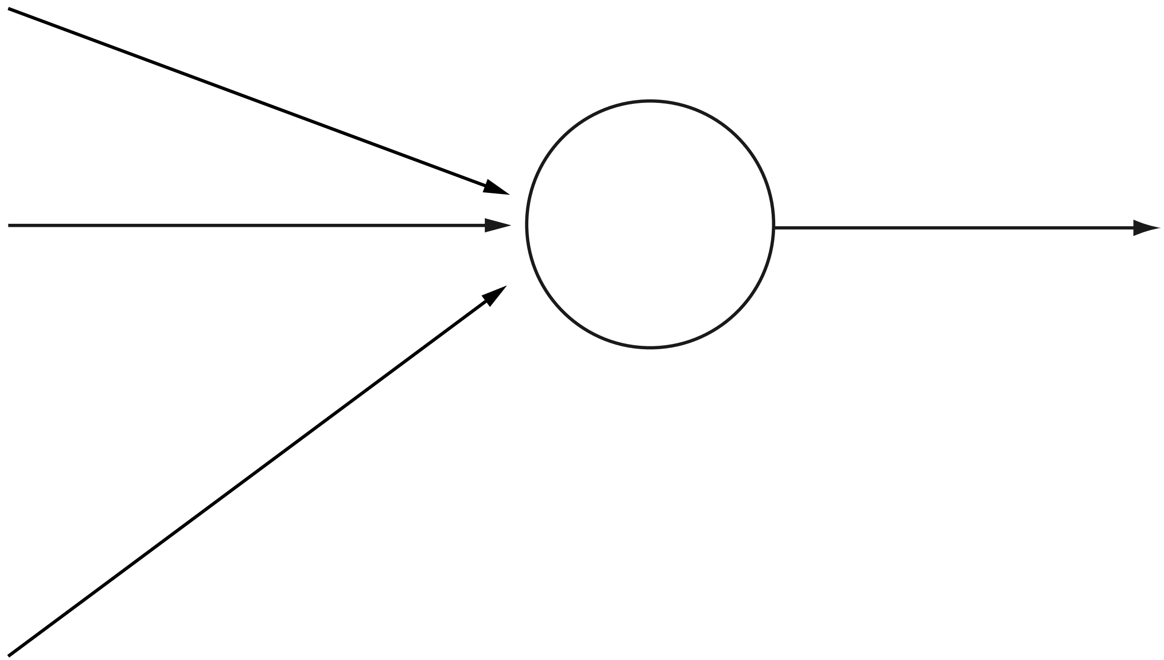
- **More Data:** New, labeled large-scale data sets (e.g. ImageNet)
- **More Computing Power:** More powerful GPUs with NN-optimized hardware, being able to run through big data sets
- **Algorithmic Advancements / Heuristics:**
  - New activation functions (i.e. RELU) and initialization procedures (i.e. Gloroth initialization) improving backpropagation
  - New network architectures (e.g. ResNet, Transformer Networks)
  - New regularization techniques (e.g. Dropout, Batch Normalization)
  - Robust and efficient optimizers (i.e. RMSProp, Adam)
  - New software platforms - Machine Learning for “everybody”

$\Rightarrow$ Success based mostly on engineering, not on theoretical justifications.

2. The Perceptron

The Perceptron: Perceptron Learning

The Perceptron of Rosenblatt [1958]

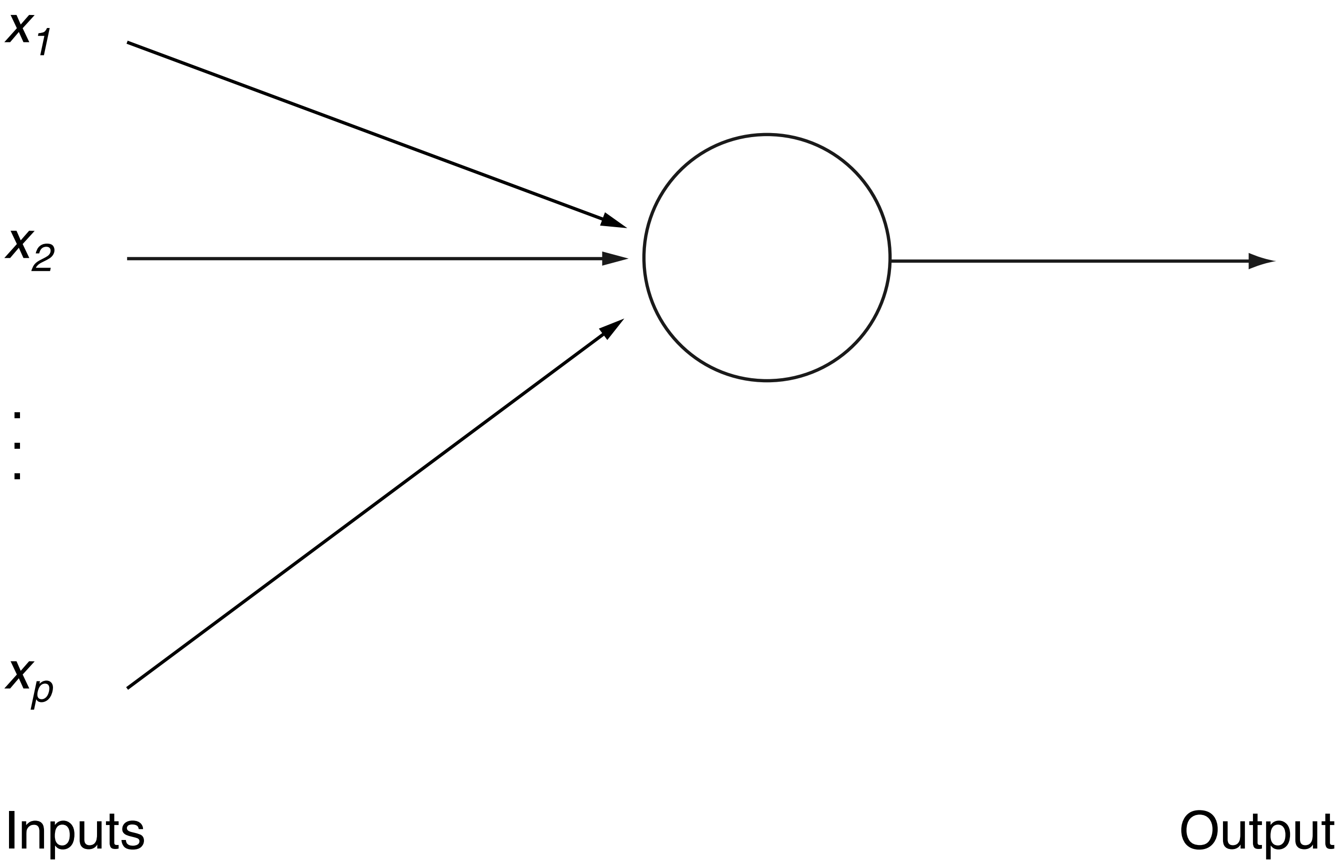


Inputs

Output

The Perceptron: Perceptron Learning

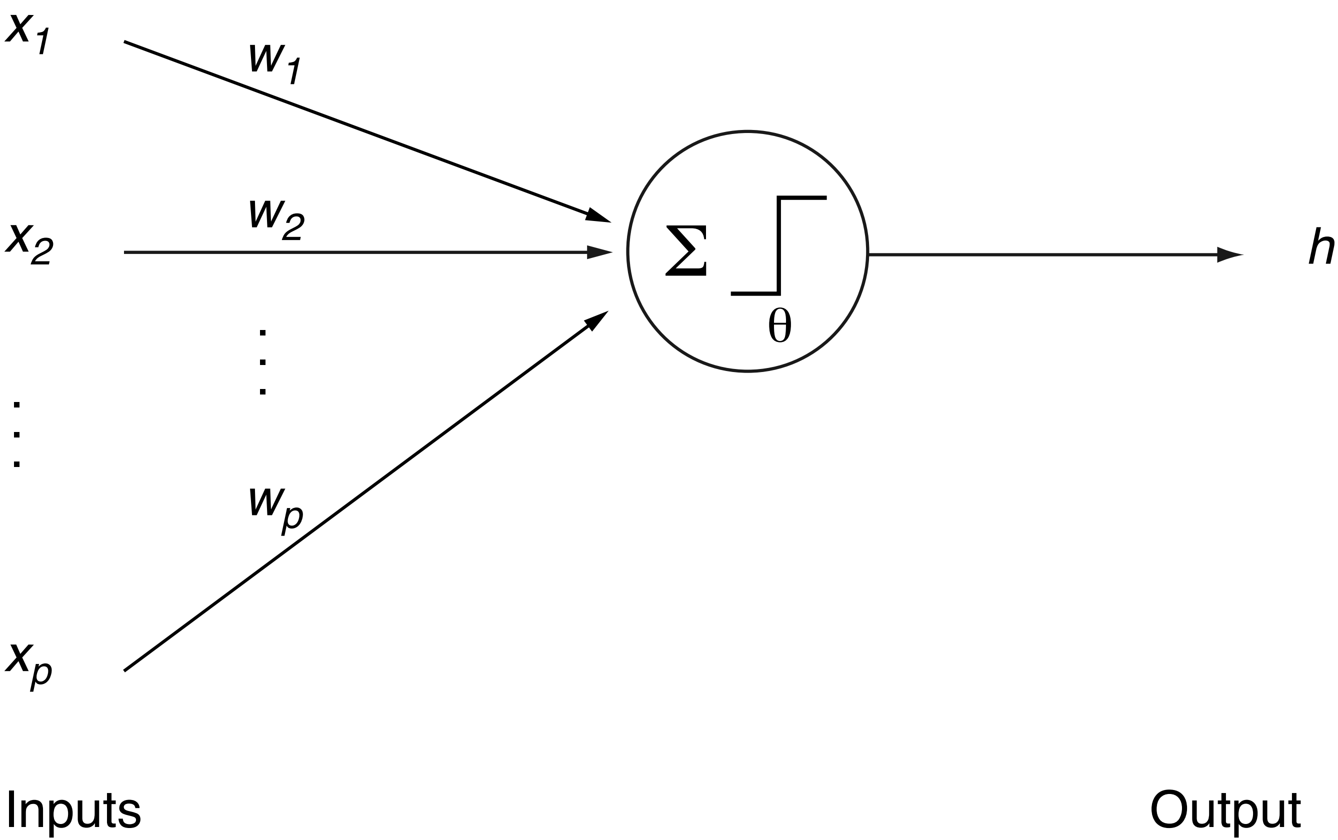
The Perceptron of Rosenblatt [1958]



$$x_j, w_j \in \mathbf{R}, \quad j = 1 \ldots p$$

The Perceptron: Perceptron Learning

The Perceptron of Rosenblatt [1958]



$$x_j, w_j \in \mathbf{R}, \quad j = 1 \ldots p$$

Remarks:

- The perceptron of Rosenblatt is based on the neuron model of McCulloch and Pitts.
- The perceptron is a “feed forward system”.

The Perceptron: Perceptron Learning

Specification of Classification Problems

Characterization of the model (model world):

- X is a set of feature vectors, also called feature space. $X \subseteq \mathbf{R}^p$
- Y is a set of classes. $Y = \{0, 1\}$
- $y : X \rightarrow Y$ is the ideal classifier for X.
- $D = \{(\mathbf{x}_1, y(\mathbf{x}_1)), (\mathbf{x}_2, y(\mathbf{x}_2)) \ldots, (\mathbf{x}_n, y(\mathbf{x}_n))\} \subseteq X \times Y$ is a set of examples.

How could the hypothesis space H look like?

The Perceptron: Perceptron Learning

Computation in the Perceptron

If $\sum_{j=1}^{p} w_j x_j \geq \theta$ then $h(\mathbf{x}) = 1$, and

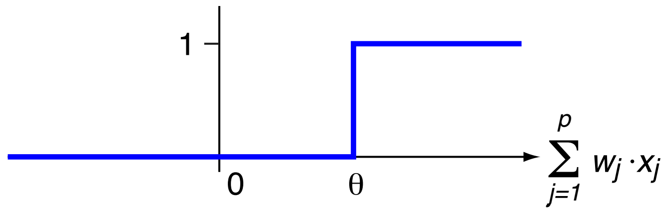
if $\sum_{j=1}^{p} w_j x_j < \theta$ then $h(\mathbf{x}) = 0$.

The Perceptron: Perceptron Learning

Computation in the Perceptron

If $\sum_{j=1}^{p} w_j x_j \geq \theta$ then $h(\mathbf{x}) = 1$, and

if $\sum_{j=1}^{p} w_j x_j < \theta$ then $h(\mathbf{x}) = 0$.



where $\sum_{j=1}^{p} w_j x_j = \mathbf{w}^T \mathbf{x}$. (or other notations for the inner product)

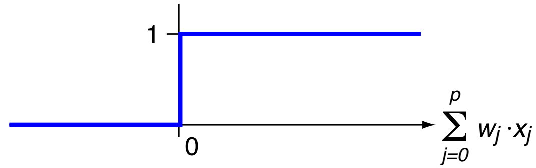
➜ A hypothesis is determined by $\theta, w_1, \ldots, w_p$.

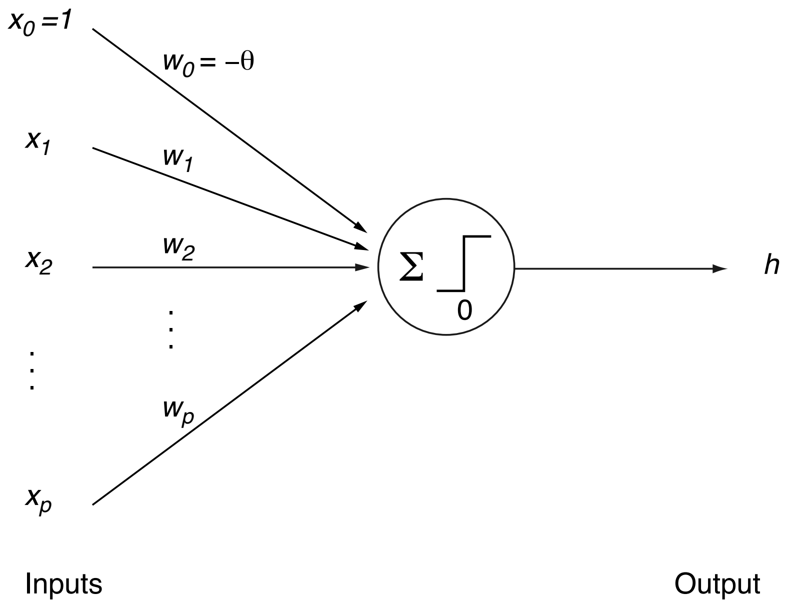
The Perceptron: Perceptron Learning

Computation in the Perceptron (continued)

$$h(\mathbf{x}) = \textit{heaviside}(\sum_{j=1}^{p} w_j x_j - \theta)$$

$$= \textit{heaviside}(\sum_{j=0}^{p} w_j x_j) \quad \text{with } w_0 = -\theta,\ x_0 = 1$$





➜ A hypothesis is determined by $w_0, w_1, \ldots, w_p$.

Remarks:

- If the weight vector is extended by $w_0 = -\theta$, and, if the feature vectors are extended by the constant feature $x_0 = 1$, the learning algorithm gets a canonical form. Implementations of neural networks introduce this extension often implicitly.
- Be careful with regard to the dimensionality of the weight vector: it is always denoted as $\mathbf{w}$ here, irrespective of the fact whether the w_0-dimension, with $w_0 = -\theta$, is included.
- The function *heaviside* is named after the mathematician Oliver Heaviside.
  [Heaviside: step function Oliver]

The Perceptron: Perceptron Learning

Weight Adaptation [*IGD* Algorithm]

Algorithm:	*PT*	Perceptron Training
Input:	D	Training examples of the form $(\mathbf{x}, y(\mathbf{x}))$ with $\lvert\mathbf{x}\rvert = p+1,\ \ y(\mathbf{x}) \in \{0,1\}$.
	η	Learning rate, a small positive constant.
Internal:	$h(D)$	Set of $h(\mathbf{x})$-values computed from the elements $\mathbf{x}$ in D given some $\mathbf{w}$.
Output:	$\mathbf{w}$	Weight vector.

PT(D, η)

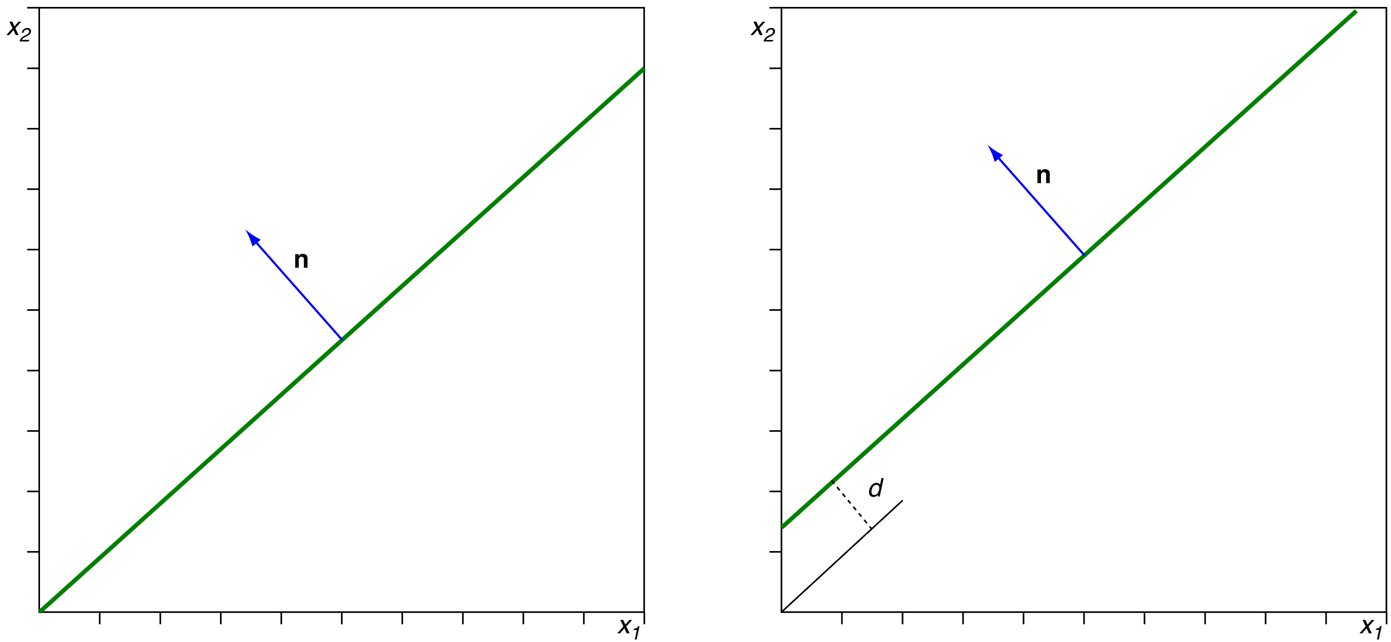
initialize_random_weights$(\mathbf{w})$, $t = 0$
REPEAT
    $t = t + 1$
    $(\mathbf{x}, y(\mathbf{x})) =$ *random_select*(D)
    error $= y(\mathbf{x}) -$ *heaviside*$(\mathbf{w}^T\mathbf{x})$
    FOR $j = 0$ **TO** p **DO**
        $\Delta w_j = \eta \cdot$ *error* $\cdot\, x_j$
        $w_j = w_j + \Delta w_j$
    ENDDO
UNTIL(*convergence*$(D, h(D))$ **OR** $t > t_{\max}$)
return$(\mathbf{w})$

Remarks:

- The variable t denotes the time. At each point in time the learning algorithm gets an example presented and, as a consequence, may adapt the weight vector.
- The weight adaptation rule compares the true class $y(\mathbf{x})$ (the ground truth) to the class computed by the perceptron. In case of a wrong classification of a feature vector $\mathbf{x}$, *Err* is either -1 or $+1$—independent of the exact numeric difference between $y(\mathbf{x})$ and $\mathbf{w}^T\mathbf{x}$.
- $h(\cdot)$ denotes the current hypothesis

The Perceptron: Perceptron Learning

Weight Adaptation (continued)

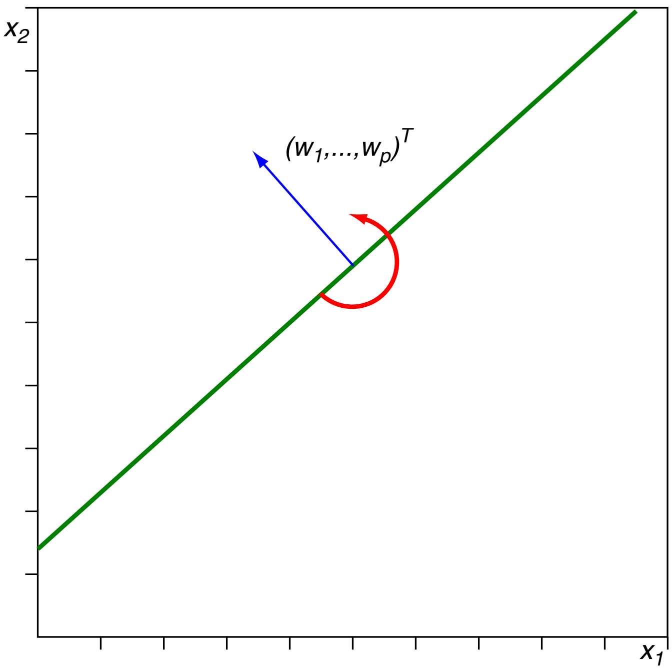


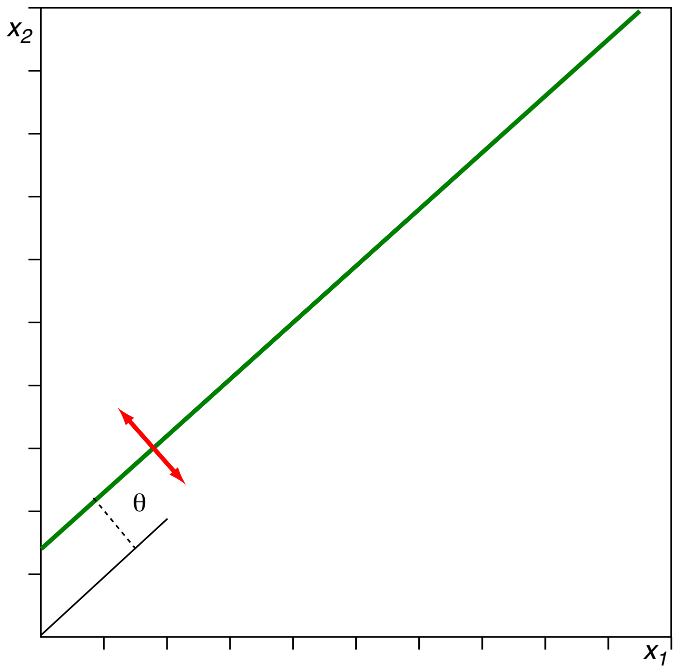
Definition of an hyperplane: $\mathbf{n}^T\mathbf{x} = d$ [Wikipedia]

- $\mathbf{n}$ denotes a normal vector that is perpendicular to the hyperplane.
- If $||\mathbf{n}|| = 1$ then $|d|$ corresponds to the distance of the origin to the hyperplane.
- If $\mathbf{n}^T\mathbf{x} < d$ and $d \geq 0$ then $\mathbf{x}$ and the origin lie on the same side of the hyperplane.

The Perceptron: Perceptron Learning

Weight Adaptation (continued)





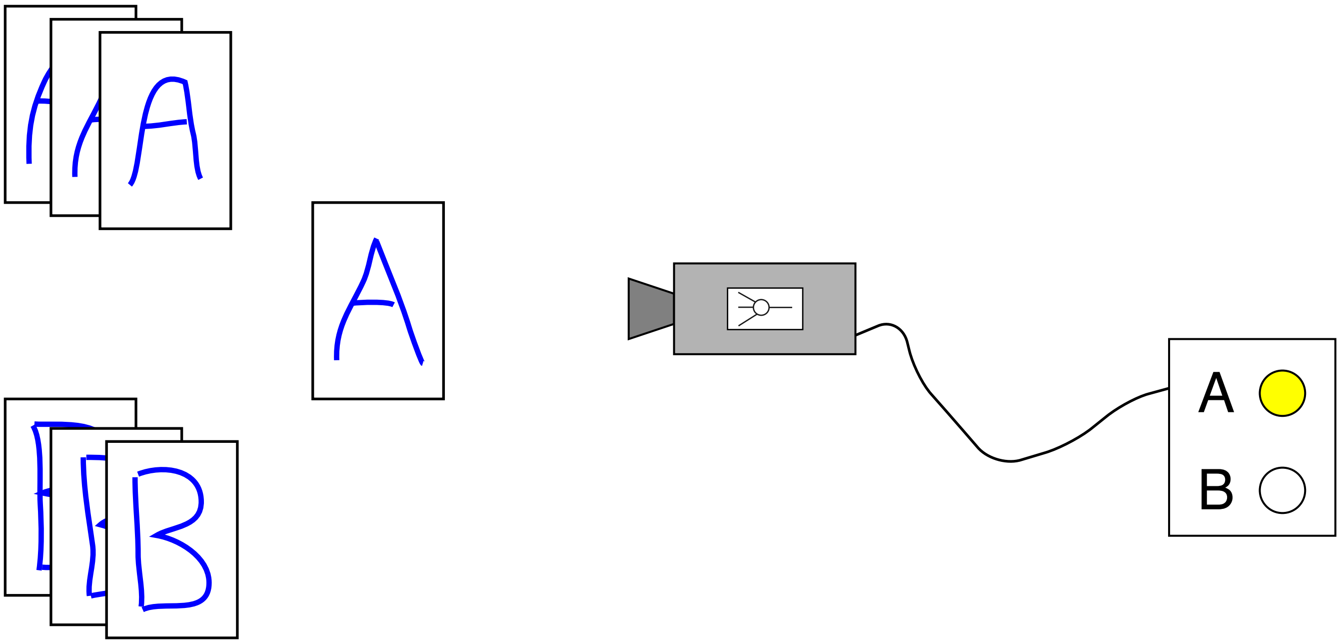
Definition of an hyperplane: $\mathbf{w}^T\mathbf{x} = 0 \;\Leftrightarrow\; \sum_{j=1}^{p} w_j x_j = \theta = -w_0$

Remarks:

- A perceptron defines a hyperplane that is perpendicular (= normal) to $(w_1, \ldots, w_p)^T$.
- θ or $-w_0$ specify the offset of the hyperplane from the origin, along $(w_1, \ldots, w_p)^T$ and as multiple of $1/||(w_1, \ldots, w_p)^T||$.
- The set of possible weight vectors $\mathbf{w} = (w_0, w_1, \ldots, w_p)^T$ form the hypothesis space H.
- Weight adaptation means learning, and the shown learning paradigm is supervised.
- The computation of the weight difference Δw_j in Line 7 of the *PT* Algorithm considers the feature vector $\mathbf{x}$ componentwise. In particular, if some x_j is zero, Δw_j will be zero as well. Keyword: Hebbian learning [Hebb 1949]

The Perceptron: Perceptron Learning

Illustration



- The examples are presented to the perceptron.
- The perceptron computes a value that is interpreted as class label.

The Perceptron: Perceptron Learning

Illustration (continued)

Encoding:

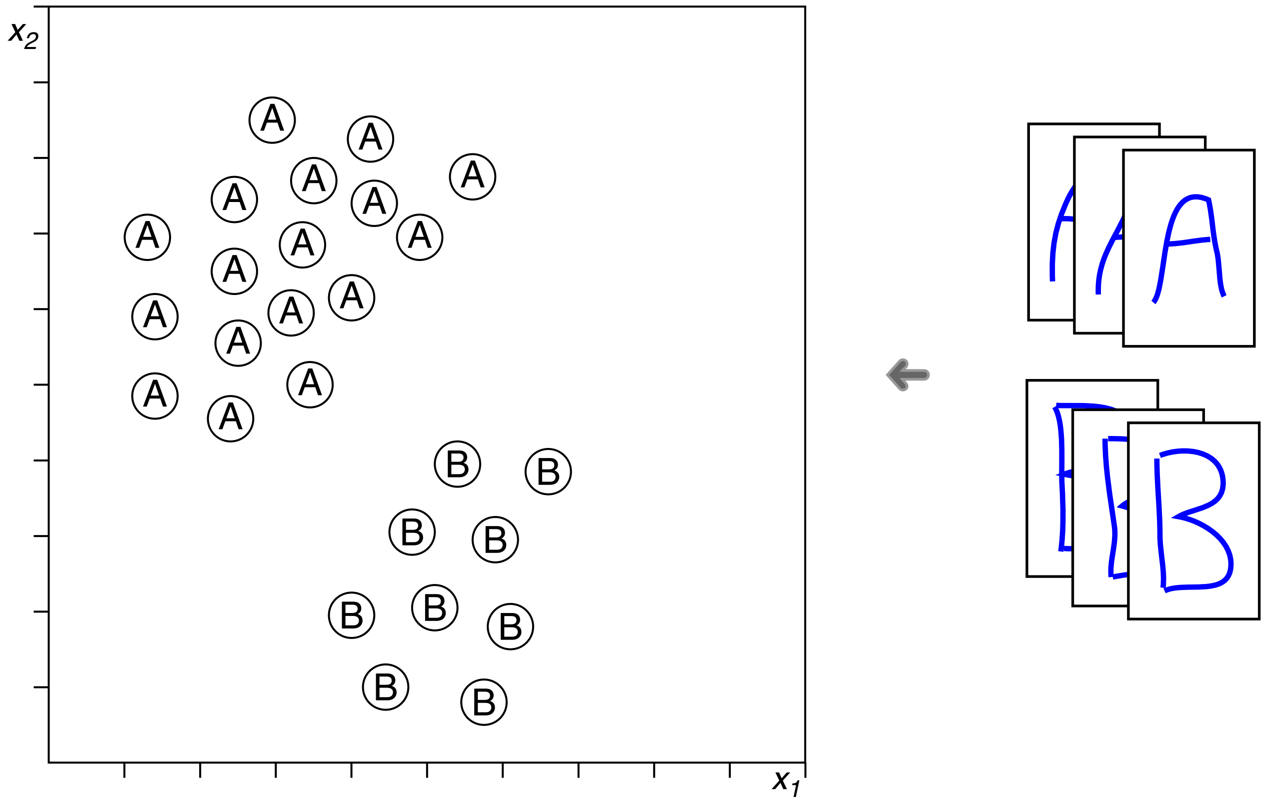
- The encoding of the examples is based on expressive features: number of line crossings, most acute angle, longest line, etc.
- The class label, $y(\mathbf{x})$, is encoded as a number. Examples from A are labeled with 1, examples from B are labeled with 0.

$$\underbrace{\begin{pmatrix} x_{1_1} \\ x_{1_2} \\ \vdots \\ x_{1_p} \end{pmatrix} \quad \ldots \quad \begin{pmatrix} x_{k_1} \\ x_{k_2} \\ \vdots \\ x_{k_p} \end{pmatrix}}_{\text{Class } A \;\simeq\; y(\mathbf{x}) = 1} \qquad \underbrace{\begin{pmatrix} x_{l_1} \\ x_{l_2} \\ \vdots \\ x_{l_p} \end{pmatrix} \quad \ldots \quad \begin{pmatrix} x_{m_1} \\ x_{m_2} \\ \vdots \\ x_{m_p} \end{pmatrix}}_{\text{Class } B \;\simeq\; y(\mathbf{x}) = 0}$$

The Perceptron: Perceptron Learning

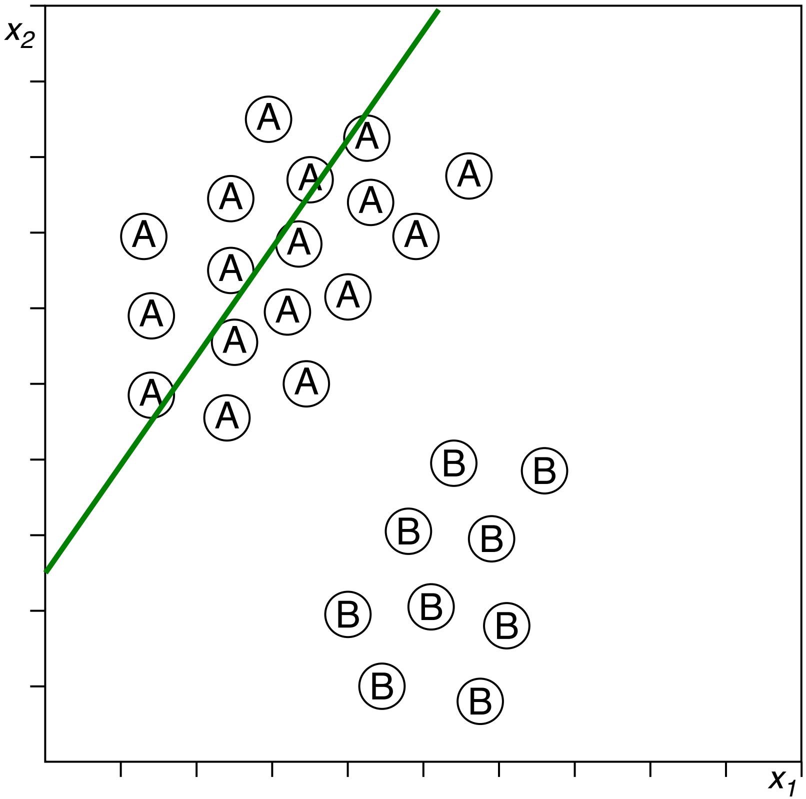
Illustration (continued)

A possible configuration of encoded objects in the feature space X :



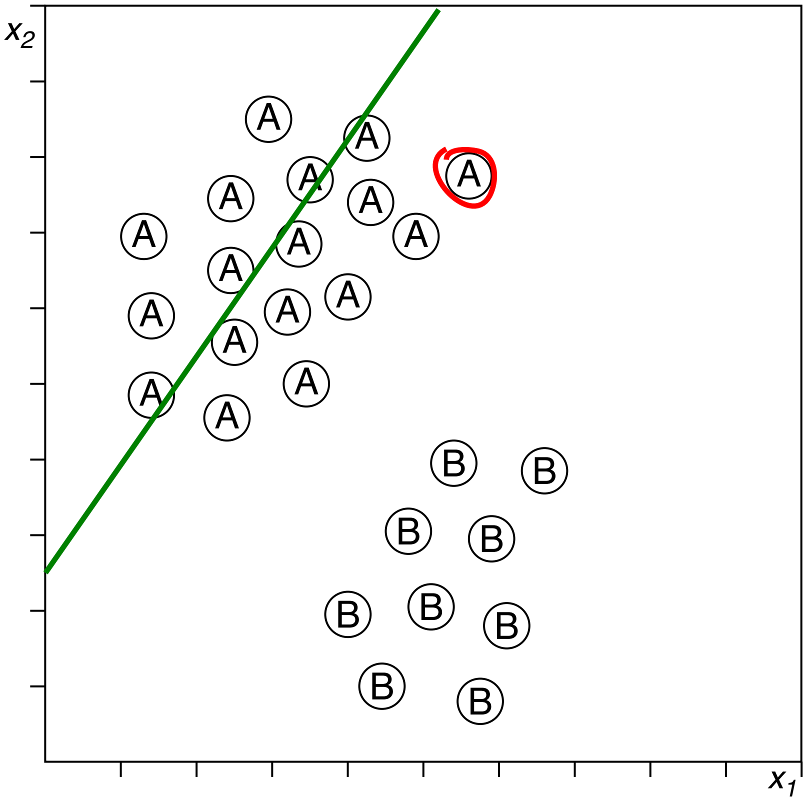
The Perceptron: Perceptron Learning

Illustration (continued) [*PT* Algorithm]



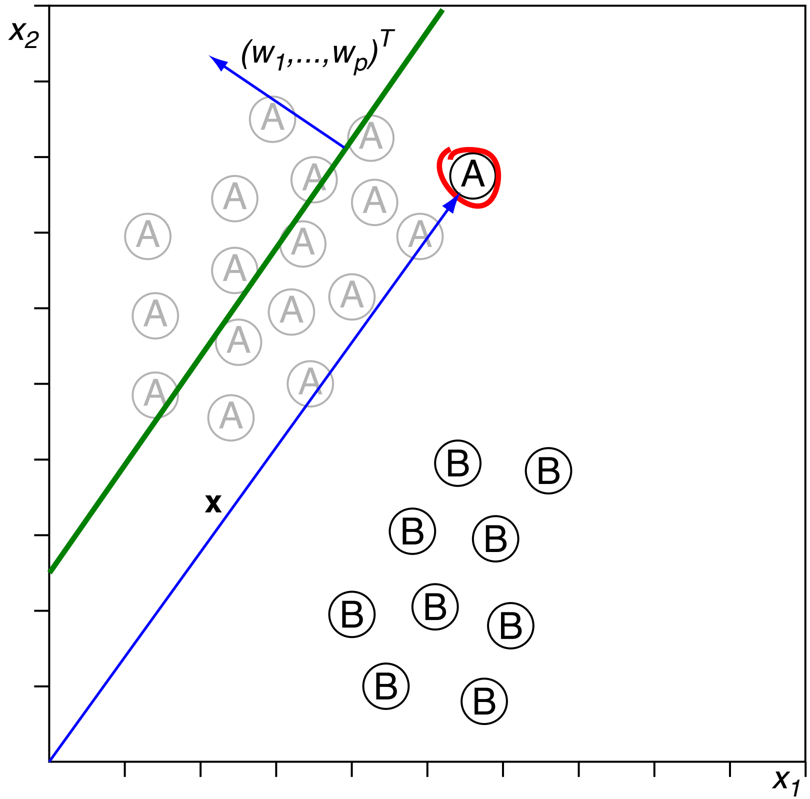
The Perceptron: Perceptron Learning

Illustration (continued) [*PT* Algorithm]



The Perceptron: Perceptron Learning

Illustration (continued) [*PT* Algorithm]



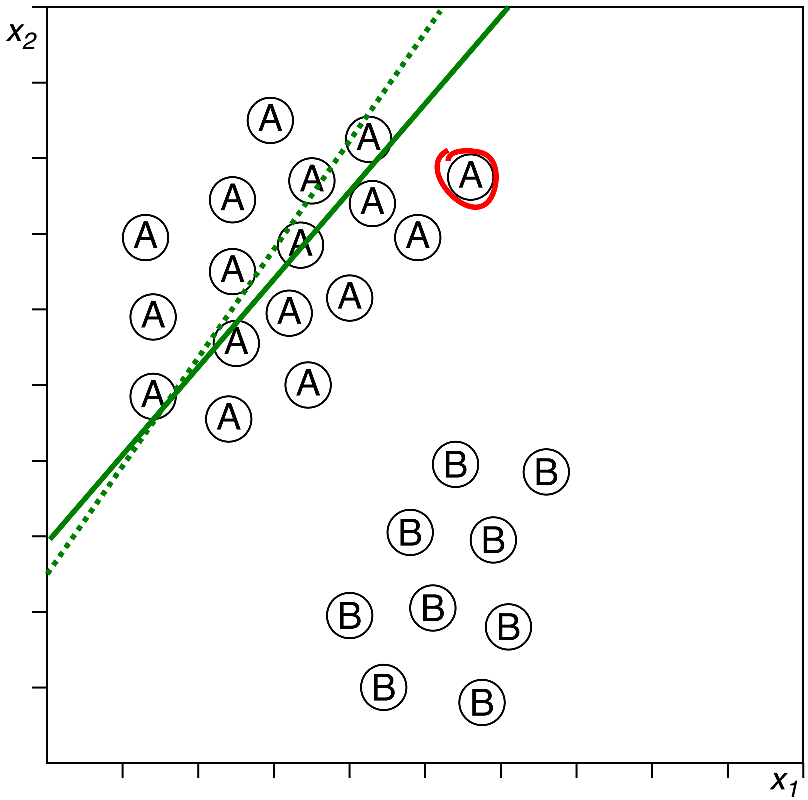
We predict $h(\mathbf{x}) = 0$ (B), instead of $y(\mathbf{x}) = 1$ (A)

$\Rightarrow$ *error* $= 1 - 0 > 0$

$\Rightarrow$ *Update* $\mathbf{w}$ *proportional to* $\mathbf{x}$

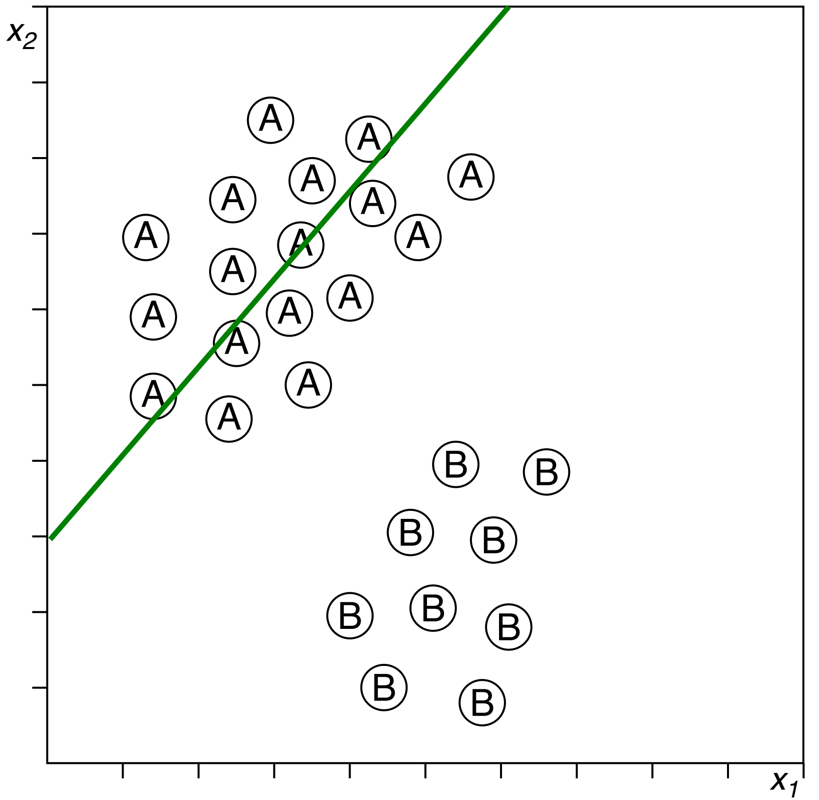
The Perceptron: Perceptron Learning

Illustration (continued) [PT Algorithm]



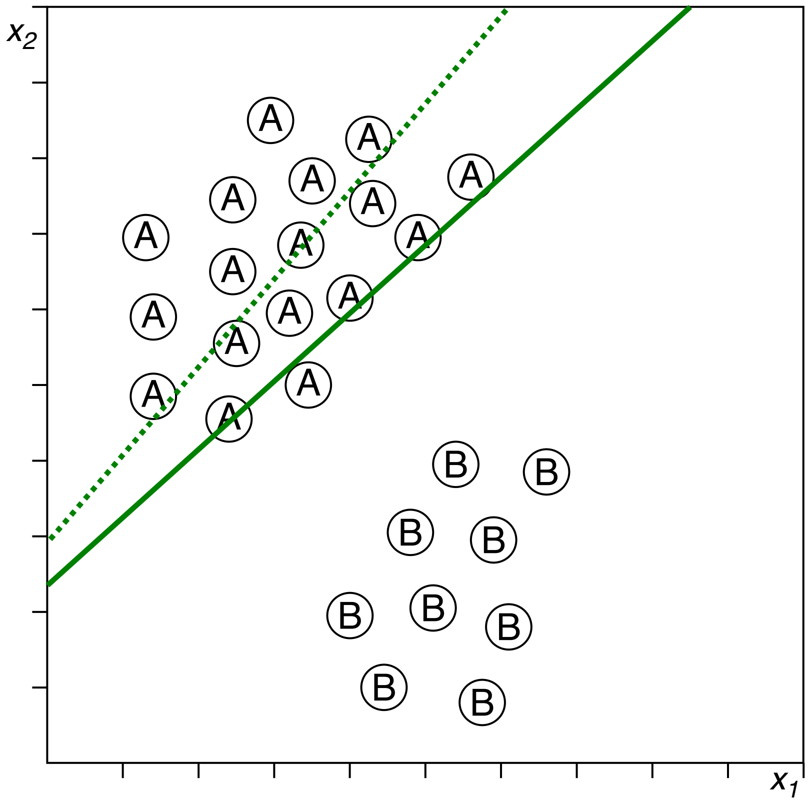
The Perceptron: Perceptron Learning

Illustration (continued) [*PT* Algorithm]



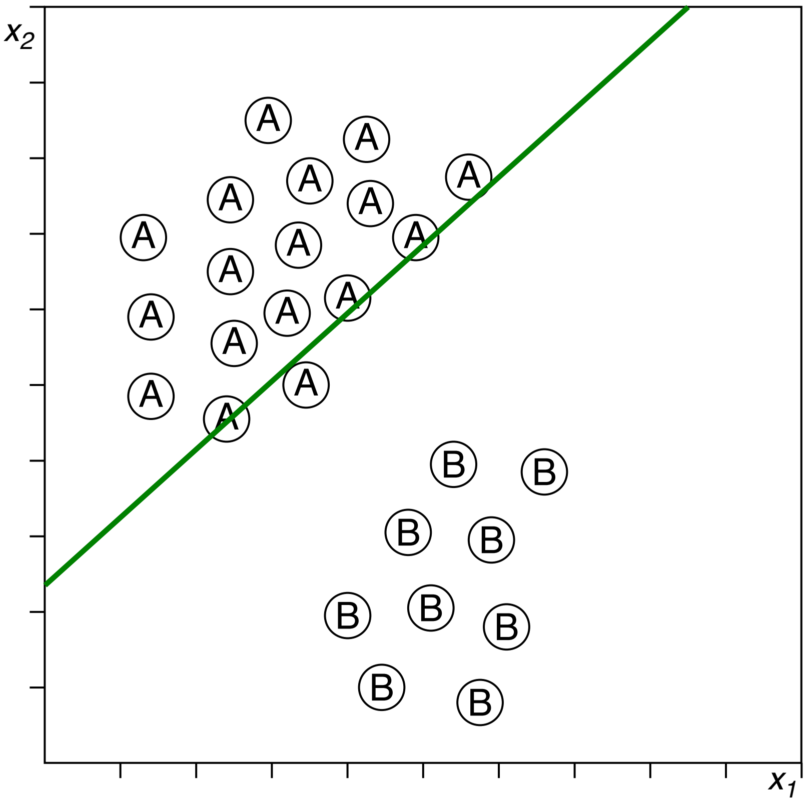
The Perceptron: Perceptron Learning

Illustration (continued) [*PT* Algorithm]



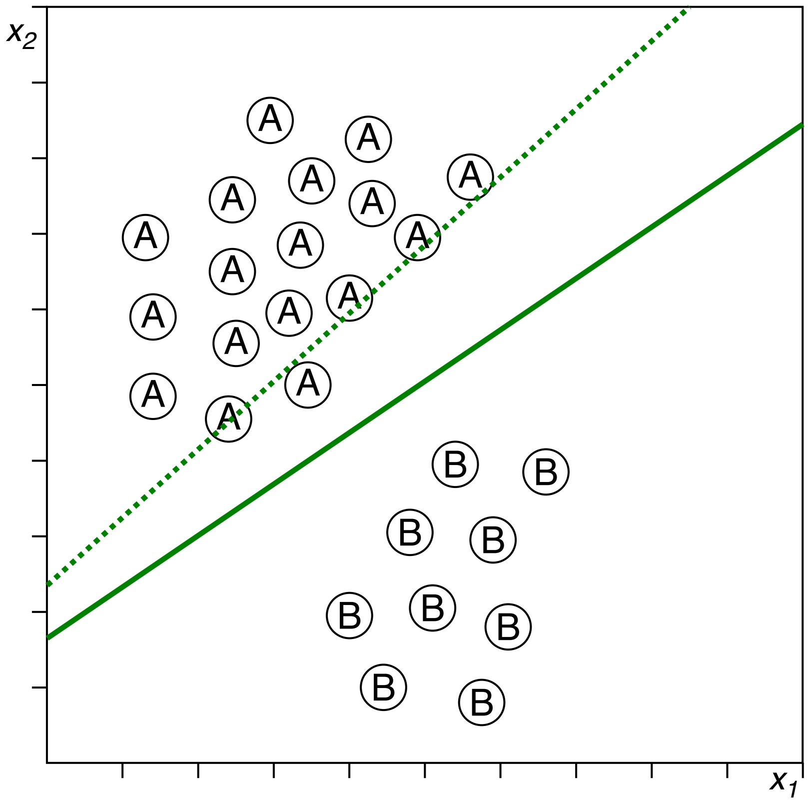
The Perceptron: Perceptron Learning

Illustration (continued) [*PT* Algorithm]



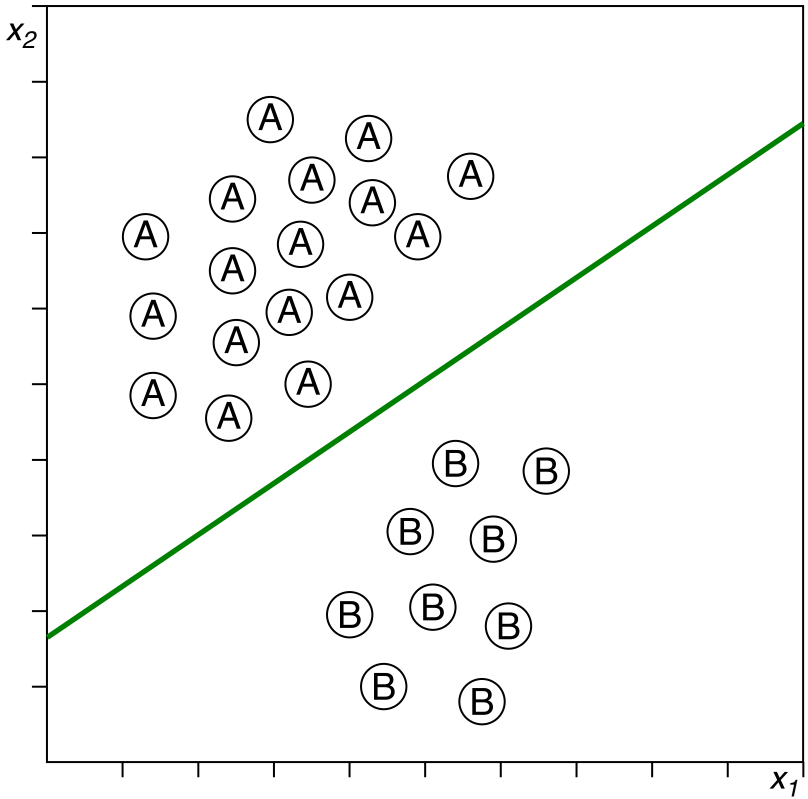
The Perceptron: Perceptron Learning

Illustration (continued) [*PT* Algorithm]



The Perceptron: Perceptron Learning

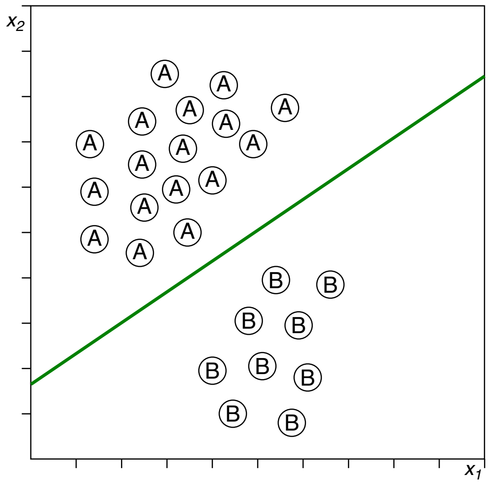
Illustration (continued) [*PT* Algorithm]

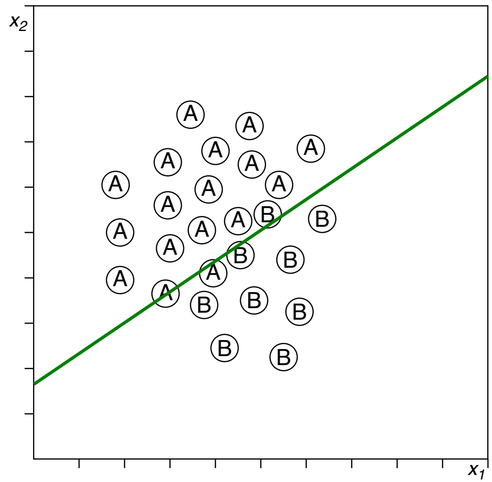


The Perceptron: PT Discussion

Perceptron Convergence Theorem: Discussion

- If there exists a solution (i.e. the training data set is linearly separable), the *PT* Algorithm will find the solution in a finite number of steps. If the data set is not linearly separable, then the algorithm will not converge.





- When the data set is linearly separable, there are many solutions. It depends on the order of the updates and the initialization of the parameters which solution is found.

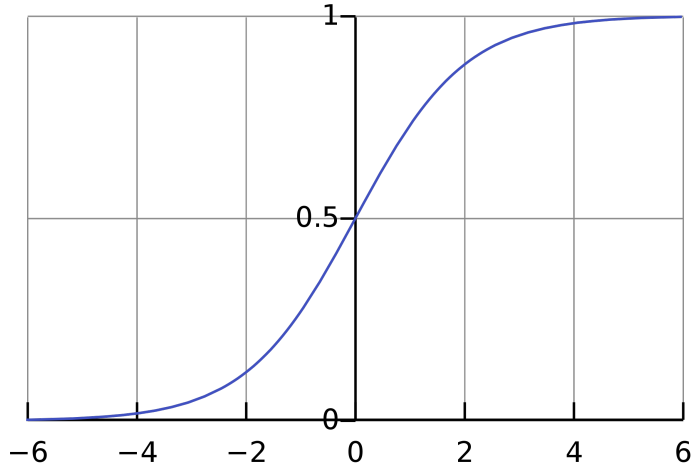
3. Logistic Regression

Logistic Regression:
Logistic Function

Logistic Regression is a (binary) *discriminative* classifier using the *logistic function* as an activation function.

Instead of predicting the class labels $y \in \{0, 1\}$ directly, Logistic Regression uses the logistic function to model the class-posterior probabilities $p(y = 1|\mathbf{x})$.

- Logistic function (also called *sigmoid*): $\sigma(x) = \frac{1}{1+e^{-x}}$
- Derivative: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- Values between 0 and 1 are interpreted as probabilities



Source: Wikipedia

Logistic Regression:
Parameterized Logistic Function

- Now: Mulit-dimensional feature vector $\mathbf{x}$

$$h(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

- where $\mathbf{w}$ are the parameters of a linear combination of the elements of the feature vector

  Only linear separation of classes possible

- The probabilities for binary classification are given by:

$$p(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$$
$$p(y = 0|\mathbf{x}) = 1 - \sigma(\mathbf{w}^T\mathbf{x})$$

- Training: Estimate parameter vector $\mathbf{w}$ to maximize the likelihood that an unknown feature vector is classified correctly
  - Labelled training set required
  - Method: Maximum-Likelihood Estimation

Logistic Regression:

Logistic Function and Posterior Probabilities

What is the relationship between posterior probabilities and the logistic function?

$$
\begin{aligned}
p(y=1|\mathbf{x}) &= \frac{p(y=1)p(\mathbf{x}|y=1)}{p(\mathbf{x})} \\
&= \frac{p(y=1)p(\mathbf{x}|y=1)}{p(y=0)p(\mathbf{x}|y=0)+p(y=1)p(\mathbf{x}|y=1)} \\
&= \frac{1}{1+\frac{p(y=0)p(\mathbf{x}|y=0)}{p(y=1)p(\mathbf{x}|y=1)}} \\
&\rightarrow \text{extend with exponential / logarithm} \\
&= \frac{1}{1+e^{\log\frac{p(y=0)p(\mathbf{x}|y=0)}{p(y=1)p(\mathbf{x}|y=1)}}} \\
&= \frac{1}{1+e^{-\left(\log\frac{p(y=1)}{p(y=0)}+\log\frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)}\right)}} \\
&= \frac{1}{1+e^{-f(\mathbf{x})}} = \sigma(f(\mathbf{x}))
\end{aligned}
$$

where we rewrite $f(\mathbf{x}) = \log\frac{p(y=1)}{p(y=0)} + \log\frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)}$

Logistic Regression:

Logistic Function and Posterior Probabilities

In Logistic Regression, $f(\mathbf{x})$ is chosen to be a linear combination of the input features: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

$$p(y=1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}} = \sigma(\mathbf{w}^T\mathbf{x})$$
$$p(y=0|\mathbf{x}) = 1 - p(y=1|\mathbf{x}) = \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}} = \sigma(-\mathbf{w}^T\mathbf{x})$$

Notice that we can merge the two equations into one:

$$p(y|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})^y \cdot (1-\sigma(\mathbf{w}^T\mathbf{x}))^{(1-y)}$$

Decision boundary between the two classes at $\mathbf{w}^T\mathbf{x} = 0$

$\rightarrow$ this is where $p(y=0|\mathbf{x}) = p(y=1|\mathbf{x}) = 0.5$

Logistic Regression:
Training

- Given n training examples $(\mathbf{x}_i, y_i) \in \mathcal{D}$
- Estimate parameter vector $\mathbf{w}$ to maximize the likelihood that a feature vector is classified correctly
- Likelihood
  - Assume that training examples are mutually independent
  - Likelihood function: $\prod_{i=1}^{n} p(y_i|\mathbf{x}_i)$
  - Maximize Likelihood function
- Exponential function $\rightarrow$ Maximize *Log-Likelihood* instead

$$\mathcal{L}(\mathbf{w}) = \log\left(\prod_{i=1}^{n} p(y_i|\mathbf{x}_i)\right) = \sum_{i=1}^{n} \log p(y_i|\mathbf{x}_i)$$

log is a monotonous function $\rightarrow$ Maximum stays the same

Logistic Regression:
Log-Likelihood

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \sum_{i=1}^{n} \log p(y_i|\mathbf{x}_i) \\
&= \sum_{i=1}^{n} \log \left(\sigma(\mathbf{w}^T\mathbf{x}_i)^{y_i} \cdot (1-\sigma(\mathbf{w}^T\mathbf{x}_i))^{(1-y_i)} \right) \\
&= \dots \\
&= \sum_{i=1}^{n} (y_i \mathbf{w}^T\mathbf{x}_i + \log(1-\sigma(\mathbf{w}^T\mathbf{x}_i)))
\end{aligned}$$

For maximization, we require the gradient $\nabla_{\mathbf{w}}\mathcal{L}$:

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \sigma(\mathbf{w}^T\mathbf{x}_i))\mathbf{x}_i$$

Or

$$\frac{\partial}{\partial w_j}\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \sigma(\mathbf{w}^T\mathbf{x}_i))x_{i,j}$$

where $x_{i,j}$ is the j-th element of the i-th feature vector.

Logistic Regression:
Summary

- Logistic Regression is a *supervised* and *parametric* classifier
- Logistic Regression is a *linear* classifier
  Not to be confused with other !regression! methods.
- Posterior probabilities can be written in terms of a logistic function
- Training:
  - Estimate unknown parameters $\mathbf{w}$ from training data set
  - using Maximum-Likelihood Estimation (MLE)
  - Determine Log-Likelihood function
  - Compute gradient
  - Use numerical optimization methods (gradient descent, Newton's method, ...)
- Model formulation $\sigma(\mathbf{w}^T\mathbf{x})$ used as building block inside neural networks
- Implementation of Logistic Regression in scikit-learn

References

[Bea17] Andrew L. Beam. Deep learning 101 - part 1: History and background, 2017.

[Hay09] Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[M+97] Tom M Mitchell et al. *Machine learning*. McGraw-Hill Boston, MA, 1997.

