Inf2b - Learning

Lecture 11: Single layer Neural Networks (1)

Hiroshi Shimodaira
(Credit: Iain Murray and Steve Renals)

Centre for Speech Technology Research (CSTR)
School of Informatics
University of Edinburgh

http://www.inf.ed.ac.uk/teaching/courses/inf2b/
https://piazza.com/ed.ac.uk/spring2020/infr08028
Office hours: Wednesdays at 14:00-15:00 in IF-3.04

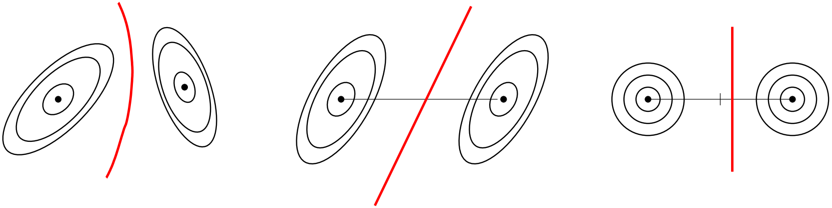
Jan-Mar 2020

Today's Schedule

1. Discriminant functions (recap)
2. Decision boundary of linear discriminants (recap)
3. Discriminative training of linear discriminans (Perceptron)
4. Structures and decision boundaries of Perceptron
5. LSE Training of linear discriminants
6. Appendix - calculus, gradient descent, linear regression

Discriminant functions (recap)

$$
\begin{aligned}
y_k(\mathbf{x}) &= \ln \left(P(\mathbf{x}|C) P(C_k) \right) \\
&= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) - \frac{1}{2} \ln |\boldsymbol{\Sigma}_k| + \ln P(C_k) \\
&= -\frac{1}{2}\mathbf{x}^T \boldsymbol{\Sigma}_k^{-1} \mathbf{x} + \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}_k^{-1} \mathbf{x} - \frac{1}{2}\boldsymbol{\mu}_k^T \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \ln |\boldsymbol{\Sigma}_k| + \ln P(C_k)
\end{aligned}
$$



(a) (b) (c)

Linear discriminants for a 2-class problem

$$y_1(\boldsymbol{x}) = \boldsymbol{w}_1^T \boldsymbol{x} + w_{10}$$
$$y_2(\boldsymbol{x}) = \boldsymbol{w}_2^T \boldsymbol{x} + w_{20}$$

Combined discriminant function:

$$\begin{aligned} y(\boldsymbol{x}) = y_1(\boldsymbol{x}) - y_2(\boldsymbol{x}) &= (\boldsymbol{w}_1 - \boldsymbol{w}_2)^T \boldsymbol{x} + (w_{10} - w_{20}) \\ &= \boldsymbol{w}^T \boldsymbol{x} + w_0 \end{aligned}$$

Decision:

$$C = \begin{cases} 1, & \text{if } y(\boldsymbol{x}) \geq 0, \\ 2, & \text{if } y(\boldsymbol{x}) < 0 \end{cases}$$

Decision boundary of linear discriminants

- Decision boundary:

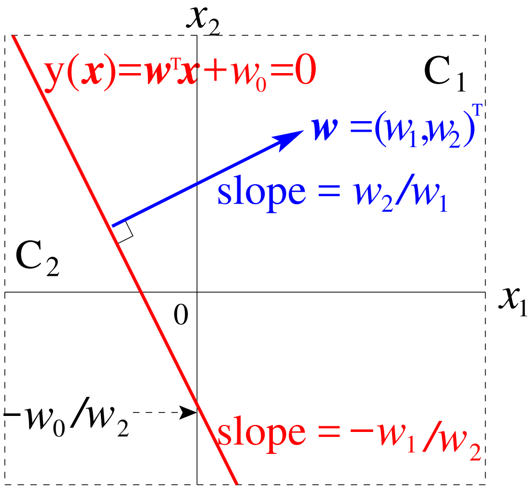
$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0$$

Dimension	Decision boundary	
2	line	$w_1 x_1 + w_2 x_2 + w_0 = 0$
3	plane	$w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 = 0$
D	hyperplane	$\left(\sum_{i=1}^{D} w_i x_i\right) + w_0 = 0$

NB: $\boldsymbol{w}$ is a normal vector to the hyperplane

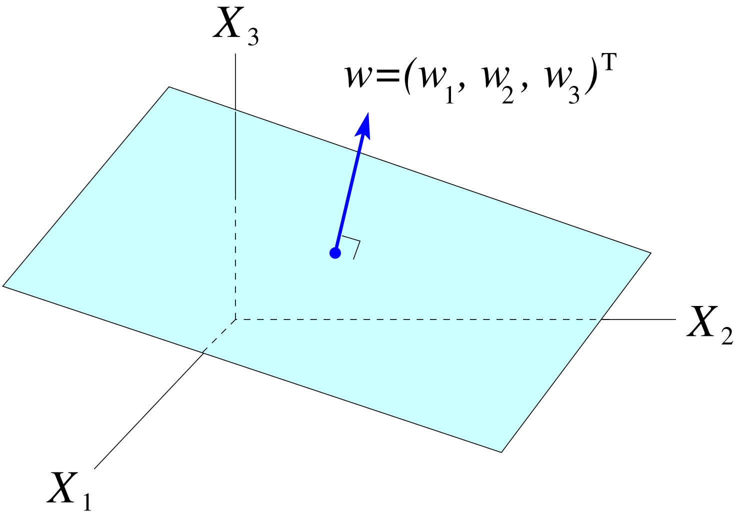
Decision boundary of linear discriminant (2D)

$$y(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + w_0 = 0 \quad \left(x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}, \quad \text{when } w_2 \neq 0\right)$$



Decision boundary of linear discriminant (3D)

$$y(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 = 0$$



Approach to linear discminant functions

Generative models : $p(\mathbf{x}|C_k)$

Discriminant function based on Bayes decision rule

$$y_k(\boldsymbol{x}) = \ln p(\mathbf{x}|C_k) + \ln P(C_k)$$

$\downarrow$ Gaussian pdf (model)

$$y_k(\boldsymbol{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) - \frac{1}{2}\ln|\boldsymbol{\Sigma}_k| + \ln P(C_k)$$

$\downarrow$ Equal covariance assumption

$$y_k(\boldsymbol{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

$\uparrow$ Why not estimating the decision boundary or $P(C_k|\mathbf{x})$ directly?

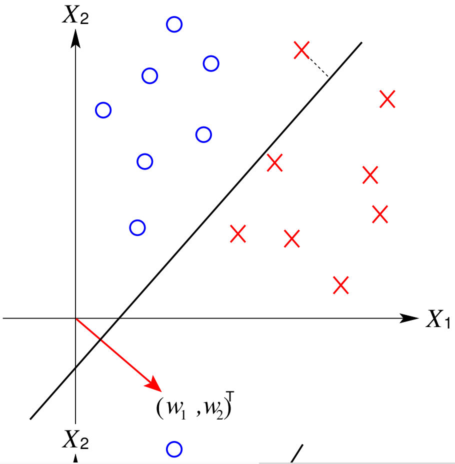
Discriminative training / models

(Logistic regression, Percepton / Neural network, SVM)

Training linear discriminant functions directly

- A discriminant for a two-class problem:

$$y(\boldsymbol{x}) = y_1(\boldsymbol{x}) - y_2(\boldsymbol{x}) = (\boldsymbol{w}_1 - \boldsymbol{w}_2)^T \boldsymbol{x} + (w_{10} - w_{20})$$
$$= \boldsymbol{w}^T \boldsymbol{x} + w_0$$



Perceptron error correction algorithm

$$a(\dot{\mathbf{x}}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 \;=\; \dot{\mathbf{w}}^T \dot{\mathbf{x}}$$

where $\dot{\mathbf{w}} = (w_0, \boldsymbol{w}^T)^T,\; \dot{\mathbf{x}} = (1, \boldsymbol{x}^T)^T$

Let's just use $\boldsymbol{w}$ and $\boldsymbol{x}$ to denote $\dot{\mathbf{w}}$ and $\dot{\mathbf{x}}$ from now on!

$$y(\boldsymbol{x}) = g(\,a(\boldsymbol{x})\,) \;=\; g(\,\boldsymbol{w}^T \boldsymbol{x}\,)$$

where $g(a) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases}$

$g(a)$: activation / transfer function

- Training set : $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$

  where $t_i \in \{0, 1\}$: target value

- Modify $\boldsymbol{w}$ if $\mathbf{x}_i$ was misclassified

$$\boldsymbol{w}^{(\mathrm{new})} \;\leftarrow\; \boldsymbol{w} + \eta\,(t_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

learning rate

NB:

$$(\boldsymbol{w}^{(\mathrm{new})})^T \mathbf{x}_i = \boldsymbol{w}^T \mathbf{x}_i + \eta\,(t_i - y(\mathbf{x}_i))\,\|\mathbf{x}_i\|^2$$

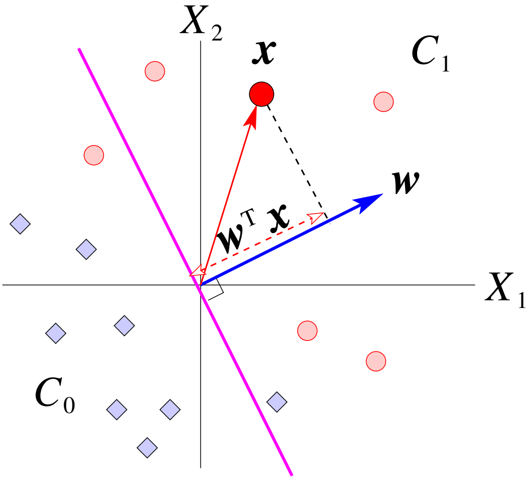
Geometry of Perceptron's error correction

$$y(\mathbf{x}_i) = g(\boldsymbol{w}^T \mathbf{x}_i)$$

$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \eta\,(t_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

$t_i - y(\boldsymbol{x}_i)$		$y(\mathbf{x}_i)$: 0	$y(\mathbf{x}_i)$: **1**
t_i	0	0	-1
	1	1	**0**

$\boldsymbol{w}^T \boldsymbol{x} = \|\boldsymbol{w}\| \|\boldsymbol{x}\| \cos\theta$



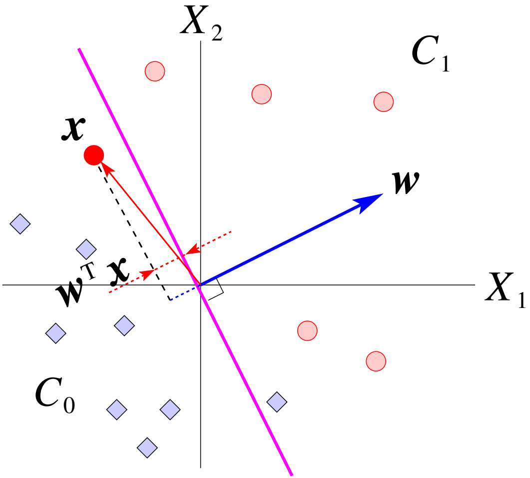
Geometry of Perceptron's error correction *(cont.)*

$$y(\mathbf{x}_i) = g(\boldsymbol{w}^T \mathbf{x}_i)$$

$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \eta \, (t_i - y(\mathbf{x}_i)) \, \mathbf{x}_i \qquad (0 < \eta < 1)$$

$t_i - y(\boldsymbol{x}_i)$		$y(\mathbf{x}_i)$ **0**	$y(\mathbf{x}_i)$ 1
t_i	0	0	-1
	1	**1**	0

$\boldsymbol{w}^T \boldsymbol{x} = \|\boldsymbol{w}\| \|\boldsymbol{x}\| \cos\theta$

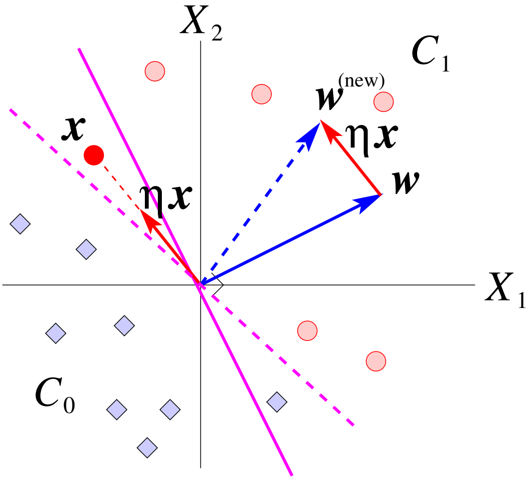


Geometry of Perceptron's error correction *(cont.)*

$$y(\mathbf{x}_i) = g(\boldsymbol{w}^T \mathbf{x}_i)$$
$$\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \eta\,(t_i - y(\mathbf{x}_i))\,\mathbf{x}_i \qquad (0 < \eta < 1)$$

$t_i - y(\boldsymbol{x}_i)$		$y(\mathbf{x}_i)$: 0	1
t_i	0	0	-1
	1	1	0

$\boldsymbol{w}^T \boldsymbol{x} = \|\boldsymbol{w}\| \|\boldsymbol{x}\| \cos\theta$



The Perceptron learning algorithm

Incremental (online) Perceptron algorithm:

$$
\begin{aligned}
&\text{for } i = 1, \ldots, N \\
&\quad \boldsymbol{w} \;\leftarrow\; \boldsymbol{w} + \eta \, (t_i - y(\mathbf{x}_i)) \, \mathbf{x}_i
\end{aligned}
$$

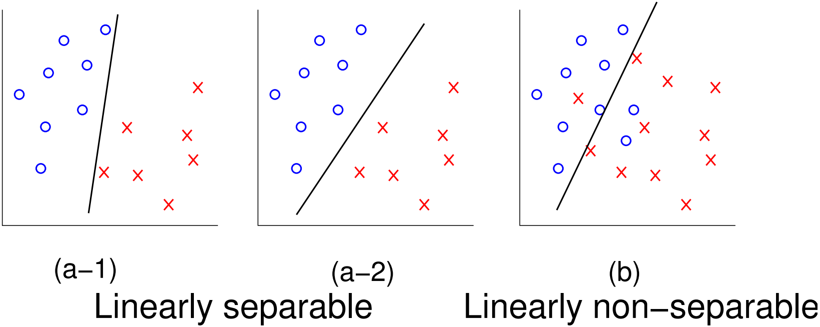
Batch Perceptron algorithm:

$$
\begin{aligned}
&\boldsymbol{v}_{sum} = \mathbf{0} \\
&\text{for } i = 1, \ldots, N \\
&\quad \boldsymbol{v}_{sum} = \boldsymbol{v}_{sum} + (t_i - y(\mathbf{x}_i)) \; \mathbf{x}_i \\
&\boldsymbol{w} \;\leftarrow\; \boldsymbol{w} + \eta \, \boldsymbol{v}_{sum}
\end{aligned}
$$

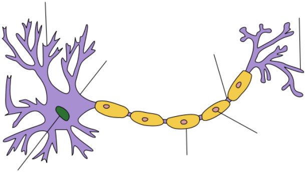
What about convergence?

The Perceptron learning algorithm terminates if training samples are linearly separable.

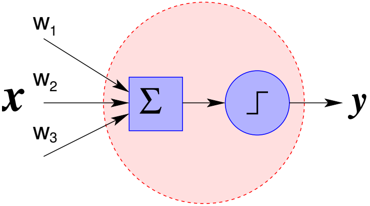
Linearly separable vs linearly non-separable



Background of Perceptron



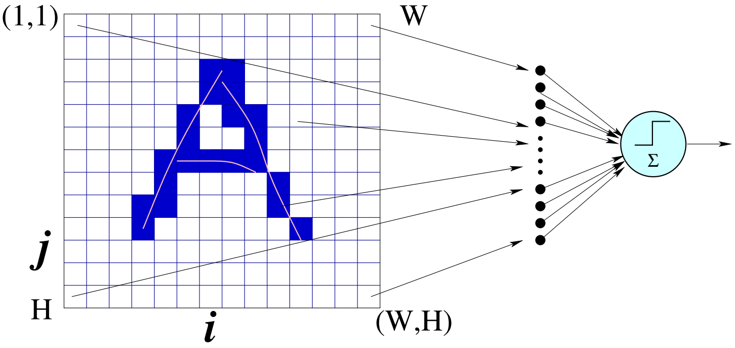
(https://en.wikipedia.org/wiki/File:Neuron_Hand-tuned.svg)



(a) function unit

1940s	Warren McCulloch and Walter Pitts : 'threshold logic'
	Donald Hebb : 'Hebbian learning'
1957	Frank Rosenblatt : 'Perceptron'

Character recognition by Perceptron

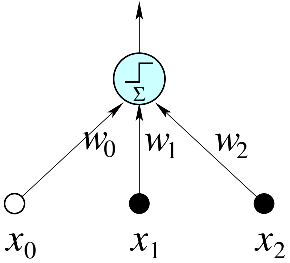


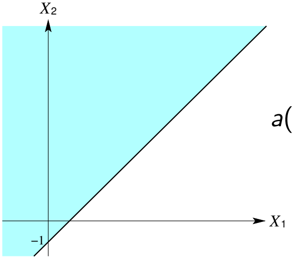
Perceptron structures and decision boundaries

$$y(\boldsymbol{x}) = g(a(\boldsymbol{x})) \qquad \boldsymbol{w} = (w_0, w_1, \ldots, w_D)^T$$

$$= g(\boldsymbol{w}^T\boldsymbol{x}) \qquad \boldsymbol{x} = (1, x_1, \ldots, x_D)^T$$

$$\text{where } g(a) = \begin{cases} 1, & \text{if } a \geq 0, \\ 0, & \text{if } a < 0 \end{cases}$$





$$x_2 \geq x_1 - 1$$

$$a(\boldsymbol{x}) = 1 - x_1 + x_2$$
$$= w_0 + w_1 x_1 + w_2 x_2$$

$$w_0 = 1,\ w_1 = -1,\ w_2 = 1$$

NB: A one node/neuron constructs a decision boundary, which splits the input space into two regions

Perceptron as a logical function

NOT

x_1	y
0	1
1	0

OR

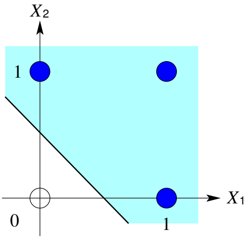
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

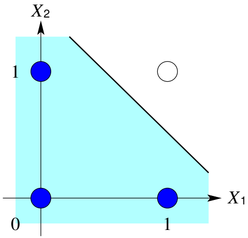
NAND

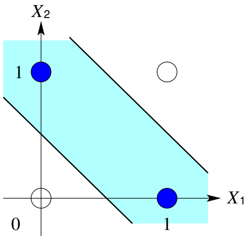
x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

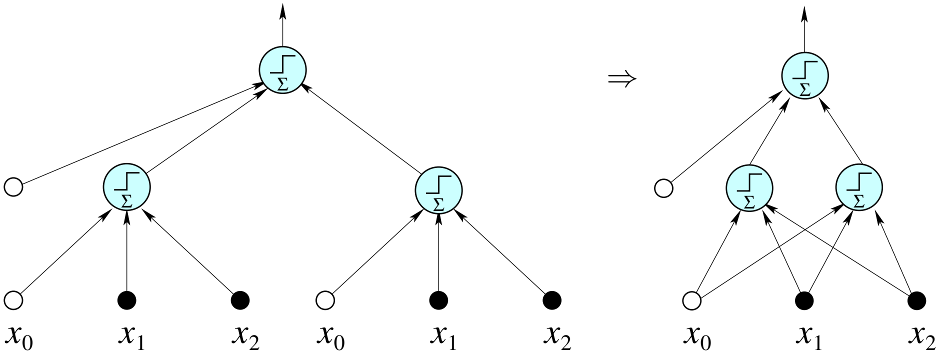




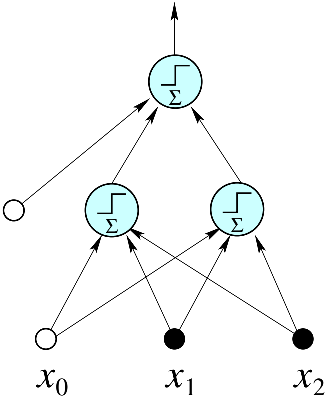


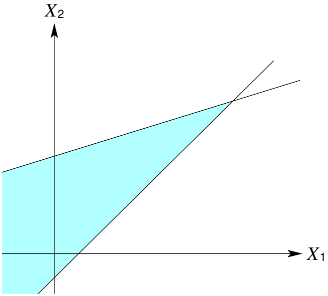
Question: find the weights for each function

Perceptron structures and decision boundaries *(cont.)*

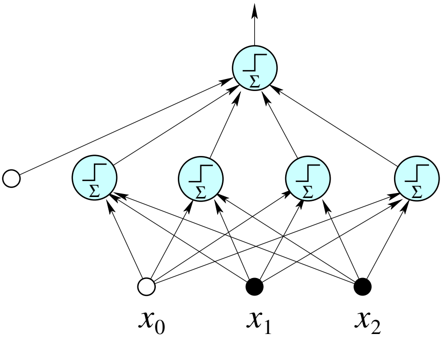


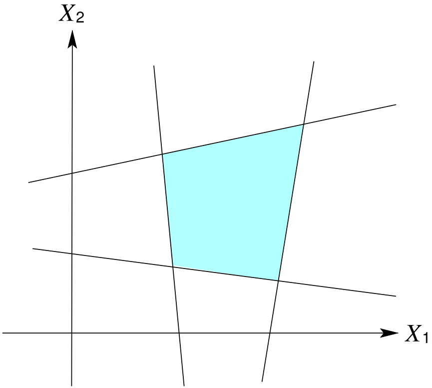
Perceptron structures and decision boundaries (*cont.*)



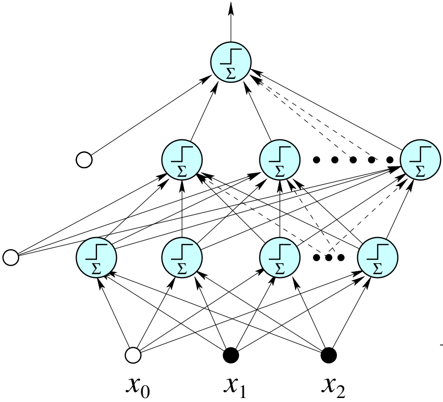


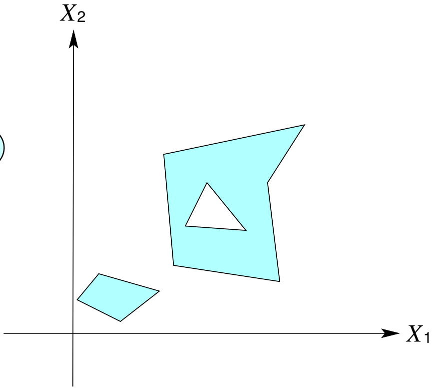
Perceptron structures and decision boundaries *(cont.)*





Perceptron structures and decision boundaries (*cont.*)





Problems with the Perceptron learning algorithm

- No training algorithms for multi-layer Percepton
- Non-convergence for linearly non-separable data
- Weights $\boldsymbol{w}$ are adjusted for misclassified data only (correctly classified data are not considered at all)

$\Rightarrow$

- Consider not only mis-classification (on train data), but also the optimality of decision boundary
    - Least squares error training
    - Large margin classifiers (e.g. SVM)

Training with least squares

- Squared error function:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{w}^T\mathbf{x}_n - t_n\right)^2$$

- Optimisation problem:

$$\min_{\boldsymbol{w}} E(\boldsymbol{w})$$

- One way to solve this is to apply gradient descent (steepest descent):

$$\boldsymbol{w} \;\leftarrow\; \boldsymbol{w} - \eta\,\nabla_{\boldsymbol{w}} E(\boldsymbol{w})$$

where η : step size (a small positive const.)

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \left(\frac{\partial E}{\partial w_0}, \ldots \frac{\partial E}{\partial w_D}\right)^T$$

Training with least squares *(cont.)*

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{n=1}^{N} \left(\boldsymbol{w}^T \mathbf{x}_n - t_n \right)^2 \\
&= \sum_{n=1}^{N} \left(\boldsymbol{w}^T \mathbf{x}_n - t_n \right) \frac{\partial}{\partial w_i} \boldsymbol{w}^T \mathbf{x}_n \\
&= \sum_{n=1}^{N} \left(\boldsymbol{w}^T \mathbf{x}_n - t_n \right) x_{ni}
\end{aligned}
$$

- Trainable in linearly non-separable case
- Not robust (sensitive) against errornous data (outliers) far away from the boundary
- More or less a linear discriminant

Appendix – derivatives

- Derivatives of functions of one variable

$$\frac{\mathrm{d}f}{\mathrm{d}x} = f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

e.g., $f(x) = 4x^3,\ f'(x) = 12x^2$

- Partial derivatives of functions of more than one variable

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x, y)}{\epsilon}$$

e.g., $f(x, y) = y^3 x^2,\ \frac{\partial f}{\partial x} = 2y^3 x$

Derivative rules

	Function	Derivative
Constant	c	0
	x	1
Power	x^n	nx^{n-1}
	$\frac{1}{x}$	$-\frac{1}{x^2}$
	$\sqrt{x}$	$\frac{1}{2}x^{-\frac{1}{2}}$
Exponential	e^x	e^x
Logarithms	$\ln(x)$	$\frac{1}{x}$
Sum rule	$f(x) + g(x)$	$f'(x) + g'(x)$
Product rule	$f(x)g(x)$	$f'(x)g(x) + f(x)g'(x)$
Reciprocal rule	$\frac{1}{f(x)}$	$-\frac{f'(x)}{f^2(x)}$
	$\frac{f(x)}{g(x)}$	$-\frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)}$
Chain rule	$f(g(x))$	$f'(g(x))g'(x)$
	$z = f(y), y = g(x)$	$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}$

Vectors of derivatives

Consider $f(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, \ldots, x_D)^T$

Notation: all partial derivatives put in a vector:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_D}\right)^T$$

Example: $f(\boldsymbol{x}) = x_1^3 x_2^2$

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{pmatrix} 3x_1^2 x_2^2 \\ 2x_1^3 x_2 \end{pmatrix}$$

Fact: $f(\boldsymbol{x})$ changes most quickly in direction $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$

Gradient descent (steepest descent)

- First order optimisation algorithm using $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
- Optimisation problem: $\min_{\boldsymbol{x}} f(\boldsymbol{x})$
- Useful when analytic solutions (closed forms) are not available or difficult to find
- Algorithm
  1. Set an initial value $\boldsymbol{x}_0$ and set $t = 0$
  2. If $\|\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_t)\| \simeq 0$, then stop. Otherwise, do the following.
  3. $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_t)$ for $\eta > 0$
  4. $t = t + 1$, and go to step 2.
- Problem: stops at a local minimum (difficult to find a global maximum).

Linear regression (one variable) least squares line fitting

- Training set: $\mathcal{D} = \{(x_n, t_n)\}_{n=1}^{N}$
- Linear regression: $\hat{t}_n = a x_n + b$
- Objective function: $E = \sum_{n=1}^{N} (t_i - (a x_i + b))^2$
- Optimisation problem: $\min_{a,b} E$
- Partial derivatives:

$$\frac{\partial E}{\partial a} = 2 \sum_{n=1}^{N} ((t_i - (a x_i + b)) (-x_i)$$

$$= 2a \sum_{n=1}^{N} x_i^2 + 2b \sum_{n=1}^{N} x_i - 2 \sum_{n=1}^{N} t_i x_i$$

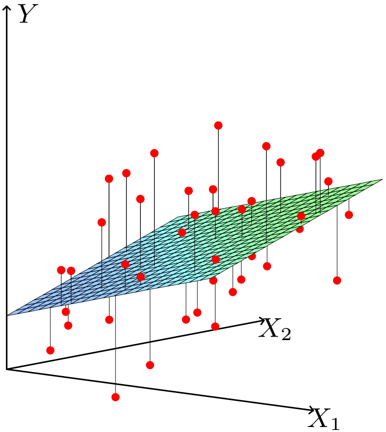
$$\frac{\partial E}{\partial b} = -2 \sum_{n=1}^{N} ((t_i - (a x_i + b))$$

$$= 2a \sum_{n=1}^{N} x_i + 2b \sum_{n=1}^{N} 1 - 2 \sum_{n=1}^{N} t_i$$

Letting $\dfrac{\partial E}{\partial a} = 0$ and $\dfrac{\partial E}{\partial b} = 0$

$$\begin{pmatrix} \sum_{n=1}^{N} x_i^2 & \sum_{n=1}^{N} x_i \\ \sum_{n=1}^{N} x_i & \sum_{n=1}^{N} 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_{n=1}^{N} t_i x_i \\ \sum_{n=1}^{N} t_i \end{pmatrix}$$

Linear regression (multiple variables)



- Training set:
  $\mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^{N}$, where
  $\mathbf{x}_n = (1, x_1, \ldots, x_D)^T$
- Linear regression:
  $\hat{t}_n = \boldsymbol{w}^T \mathbf{x}_n$
- Objective function:
  $$E = \sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^T \mathbf{x}_n\right)^2$$
- Optimisation problem:
  $\min_{a,b} E$

Elements of Statistical Learning (2nd Ed.) © Hastie, Tibshirani & Friedman 2009

Linear regression (multiple variables) *(cont.)*

- $E = \sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^T \mathbf{x}_n \right)^2$
- Partial derivatives: $\dfrac{\partial E}{\partial w_i} = -2 \sum_{n=1}^{N} (t_n - \boldsymbol{w}^T \mathbf{x}_n) x_{ni}$
- Vector/matrix representation *(NE)* :

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{10}, & \ldots, & x_{1d} \\ \vdots & & \vdots \\ x_{N0}, & \ldots, & x_{Nd} \end{bmatrix}, T = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}$$

$$E = (T - X\boldsymbol{w})^T (T - X\boldsymbol{w})$$

$$\frac{\partial E}{\partial \boldsymbol{w}} = -2X^T (T - XW)$$

Letting $\frac{\partial E}{\partial \boldsymbol{w}} = \mathbf{0} \quad \Rightarrow X^T (T - XW) = \mathbf{0}$

$$X^T X W = X^T T$$

$$W = (X^T X)^{-1} X^T T$$ $\cdots$ analytic solution if the inverse exists

Summary

- Training discriminant functions directly (discriminative training)
- Perceptron training algorithm
  - Perceptron error correction algorithm
  - Least squares error + gradient descent algorithm
- Linearly separable vs linearly non-separable
- Perceptron structures and decision boundaries
- See Notes 11 for a Perceptron with multiple output nodes