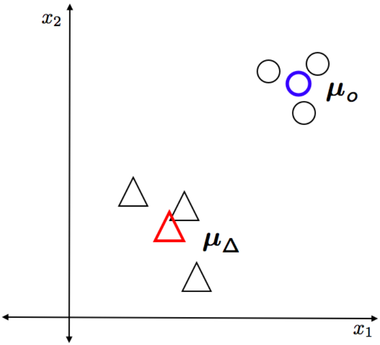
Machine Learning

Lecture 7
Perceptrons

Felix Bießmann

Beuth University & Einstein Center for Digital Future

Recap: Nearest Centroid Classification



Prototypes $\boldsymbol{\mu}_\Delta$ and $\boldsymbol{\mu}_o$ can be the class means

$$\boldsymbol{\mu}_\Delta = 1/N_\Delta \sum_n^{N_\Delta} \mathbf{x}_{\Delta,n}$$

$$\boldsymbol{\mu}_o = 1/N_o \sum_n^{N_o} \mathbf{x}_{o,n}$$

Distance from w_Δ to new data x

$$\|\boldsymbol{\mu}_\Delta - \mathbf{x}\|_2$$

From Prototypes to Linear Classification

$$\begin{aligned} \text{distance}(\mathbf{x}, \boldsymbol{\mu}_\Delta) &> \text{distance}(\mathbf{x}, \boldsymbol{\mu}_o) \\ \|\mathbf{x} - \boldsymbol{\mu}_\Delta\| &> \|\mathbf{x} - \boldsymbol{\mu}_o\| \end{aligned} \tag{1}$$

From Prototypes to Linear Classification

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \boldsymbol{\mu}_\Delta) &> \text{distance}(\mathbf{x}, \boldsymbol{\mu}_o) \qquad (1)\\
\|\mathbf{x} - \boldsymbol{\mu}_\Delta\| &> \|\mathbf{x} - \boldsymbol{\mu}_o\| \\
\Leftrightarrow \|\mathbf{x} - \boldsymbol{\mu}_\Delta\|^2 &> \|\mathbf{x} - \boldsymbol{\mu}_o\|^2 \\
\Leftrightarrow \mathbf{x}^\top \mathbf{x} - 2\boldsymbol{\mu}_\Delta^\top \mathbf{x} + \boldsymbol{\mu}_\Delta^\top \boldsymbol{\mu}_\Delta &> \mathbf{x}^\top \mathbf{x} - 2\boldsymbol{\mu}_o^\top \mathbf{x} + \boldsymbol{\mu}_o^\top \boldsymbol{\mu}_o \\
\Leftrightarrow \boldsymbol{\mu}_\Delta^\top \mathbf{x} - \boldsymbol{\mu}_\Delta^2/2 &< \boldsymbol{\mu}_o^\top \mathbf{x} - \boldsymbol{\mu}_o^2/2 \\
\Leftrightarrow 0 &< \underbrace{(\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta)}_{\mathbf{w}}^\top \mathbf{x} - 1/2 \underbrace{(\boldsymbol{\mu}_o^\top \boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta^\top \boldsymbol{\mu}_\Delta)}_{\beta}
\end{aligned}
$$

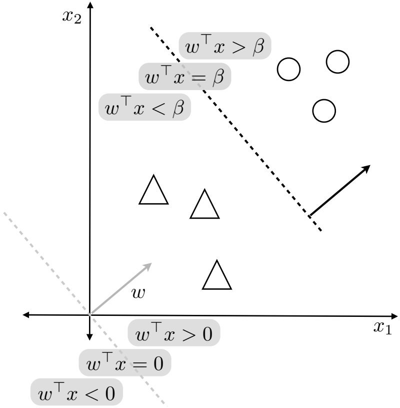
From Prototypes to Linear Classification

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \boldsymbol{\mu}_\Delta) &> \text{distance}(\mathbf{x}, \boldsymbol{\mu}_o) \\
\|\mathbf{x} - \boldsymbol{\mu}_\Delta\| &> \|\mathbf{x} - \boldsymbol{\mu}_o\| \\
\Leftrightarrow \|\mathbf{x} - \boldsymbol{\mu}_\Delta\|^2 &> \|\mathbf{x} - \boldsymbol{\mu}_o\|^2 \\
\Leftrightarrow \mathbf{x}^\top \mathbf{x} - 2\boldsymbol{\mu}_\Delta^\top \mathbf{x} + \boldsymbol{\mu}_\Delta^\top \boldsymbol{\mu}_\Delta &> \mathbf{x}^\top \mathbf{x} - 2\boldsymbol{\mu}_o^\top \mathbf{x} + \boldsymbol{\mu}_o^\top \boldsymbol{\mu}_o \\
\Leftrightarrow \boldsymbol{\mu}_\Delta^\top \mathbf{x} - \boldsymbol{\mu}_\Delta^2/2 &< \boldsymbol{\mu}_o^\top \mathbf{x} - \boldsymbol{\mu}_o^2/2 \\
\Leftrightarrow 0 &< \underbrace{(\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta)}_{\mathbf{w}}{}^\top \mathbf{x} - 1/2 \underbrace{(\boldsymbol{\mu}_o^\top \boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta^\top \boldsymbol{\mu}_\Delta)}_{\beta}
\end{aligned}
\tag{1}
$$

Linear Classification

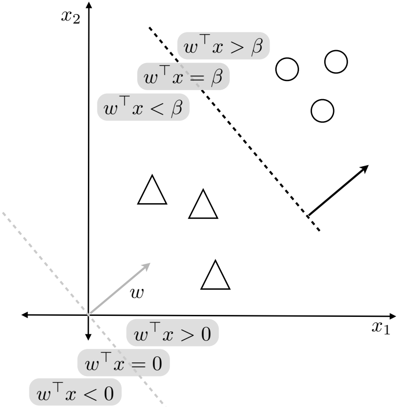
$$
\mathbf{w}^\top \mathbf{x} - \beta = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ belongs to class } \circ \\ < 0 & \text{if } \mathbf{x} \text{ belongs to class } \Delta \end{cases}
\tag{2}
$$

Linear Classification



$$\mathbf{w}^\top \mathbf{x} - \beta = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ belongs to } o \\ < 0 & \text{if } \mathbf{x} \text{ belongs to } \Delta \end{cases}$$

Linear Classification



What is a good **w**?

→ We need an **error function** that tells us how good **w** is.

Two classical Error Functions

Given data $\mathbf{x} \in \mathbb{R}^D$ and corresponding labels $y \in \{-1, +1\}$, two classical error functions $\mathcal{E}(\mathbf{x}, y, \mathbf{w})$ to find the optimal $\mathbf{w} \in \mathbb{R}^D$ are:

Error Function	Used in
$\frac{1}{2}(y - \mathbf{w}^\top \mathbf{x})^2$	Adaline [**?**]
$\max(0, -y\mathbf{w}^\top \mathbf{x})$	Perceptron [**?**]

Rosenblatt's Perceptron

Frank Rosenblatt
(1928-1969)

Rosenblatt proposed the **perceptron**, an artificial neural network for pattern recognition [**?**]

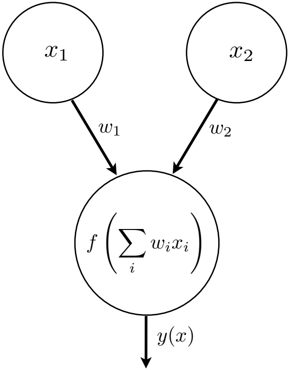
Perceptrons gave rise to the field of artificial neural networks

Rosenblatt's Perceptron

Rosenblatt studied Perceptrons, so that

"[...] the fundamental laws of organization which are common to all information handling systems, machines and men included, may eventually be understood."

Artificial Neural Networks



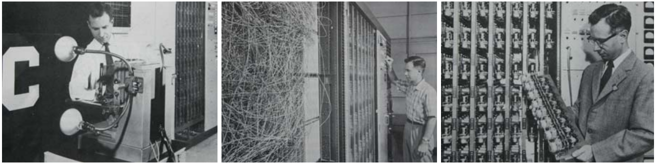
Input nodes x_i receive information

Inputs are multiplied with a weighting factor w_i and summed up

Integrated input is mapped through some (non-linear) function $f(.)$

$$f(x) = \begin{cases} +1 & \text{if } \mathbf{x} \text{ is preferred stimulus} \\ -1 & \text{if } \mathbf{x} \text{ is any other stimulus} \end{cases}$$

Rosenblatt's Perceptron



Perceptron Hardware (pictures from [?])

The Perceptron Learning Algorithm

Goal Binary classification of multivariate data $\mathbf{x} \in \mathbb{R}^D$

The Perceptron Learning Algorithm

Goal Binary classification of multivariate data $\mathbf{x} \in \mathbb{R}^D$

Input Learning rate η and N tupels $(\mathbf{x}_n, y_n)$ where
$\mathbf{x}_n \in \mathbb{R}^D$ is the D-dimensional data
$y_n \in \{-1, +1\}$ is the corresponding label, such that

$$y_n = \begin{cases} +1 & \text{if } \mathbf{x}_n \text{ is a correct example} \\ -1 & \text{if } \mathbf{x}_n \text{ is } \textbf{not} \text{ a correct example} \end{cases}$$

The Perceptron Learning Algorithm

Goal Binary classification of multivariate data $\mathbf{x} \in \mathbb{R}^D$

Input Learning rate η and N tupels $(\mathbf{x}_n, y_n)$ where

$\mathbf{x}_n \in \mathbb{R}^D$ is the D-dimensional data

$y_n \in \{-1, +1\}$ is the corresponding label, such that

$$y_n = \begin{cases} +1 & \text{if } \mathbf{x}_n \text{ is a correct example} \\ -1 & \text{if } \mathbf{x}_n \text{ is } \textbf{not} \text{ a correct example} \end{cases}$$

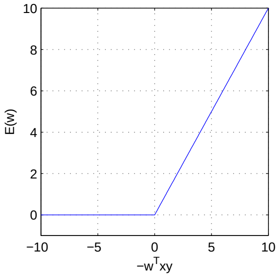
Output Weight vector $w \in \mathbb{R}^D$ such that

$$\mathbf{w}^\top \mathbf{x}_n = \begin{cases} \geq 0 & \text{if } y_n = +1 \\ < 0 & \text{if } y_n = -1 \end{cases}$$

The Perceptron Error Function

$$\mathcal{E}_{\mathcal{P}}(\mathbf{w}) = -\sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m \tag{3}$$

Classification Error as Function of Weights

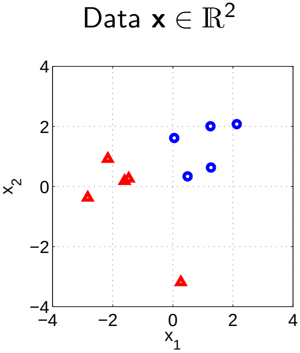


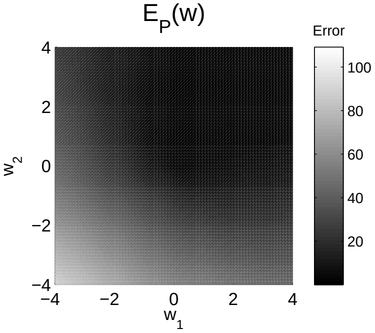
Given data $\mathbf{x} \in \mathbb{R}^D$ and corresponding labels $y \in \{-1, +1\}$ the classification error $\mathcal{E}$ is a function of the weights $\mathbf{w}$ (and the data $\mathbf{x}$, y)

$$\mathcal{E}(\mathbf{w}, \mathbf{x}_m, y_m) = -\sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m \quad (4)$$

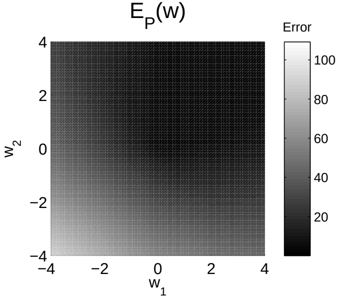
where $\mathcal{M}$ denotes the index set of all *misclassified* data $\mathbf{x}_m$

Classification Error as Function of Weights





Gradient Descent

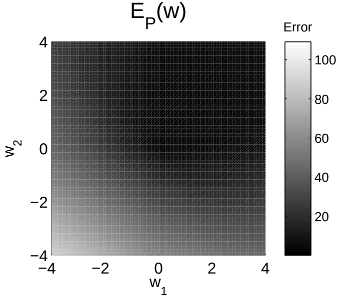


How to minimize the error function?

$$\mathcal{E}(\mathbf{w}, \mathbf{x}_m, y_m) = -\sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m$$

$\rightarrow$ **Gradient Descent**

Gradient Descent

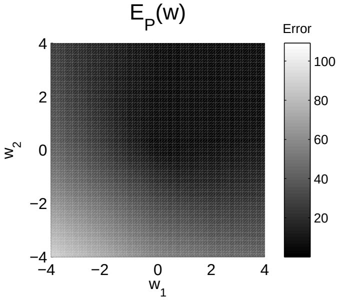


We minimize $\mathcal{E}(\mathbf{w}, \mathbf{x}_m, y_m)$ by walking in the opposite direction of the gradient.

$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta \frac{1}{|\mathcal{X}|} \sum_{i=1}^{|\mathcal{X}|} \nabla \mathcal{E}(\mathbf{w}, \mathbf{x}_i, y_i)$$

where $\mathcal{X}$ is the set of data points and η is called a **learning rate**.

Stochastic Gradient Descent



A noisy estimate of

$$\frac{1}{|\mathcal{X}|}\sum_{i=1}^{|\mathcal{X}|} \nabla\mathcal{E}(\mathbf{w}, \mathbf{x}, y)$$

is obtained by [?]

$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta\nabla\mathcal{E}(\mathbf{w}, \mathbf{x}_i, y_i)$$

Note that only $\mathbf{w}$ is stored and only one data point $\mathbf{x}_i$ and label y_i are considered at a time!

$\rightarrow$ Scales to large data sets [?]

Perceptron Training

Training a Perceptron means finding weights $\mathbf{w}$ that minimize $\mathcal{E}_{\mathcal{P}}$:

$$\underset{w}{\operatorname{argmin}} \quad \left(- \sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m \right) \tag{5}$$

where $\mathcal{M}$ denotes the index set of all *misclassified* data $\mathbf{x}_m$

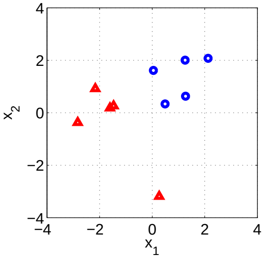
Perceptron Training

Training a Perceptron means finding weights $\mathbf{w}$ that minimize $\mathcal{E}_{\mathcal{P}}$:

$$\underset{w}{\operatorname{argmin}} \quad \left(- \sum_{m \in \mathcal{M}} \mathbf{w}^{\top} \mathbf{x}_m y_m \right) \tag{5}$$

where $\mathcal{M}$ denotes the index set of all *misclassified* data $\mathbf{x}_m$

Data $\mathbf{x} \in \mathbb{R}^2$



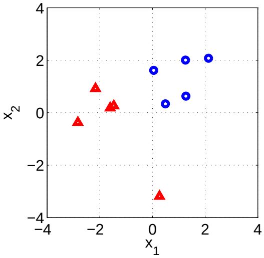
Perceptron Training

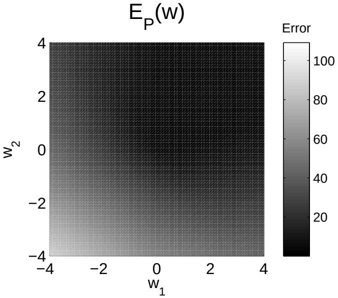
Training a Perceptron means finding weights $\mathbf{w}$ that minimize $\mathcal{E}_{\mathcal{P}}$:

$$\underset{w}{\operatorname{argmin}} \quad \left(- \sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m \right) \tag{5}$$

where $\mathcal{M}$ denotes the index set of all *misclassified* data $\mathbf{x}_m$

Data $\mathbf{x} \in \mathbb{R}^2$





The Perceptron Learning Algorithm

Eq. 5 can be minimized *iterative*ly
using **stochastic gradient descent**
[**??**]

1. Initialize $\mathbf{w}^{\text{old}}$ (randomly, $1/n$, ...)
2. While there are misclassified data points $\mathbf{x}_m$

   Pick a random misclassified data point $\mathbf{x}_m$

$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta \nabla \mathcal{E}_{\mathcal{P}}(\mathbf{w}) = \mathbf{w}^{\text{old}} + \eta \mathbf{x}_m y_m \tag{6}$$

The Perceptron Learning Algorithm

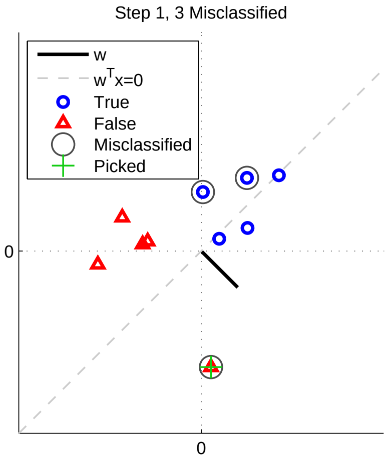
Algorithm 1 Perceptron learning

Require: Data $X = [\mathbf{x}_1, \ldots, \mathbf{x}_N], x_i \in \mathbb{R}^D$, Labels $y_1, \ldots, y_N \in \{-1, +1\}$, iterations *its*, learning rate η

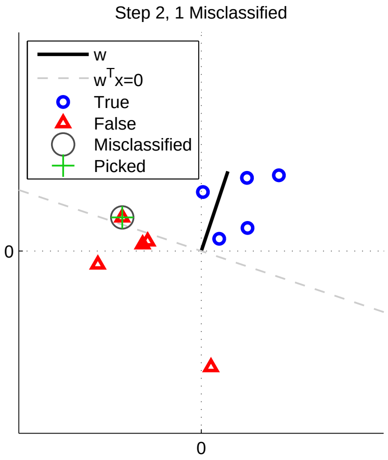
Ensure: $\mathbf{w}$

$\mathbf{w} = \mathbb{1}_D/(D)$
for $it = 1, \ldots, its$ **do**
    # Pick a random example
    $i =$ ceil(rand*N)
    # If this example is not correctly classified
    if $sign(\mathbf{w}^\top \mathbf{x}_i)\ \ != y_i$ **then**
        # Update weight vector
        $\mathbf{w} \leftarrow \mathbf{w} + \eta/it\ \mathbf{x}_i y_i$
    end if
end for

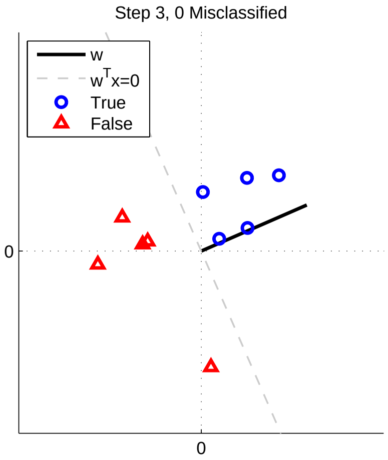
The Perceptron Learning Algorithm in Action



The Perceptron Learning Algorithm in Action



The Perceptron Learning Algorithm in Action



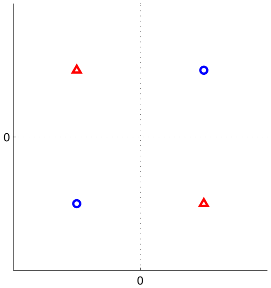
Problems with Perceptrons

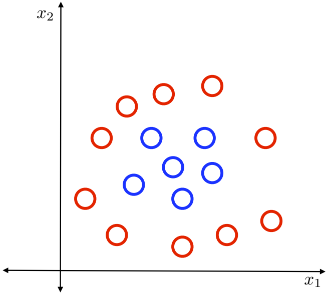
- Each update might lead to new misclassifications
- Many solutions might exist; which one is correct?
- Convergence might be slow
- Only solves linearly separable problems
- If there is no solution, the algorithm will not converge

Some solutions have been proposed by ?

Problems with Perceptrons

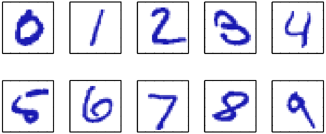
Perceptrons can only learn linearly separable problems.





Application example: Handwritten Digit Recognition

Handwritten digits
from UPS data set



Each digit represented as 16×16 pixel image

$\rightarrow$ $x \in \mathbb{R}^{256}$ input nodes

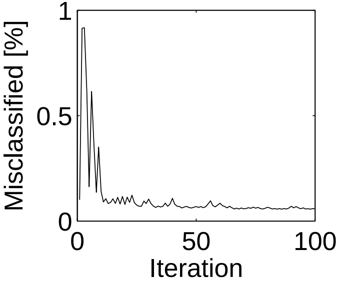
Each image is associated with a label $y \in \{0, 1, \ldots, 9\}$

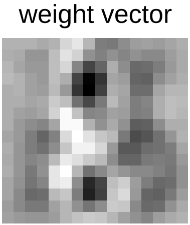
Goal Artificial neural network that recognizes the digit 8

$\Rightarrow$ We need a function $f(.)$ such that

$$f(x) = \begin{cases} -1 & \text{if } y \in \{0, 1, \ldots, 7, 9\} \\ +1 & \text{if } y = 8 \end{cases}$$

Application example: Handwritten Digit Recognition





Application example: Handwritten Digit Recognition

- Handwritten Digit Recognition is a **Multiclass Problem**
- But the Perceptron is a **two-class** (binary) classifier
- How can we **binarize** the multi class problem?

→ Train a perceptron for each digit against all others

→ Concatenate all weight vectors $W = [\mathbf{w_0}, \mathbf{w}_1, \ldots, \mathbf{w}_9]$

→ For new data point $\mathbf{x}$ compute the label as $\operatorname{argmax}\left(W^\top \mathbf{x}\right)$

Minimizing Functions

Remember:

- ML algorithms are functions f that need to be fitted to data
- ML is finding minima/maxima of functions
- Function minimization is done by **Gradient Descent**

Taking Derivatives of Univariate Functions

Function	Derivative
$f(x) = x^n$	
$cf(x)$	
$f(x) + g(x)$	
$f(x)g(x)$	
$\frac{f(x)}{g(x)}$	
$f(g(x))$	

Taking Derivatives of Univariate Functions

Function	Derivative
$f(x) = x^n$	$f'(x) = nx^{n-1}$
$cf(x)$	$cf'(x)$
$f(x) + g(x)$	$f'(x) + g'(x)$
$f(x)g(x)$	$f'(x)g(x) + f(x)g'(x)$
$\frac{f(x)}{g(x)}$	$\frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$
$f(g(x))$	$f'(g(x))\, g'(x)$

Taking Derivatives of Vector-Valued Functions

Consider a vector-valued function $f(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^U, \mathbf{x} \in \mathbb{R}^D$

The gradient $\nabla f(\mathbf{x})$ at position $\mathbf{x}$ is

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_D}\right] \tag{7}$$

Taking Derivatives of Vector-Valued Functions

Function	Derivative
$f(\mathbf{x}) = \mathbf{u}^\top \mathbf{x}$	
$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{x}$	
$f(\mathbf{x}) = A\mathbf{x}$	
$f(\mathbf{x}) = \mathbf{x}^\top A\mathbf{x}$	

More complicated examples: The Matrix Cookbook

Taking Derivatives of Vector-Valued Functions

Function	Derivative
$f(\mathbf{x}) = \mathbf{u}^\top \mathbf{x}$	$f'(\mathbf{x}) = \mathbf{u}$
$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{x}$	$f'(\mathbf{x}) = 2\mathbf{x}$
$f(\mathbf{x}) = A\mathbf{x}$	$f'(\mathbf{x}) = A$
$f(\mathbf{x}) = \mathbf{x}^\top A\mathbf{x}$	$f'(\mathbf{x}) = \mathbf{x}^\top (A + A^\top)$

More complicated examples: The Matrix Cookbook

References

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer US, 2007.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, 2010. Springer.

Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, pages 277–296, 1998.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400—407, 1951.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, Nov. 1958.

B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 1960. IRE.