ARISTOTLE UNIVERSITY OF THESSALONIKI

FACULTY OF ENGINEERING

Pattern Recognition & Machine Learning

Linear Discriminant Functions and Models

Panagiotis C. Petrantonakis

Assistant Professor

Dept. of Electrical and Computer Engineering

ppetrant@ece.auth.gr

Fall Semester

Until now…

- We have seen only one out of three different approaches of solving decision problems:
  - *Generative probabilistic models*:
    - solve the inference problem of determining the class-conditional density function $p(\boldsymbol{x}|\omega_i)$ and the priors $P(\omega_i)$ and then use the Bayes formula to find posterior class probabilities.

$$P(\omega_i\ |\boldsymbol{x}) = \frac{p(x|\omega_i)P(\omega_i)}{p(x)}$$

    - We used training data to infer $p(\boldsymbol{x}|\omega_i)$.

Until now…

- We have seen only one out of three different approaches of solving decision problems:
  - *Generative probabilistic models*:
    - solve the inference problem of determining the class-conditional density function $p(\boldsymbol{x}|\omega_i)$ and the priors $P(\omega_i)$ and then use the Bayes formula to find posterior class probabilities.

$$P(\omega_i\,|\boldsymbol{x}) = \frac{p(x|\omega_i)P(\omega_i)}{p(x)}$$

    - We used training data to infer $p(\boldsymbol{x}|\omega_i)$.
- We have also talked about *discriminant functions*
  - In the above case, we replaced $P(\omega_i\,|\boldsymbol{x})$ with $y_i(\boldsymbol{x})$ (in the BDT lecture we use the symbol g instead of y) and under certain assumptions about the class-conditional density $p(\boldsymbol{x}|\omega_i)$ we ended up with linear or non-linear *decision boundaries.*
  - We say that the corresponding discriminant functions are *linear* or *non-linear discriminants*!

Until now…

- We have seen only one out of three different approaches of solving decision problems:
  - *Generative probabilistic models*:
    - solve the inference problem of determining the class-conditional density function $p(\boldsymbol{x}|\omega_i)$ and the priors $P(\omega_i)$ and then use the Bayes formula to find posterior class probabilities.

$$P(\omega_i\,|\boldsymbol{x}) = \frac{p(x|\omega_i)P(\omega_i)}{p(x)}$$

    - We used training data to infer $p(\boldsymbol{x}|\omega_i)$.
- We have also talked about *discriminant functions*
  - In the above case, we replaced $P(\omega_i\,|\boldsymbol{x})$ with $y_i(\boldsymbol{x})$ (in the BDT lecture we use the symbol g instead of y) and under certain assumptions about the class-conditional density $p(\boldsymbol{x}|\omega_i)$ we ended up with linear or non-linear *decision boundaries.*
  - We say that the corresponding discriminant functions are *linear* or *non-linear discriminants*!

- In this lecture we will talk solely about linear discriminants of the form

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

Today…

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

- We will see the rest two approaches of solving decision problems
  - *Linear discriminant functions* that map each input $\boldsymbol{x}$ directly onto a class label.
    - For instance, in two-class classification problems $y(\boldsymbol{x})$ takes binary values such that $y(\boldsymbol{x}) = 0$ represents class ω_1 and $y(\boldsymbol{x}) = 1$ represents class ω_2 (probabilistic modeling plays no role in this scheme)

Today…

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

- We will see the rest two approaches of solving decision problems
  - *Linear discriminant functions* that map each input $\boldsymbol{x}$ directly onto a class label.
    - For instance, in two-class classification problems $y(\boldsymbol{x})$ takes binary values such that $y(\boldsymbol{x}) = 0$ represents class ω_1 and $y(\boldsymbol{x}) = 1$ represents class ω_2 (probabilistic modeling plays no role in this scheme)
  - *Discriminative probabilistic models*
    - solve the inference problem of determining the posterior class probabilities $P(\omega_i|\boldsymbol{x})$ and then assign each sample $\boldsymbol{x}$ to one of the classes

Today…

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

- We will see the rest two approaches of solving decision problems
  - *Linear discriminant functions* that map each input $\boldsymbol{x}$ directly onto a class label.
    - For instance, in two-class classification problems $y(\boldsymbol{x})$ takes binary values such that $y(\boldsymbol{x}) = 0$ represents class ω_1 and $y(\boldsymbol{x}) = 1$ represents class ω_2 (probabilistic modeling plays no role in this scheme)
  - *Discriminative probabilistic models*
    - solve the inference problem of determining the posterior class probabilities $P(\omega_i|\boldsymbol{x})$ and then assign each sample $\boldsymbol{x}$ to one of the classes
- For the former approach we will see the algorithms:
  - Least squares
  - Perceptron
- For the latter approach we will see the algorithm:
  - Logistic Regression

Today…

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

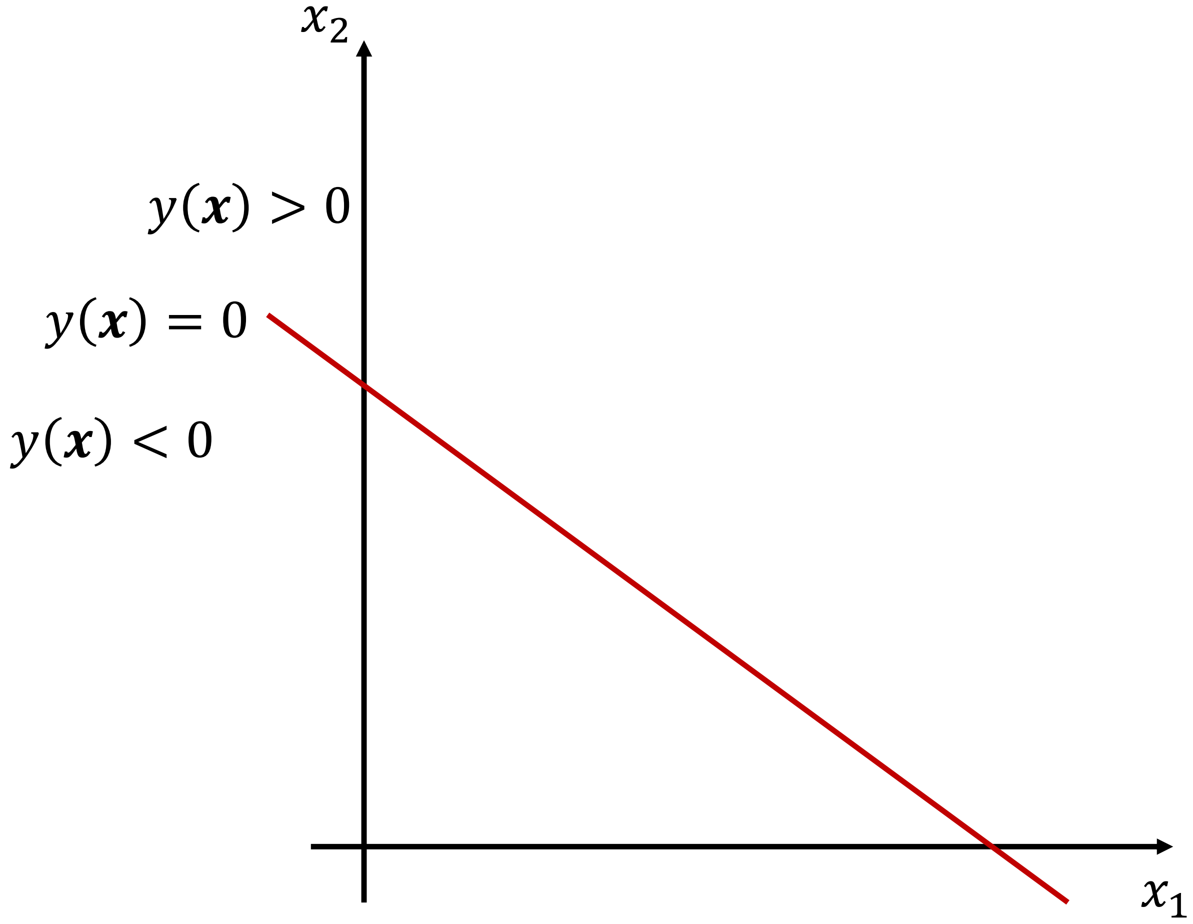
- We will see the rest two approaches of solving decision problems
  - *Linear discriminant functions* that map each input $\boldsymbol{x}$ directly onto a class label.
    - For instance, in two-class classification problems $y(\boldsymbol{x})$ takes binary values such that $y(\boldsymbol{x}) = 0$ represents class ω_1 and $y(\boldsymbol{x}) = 1$ represents class ω_2 (probabilistic modeling plays no role in this scheme)
  - *Discriminative probabilistic models*
    - solve the inference problem of determining the posterior class probabilities $P(\omega_i|\boldsymbol{x})$ and then assign each sample $\boldsymbol{x}$ to one of the classes
- For the former approach we will see the algorithms:
  - Least squares
  - Perceptron
- For the latter approach we will see the algorithm:
  - Logistic Regression

- In all algorithms that we will see in this lecture, we use training data to determine vector $\boldsymbol{w}$ and w_0. (big difference with the probabilistic generative models!)

Linear planes – Two classes

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

- We assign every input vector to ω_1 if $y(\boldsymbol{x}) \geq 0$ and to ω_2 otherwise.

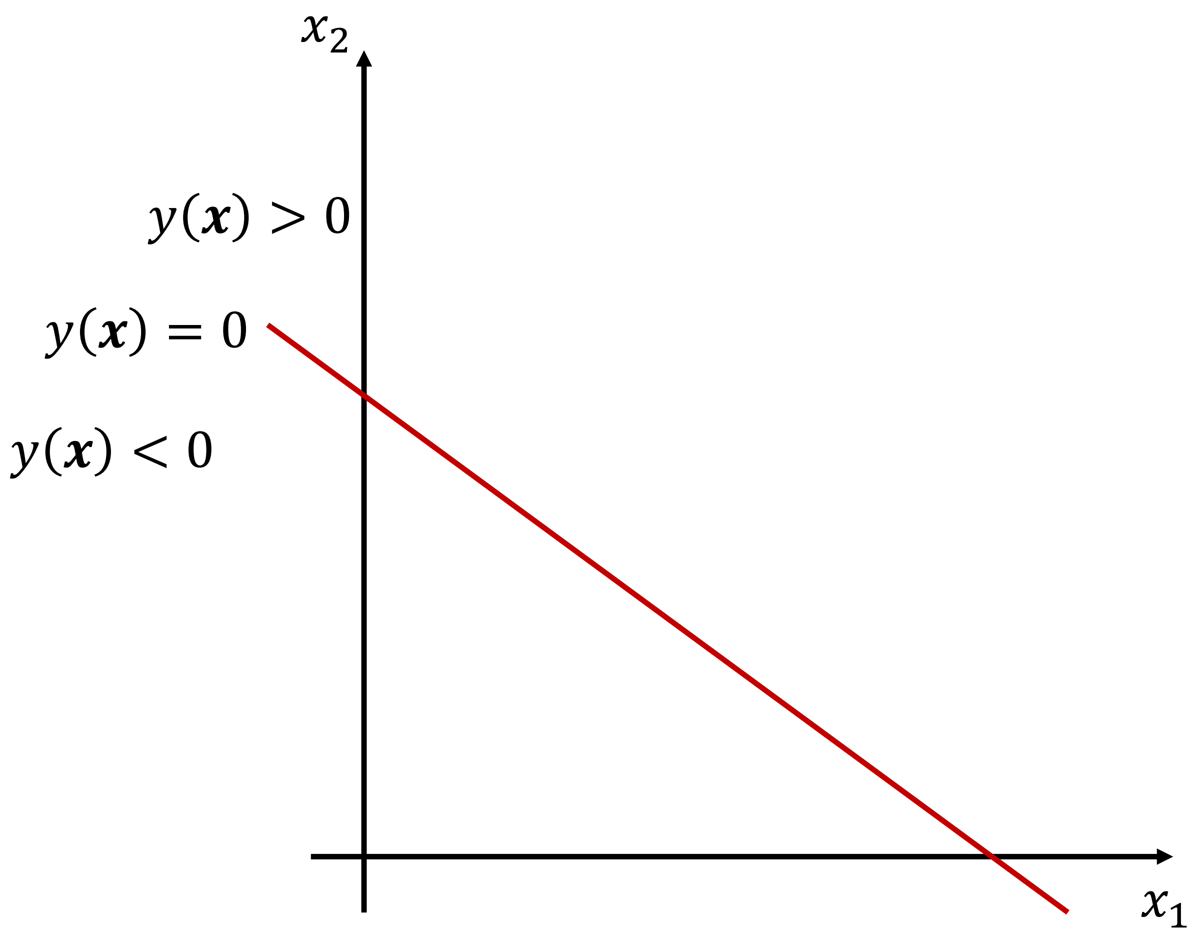


Linear planes – Two classes

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

- We assign every input vector to ω_1 if $y(\boldsymbol{x}) \geq 0$ and to ω_2 otherwise.

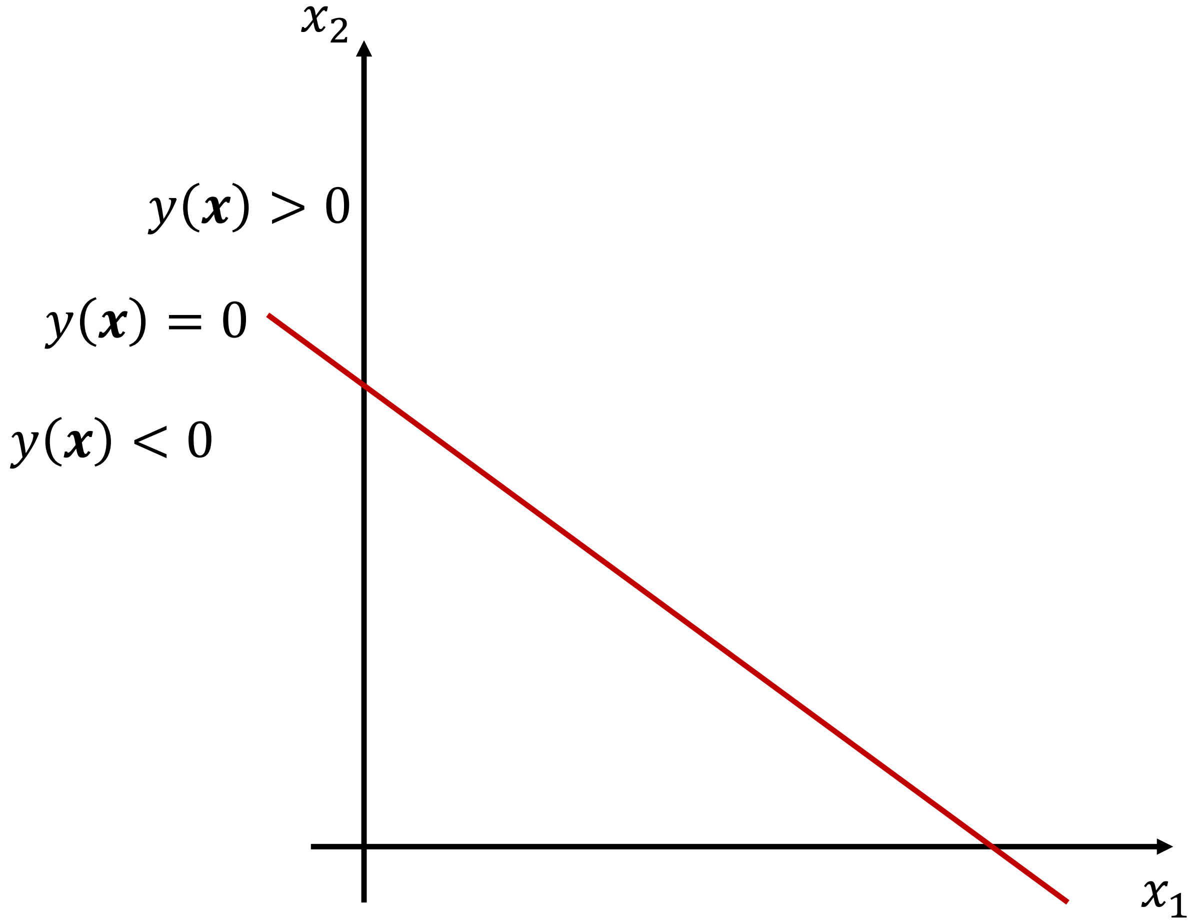
- The decision boundary (or surface) is defined by $y(\boldsymbol{x}) = 0$.
  - for a d-dimensional feature space I get a $(d-1)$-dimensional (hyper) plane.



Linear planes – Two classes

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

- We assign every input vector to ω_1 if $y(\boldsymbol{x}) \geq 0$ and to ω_2 otherwise.
- The decision boundary (or surface) is defined by $y(\boldsymbol{x}) = 0$.
  - for a d-dimensional feature space I get a $(d-1)$-dimensional (hyper) plane.
- $\boldsymbol{w}$ is the weight vector and w_0 is the bias (or threshold)
- Can you tell how $\boldsymbol{w}$ is related to every vector on the decision boundary $y(\boldsymbol{x}) = 0$?



Linear planes – Two classes

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

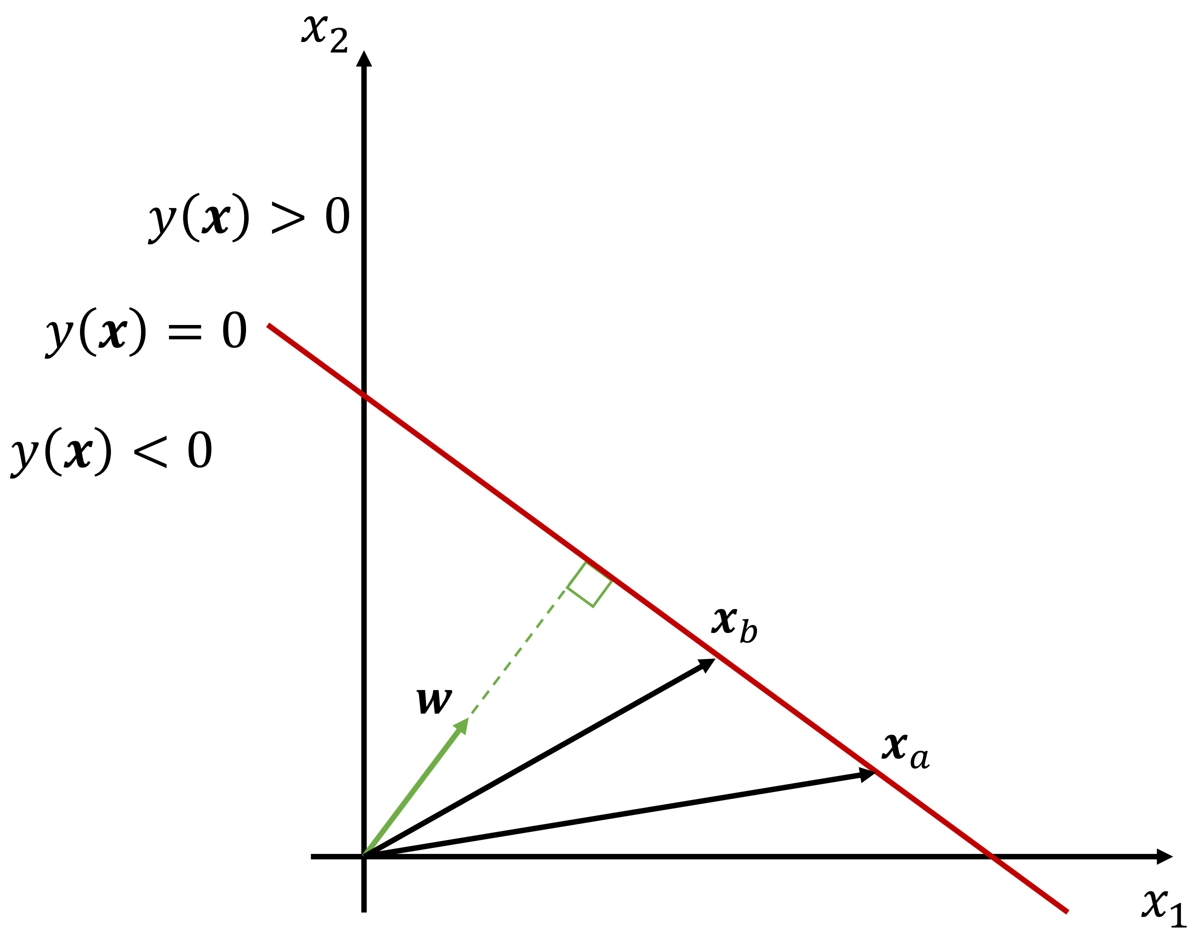
- If we consider two vectors $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ that both lie on the decision boundary then:

$$\boldsymbol{w}^t\boldsymbol{x}_a + w_0 = \boldsymbol{w}^t\boldsymbol{x}_b + w_0 \Leftrightarrow$$

$$\boldsymbol{w}^t(\boldsymbol{x}_a - \boldsymbol{x}_b) = 0$$

- Thus, $\boldsymbol{w}$ is orthogonal to the decision surface and to every vector on it.
  - Determines the orientation of the decision surface!

- What is the interpretation of the bias term w_0?



Linear planes – Two classes

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

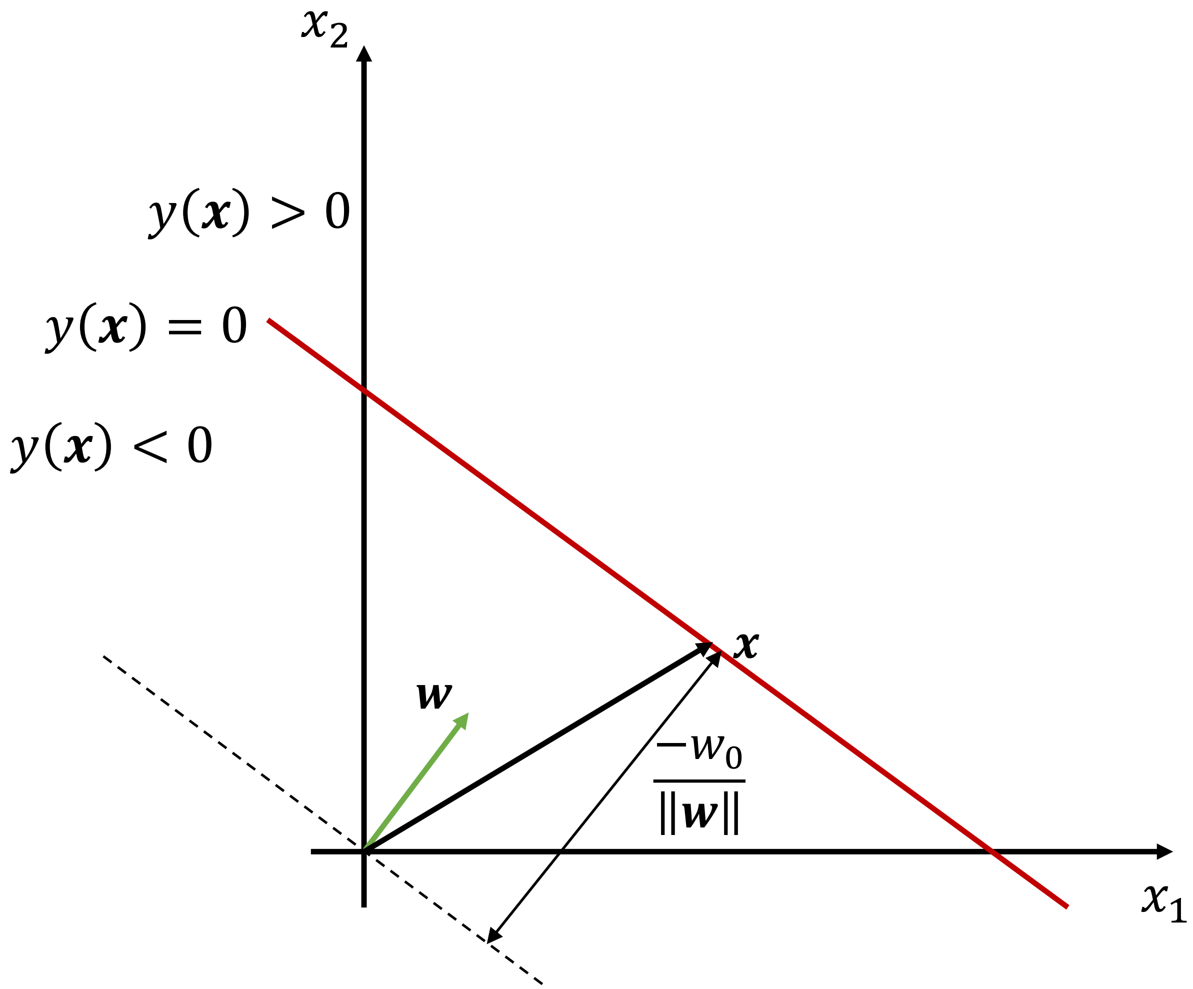
- If we consider a vector $\boldsymbol{x}$ that lie on the decision boundary then, $y(\boldsymbol{x}) = 0$ and:

$$\boldsymbol{w}^t\boldsymbol{x} + w_0 = 0 \Leftrightarrow$$

$$\frac{\boldsymbol{w}^t\boldsymbol{x}}{\|\boldsymbol{w}\|} = \frac{-w_0}{\|\boldsymbol{w}\|}$$

- Thus the bias term is actually the (signed) distance of the decision surface from the origin.

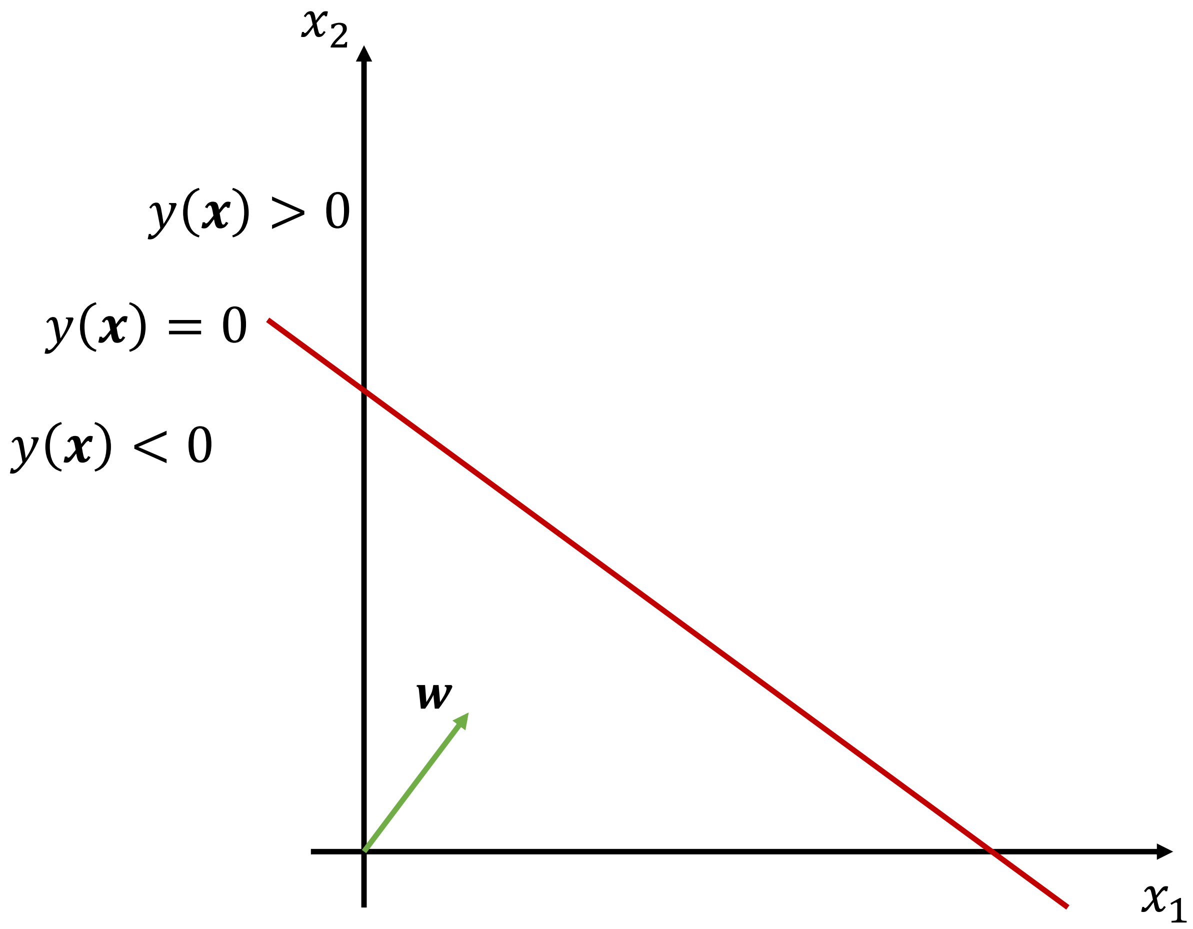
- It shifts the boundary away from the origin.



Linear planes – Two classes

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

- Finally what is the interpretation of the $y(\boldsymbol{x})$ value?



Linear planes – Two classes

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

- Finally what is the interpretation of the $y(\boldsymbol{x})$ value?

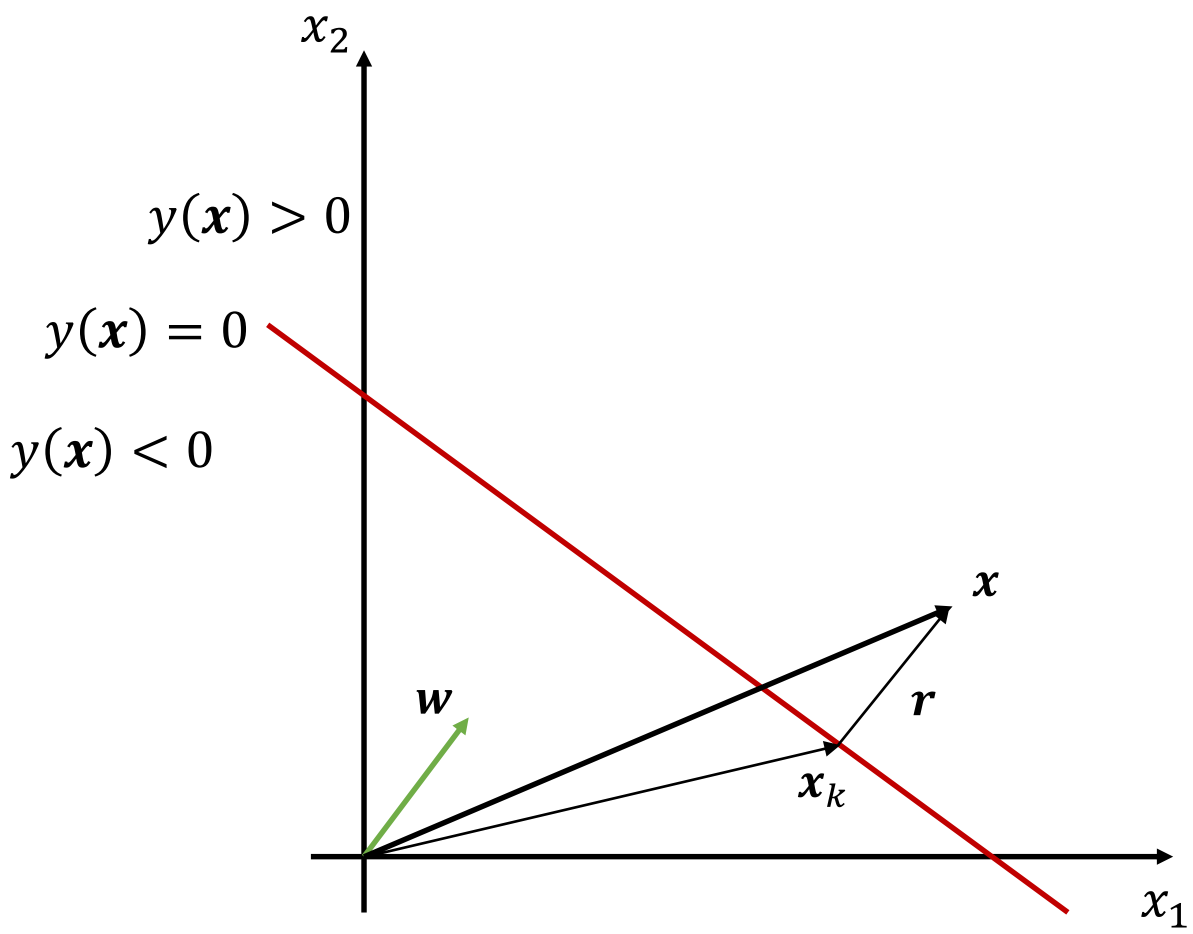
- If $\boldsymbol{x}$ is an arbitrary sample vector and $\boldsymbol{x}_k$ is its projection on the decision surface then:

$$\boldsymbol{x} = \boldsymbol{x}_k + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \Leftrightarrow$$

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = \boldsymbol{w}^T\boldsymbol{x}_k + r\frac{\boldsymbol{w}^T\boldsymbol{w}}{\|\boldsymbol{w}\|} + w_0 \Leftrightarrow$$

$$y(\boldsymbol{x}) = r\frac{\|\boldsymbol{w}\|^2}{\|\boldsymbol{w}\|} \Leftrightarrow r = \frac{y(\boldsymbol{x})}{\|\boldsymbol{w}\|}$$

- Thus, $y(\boldsymbol{x})$ determines the distance of the sample from the decision surface.



Least Squares

- We will see now how can we use the Least Squares approach to estimate $\boldsymbol{w}$ and w_0 (training!).

Least Squares

- We will see now how can we use the Least Squares approach to estimate $\boldsymbol{w}$ and w_0 (training!).
- Assume we have c classes and N training samples. Each one of the samples is associated with a target (label) which is structured using one-hot encoding.
  - For instance, if we have 5 classes and a sample $\boldsymbol{x}_n$ has $label = 3$ then the target vector using one-hot encoding is $\boldsymbol{t}_n = [0\ 0\ 1\ 0\ 0]$. We can gather all such target encoding in one matrix $\boldsymbol{T}$, where every row is the one-hot encoding of the target of the respective sample, i.e. $\boldsymbol{T} \in \mathbb{R}^{N\times c}$.

Least Squares

- We will see now how can we use the Least Squares approach to estimate $\boldsymbol{w}$ and w_0 (training!).
- Assume we have c classes and N training samples. Each one of the samples is associated with a target (label) which is structured using one-hot encoding.
  - For instance, if we have 5 classes and a sample $\boldsymbol{x}_n$ has $label = 3$ then the target vector using one-hot encoding is $\boldsymbol{t}_n = [0\ 0\ 1\ 0\ 0]$. We can gather all such target encoding in one matrix $\boldsymbol{T}$, where every row is the one-hot encoding of the target of the respective sample, i.e. $\boldsymbol{T} \in \mathbb{R}^{N \times c}$.
- Each of the c classes has its own linear discriminant:

$$y_i(\boldsymbol{x}) = \boldsymbol{w}_i^t \boldsymbol{x} + w_{i0}$$

- For simplicity we will adopt the shorter notation:

$$y_i(\boldsymbol{x}) = \widetilde{\boldsymbol{w}}_i^t \widetilde{\boldsymbol{x}}, \text{ where } \widetilde{\boldsymbol{w}}_i = \begin{bmatrix} w_{i0} \\ w_{i1} \\ \vdots \\ w_{id} \end{bmatrix} \text{ and } \widetilde{\boldsymbol{x}} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$$

Least Squares

$$y_i(\boldsymbol{x}) = \boldsymbol{w}_i^t \boldsymbol{x}$$

Note: for the rest of the lecture we will use $\boldsymbol{w}, \boldsymbol{x}$ but we will mean $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{x}}$!

Least Squares

$$y_i(\boldsymbol{x}) = \boldsymbol{w}_i^t \boldsymbol{x}$$

Note: for the rest of the lecture we will use $\boldsymbol{w}, \boldsymbol{x}$ but we will mean $\tilde{\boldsymbol{w}}$ and $\tilde{\boldsymbol{x}}$!

- We can use matrix notation to include all the discriminants:

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{W}^T \boldsymbol{x}$$

where the columns of $\boldsymbol{W}$ contain the vectors $\boldsymbol{w}_i$ for each class, i.e. $\boldsymbol{W} \in \mathbb{R}^{(d+1)\times c}$, and

$$\boldsymbol{y}(\boldsymbol{x}) = \begin{bmatrix} y_1(\boldsymbol{x}) \\ y_2(\boldsymbol{x}) \\ \vdots \\ y_c(\boldsymbol{x}) \end{bmatrix}$$

- Thus the classification problem is formed as:

assign $\boldsymbol{x}$ to class ω_i if $i = \underset{j}{\operatorname{argmax}}\, y_j(\boldsymbol{x})$

Least Squares

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{W}^T\boldsymbol{x}$$

- Now that we have formulated our problem we only need to determine matrix $\boldsymbol{W}$.

Least Squares

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{W}^T \boldsymbol{x}$$

- Now that we have formulated our problem we only need to determine matrix $\boldsymbol{W}$.
- Our objective is to make, for each sample $\boldsymbol{x}_n$, as minimum as possible the difference:

$$\boldsymbol{W}^T \boldsymbol{x}_n - \boldsymbol{t}_n^T$$

Least Squares

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{W}^T \boldsymbol{x}$$

- Now that we have formulated our problem we only need to determine matrix $\boldsymbol{W}$.
- Our objective is to make, for each sample $\boldsymbol{x}_n$, as minimum as possible the difference:

$$\boldsymbol{W}^T \boldsymbol{x}_n - \boldsymbol{t}_n^T$$

if we use the matrices $\boldsymbol{T}$ and $\boldsymbol{X}$, where every row of $\boldsymbol{X}$ is a sample of the training set, i.e. $\boldsymbol{X} \in \mathbb{R}^{N \times (d+1)}$, then our optimization problem can be described as the minimization of the sum-of-squares error function:

$$\boldsymbol{E}(\boldsymbol{W}) = \frac{1}{2} tr[(\boldsymbol{X}\boldsymbol{W} - \boldsymbol{T})^T (\boldsymbol{X}\boldsymbol{W} - \boldsymbol{T})]$$

Least Squares

- In order to minimize the sum-of-squares error function (least squares approach):

$$\boldsymbol{E}(\boldsymbol{W}) = \frac{1}{2} tr[(\boldsymbol{XW} - \boldsymbol{T})^T (\boldsymbol{XW} - \boldsymbol{T})]$$

We will find the respective derivative and set equal to zero:

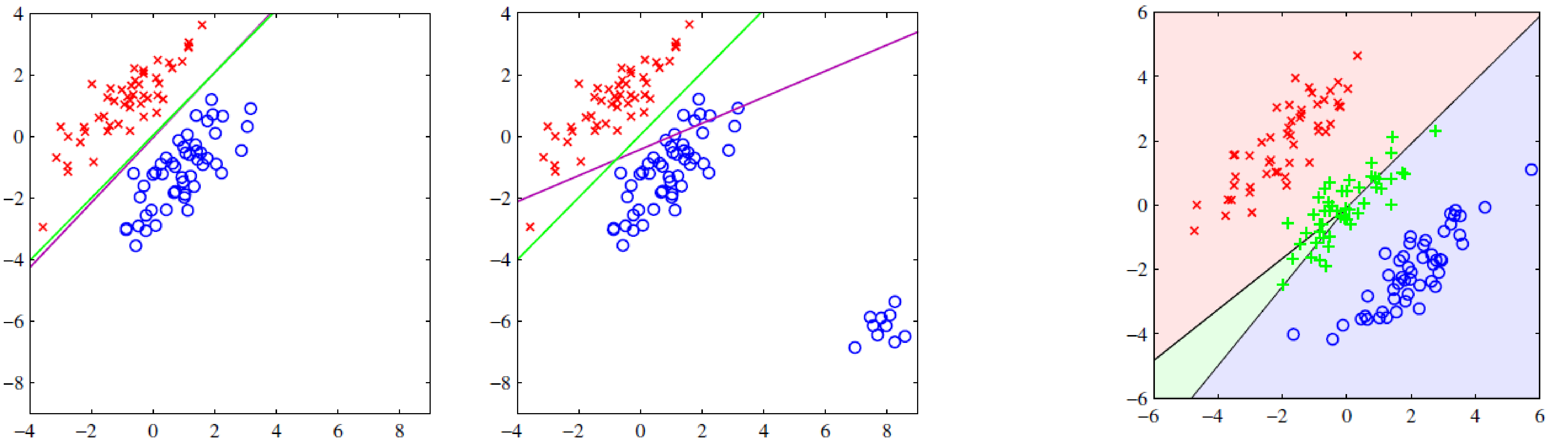
$$\frac{\partial \boldsymbol{E}(\boldsymbol{W})}{\partial \boldsymbol{W}} = 0$$

we will come up with the solution:

$$\boldsymbol{W}_{LS} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{T} = \boldsymbol{X}^{\dagger} \boldsymbol{T}$$

- Classification using the discriminant function: $\boldsymbol{y}_{LS}(\boldsymbol{x}) = \boldsymbol{W}_{LS}\boldsymbol{x}$

Least Squares – Problems!



- The decision boundaries are very sensitive to outliers.
- for $c > 2$ there are masking phenomena (decision regions become very small or completely ignored.
- $\boldsymbol{y}_{LS}(\boldsymbol{x})$ are not real probabilities and sometimes can have values outside of the range $[0,1]$.
- Very costly to compute!

Perceptron

- Perceptron algorithm is only for two-class problems.

Perceptron

- Perceptron algorithm is only for two-class problems.

- Assuming input $\boldsymbol{x} \in \mathbb{R}^{d+1}$ (again we consider the augmented samples $[1, x]$)

Perceptron

- Perceptron algorithm is only for two-class problems.
- Assuming input $\boldsymbol{x} \in \mathbb{R}^{d+1}$ (again we consider the augmented samples $[1, x]$)
- We also assume that we have targets $t \in \{\omega_1, \omega_2\} \rightarrow t \in \{-1, 1\}$

Perceptron

- Perceptron algorithm is only for two-class problems.

- Assuming input $\boldsymbol{x} \in \mathbb{R}^{d+1}$ (again we consider the augmented samples $[1, x]$)

- We also assume that we have targets $t \in \{\omega_1, \omega_2\} \rightarrow t \in \{1, -1\}$

- Thus, we can make a prediction based on outcome: $y(\boldsymbol{x}) = f(\boldsymbol{w}^T \boldsymbol{x})$ where $f(a) = \begin{cases} 1, & a > 0 \\ -1, & a \leq 0 \end{cases}$

- With this adoption I decide to assign to class ω_1 if $\boldsymbol{w}^T \boldsymbol{x} > 0$ and to class ω_2 otherwise.

$t \in \{\omega_1, \omega_2\} \rightarrow t \in \{-1, 1\}$

ω_1 if $\boldsymbol{w}^T \boldsymbol{x} \geq 0$

Perceptron - Training

- Assume we have N training samples, x_n, $n = 1, \dots, N$ with the respective targets (labels) t_n.

- Objective: find $\boldsymbol{w}$ such that for all (x_n, t_n) pairs I get:

$$w^T x_n t_n > 0$$

- What is the objective function that we should minimize?

$t \in \{\omega_1, \omega_2\} \rightarrow t \in \{-1, 1\}$

ω_1 if $\boldsymbol{w}^T \boldsymbol{x} \geq 0$

Perceptron - Training

- Assume we have N training samples, $x_n, n = 1, \dots, N$ with the respective targets (labels) t_n.

- Objective: find $\boldsymbol{w}$ such that for all (x_n, t_n) pairs I get:

$$w^T x_n t_n > 0$$

- What is the objective function that we should minimize?

$$E_P(\boldsymbol{w}) = -\sum_{n \in \mathcal{M}} \boldsymbol{w}^t \boldsymbol{x}_n t_n$$

where $\mathcal{M}: \{n: \boldsymbol{w}^t \boldsymbol{x}_n t_n \leq 0\}$.

Perceptron - Training

$$E_P(\boldsymbol{w}) = -\sum_{n\in\mathcal{M}} \boldsymbol{w}^t \boldsymbol{x}_n t_n \text{ where } \mathcal{M}: \{n: \boldsymbol{w}^t \boldsymbol{x}_n t_n \leq 0\}$$

- $E_P(\boldsymbol{w})$ is piecewise linear. Why?

Perceptron - Training

$$E_P(\boldsymbol{w}) = -\sum_{n\in\mathcal{M}} \boldsymbol{w}^t \boldsymbol{x}_n t_n \text{ where } \mathcal{M}: \{n: \boldsymbol{w}^t \boldsymbol{x}_n t_n \leq 0\}$$

- $E_P(\boldsymbol{w})$ is piecewise linear. Why?
  - If we change $\boldsymbol{w}$ linearly then $E_P(\boldsymbol{w})$ changes also linearly until the point where there is a change in the number of misclassified samples!
  - At these points the partial derivative of $E_P(\boldsymbol{w})$ cannot be defined and, thus, the derivative of $E_P(\boldsymbol{w})$ is a discontinuous function!
- Thus, how shall we proceed with the training of the perceptron algorithm?

Perceptron - Training

$$E_P(\boldsymbol{w}) = -\sum_{n\in\mathcal{M}} \boldsymbol{w}^t \boldsymbol{x}_n t_n \text{ where } \mathcal{M}: \{n: \boldsymbol{w}^t \boldsymbol{x}_n t_n \leq 0\}$$

- $E_P(\boldsymbol{w})$ is piecewise linear. Why?
  - If we change $\boldsymbol{w}$ linearly then $E_P(\boldsymbol{w})$ changes also linearly until the point where there is a change in the number of misclassified samples!
  - At these points the partial derivative of $E_P(\boldsymbol{w})$ cannot be defined and, thus, the derivative of $E_P(\boldsymbol{w})$ is a discontinuous function!
- Thus, how shall we proceed with the training of the perceptron algorithm?
- Gradient Descent!

Gradient Descent

- The gradient of a function encodes all directional derivatives.
  - It points towards the direction of the steepest ascent.
  - the magnitude of the gradient is the rate of increase in that direction.
  - It is always perpendicular the contours of a function.

Gradient Descent

- The gradient of a function encodes all directional derivatives.
  - It points towards the direction of the steepest ascent.
  - the magnitude of the gradient is the rate of increase in that direction.
  - It is always perpendicular the contours of a function.

- Whenever the loss is a sum of error terms for each datapoint we can use Gradient Descent, e.g.:

$$E(\boldsymbol{w}) = -\sum_{n \in \mathcal{M}} \boldsymbol{w}^t \boldsymbol{x}_n t_n = -\sum_{n \in \mathcal{M}} E_n(\boldsymbol{w})$$

- In general we have: $E(\boldsymbol{w}) = \sum_{\boldsymbol{n}=\mathbf{1}}^{\boldsymbol{N}} E_n(\boldsymbol{w})$

Stochastic gradient descent

- **Gradient descent:** $\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - \eta \nabla E(\boldsymbol{w}^{i})$
  - where $E(\boldsymbol{w}^{i}) = \sum_{\boldsymbol{n}=\mathbf{1}}^{\boldsymbol{N}} E_n(\boldsymbol{w}^{i})$
- We can extend the above concept by adding stochasticity in our Gradient Descent algorithm (Stochastic Gradient Descent):

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - \eta \nabla E_n(\boldsymbol{w}^{i})$$

or

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - \eta \nabla \sum_{l=1}^{L} E_l(\boldsymbol{w}^{i}) \text{ \textit{(minibatch)}}$$

- Either with single points or minibatches what I compute is an approximation of the full gradient.

Stochastic Gradient Descent - Algorithm

- SGD steps:
  - Initialize w^0, choose η
  - Iterate over datapoints: $\boldsymbol{w}^{i+1} = \boldsymbol{w}^i - \eta \nabla \sum_{l=1}^{L} E_l(\boldsymbol{w}^i)$

Stochastic Gradient Descent - Algorithm

- SGD steps:
  - Initialize w^0, choose η
  - Iterate over datapoints: $\boldsymbol{w}^{i+1} = \boldsymbol{w}^i - \eta \nabla \sum_{l=1}^{L} E_l(\boldsymbol{w}^i)$

- If E is convex in $\boldsymbol{w}$ and η is small enough SGD will converge
  - very small η slow convergence
  - large η: probably it will not converge

- Scheduling for η is important (maybe decrease η with iterations)

Stochastic Gradient Descent - Algorithm

- SGD steps:
  - Initialize w^0, choose η
  - Iterate over datapoints: $\boldsymbol{w}^{i+1} = \boldsymbol{w}^i - \eta\nabla \sum_{l=1}^{L} E_l(\boldsymbol{w}^i)$

- If E is convex in $\boldsymbol{w}$ and η is small enough SGD will converge
  - very small η slow convergence
  - large η: probably it will not converge

- Scheduling for η is important (maybe decrease η with iterations)

- Single point gradients are not necessarily equal to zero as the full gradient may be. Thus this also helps to escape local minima.

Stochastic Gradient Descent - Algorithm

- SGD steps:
  - Initialize w^0, choose η
  - Iterate over datapoints: $\boldsymbol{w}^{i+1} = \boldsymbol{w}^i - \eta \nabla \sum_{l=1}^{L} E_l(\boldsymbol{w}^i)$

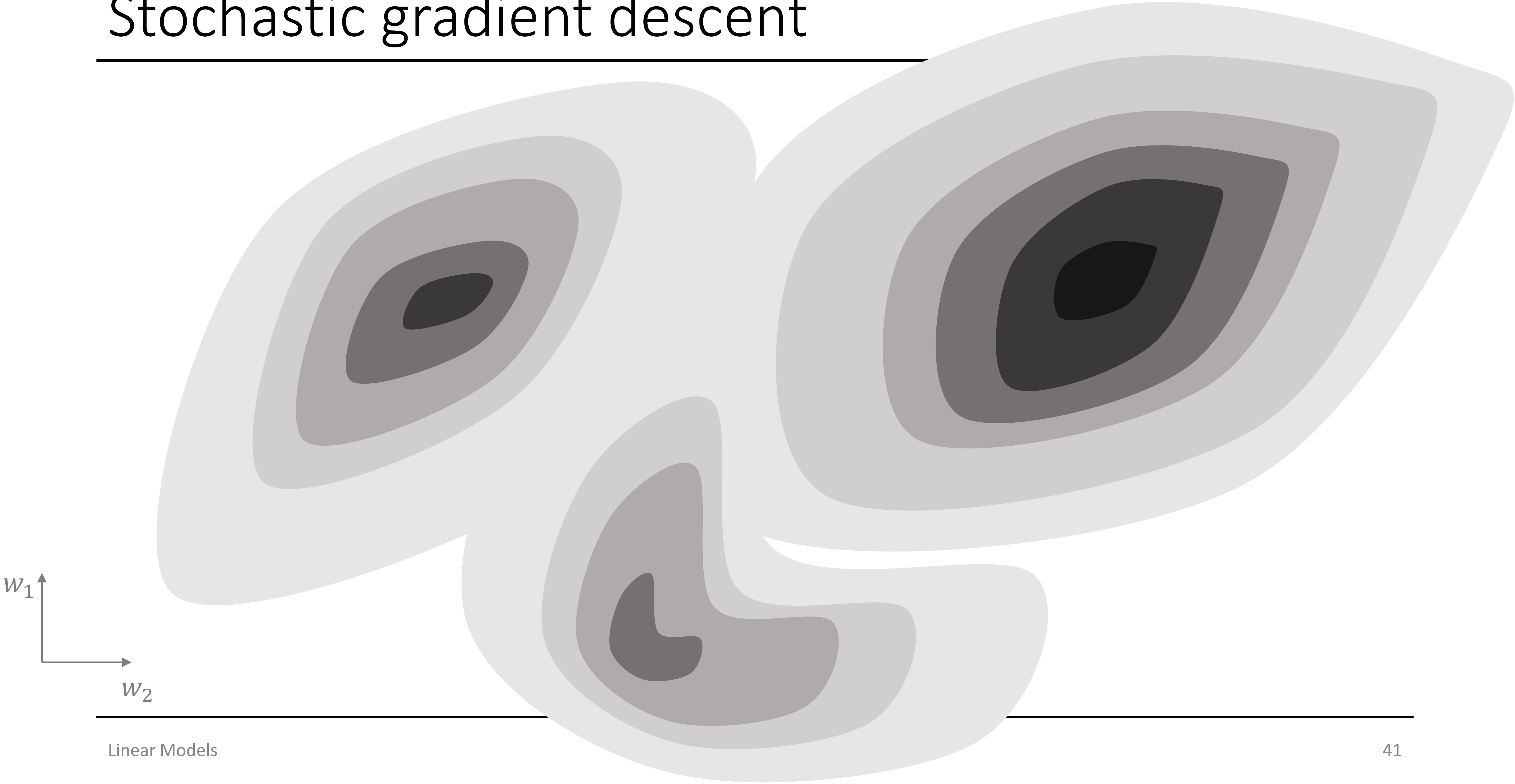
- If E is convex in $\boldsymbol{w}$ and η is small enough SGD will converge
  - very small η slow convergence
  - large η: probably it will not converge

- Scheduling for η is important (maybe decrease η with iterations)

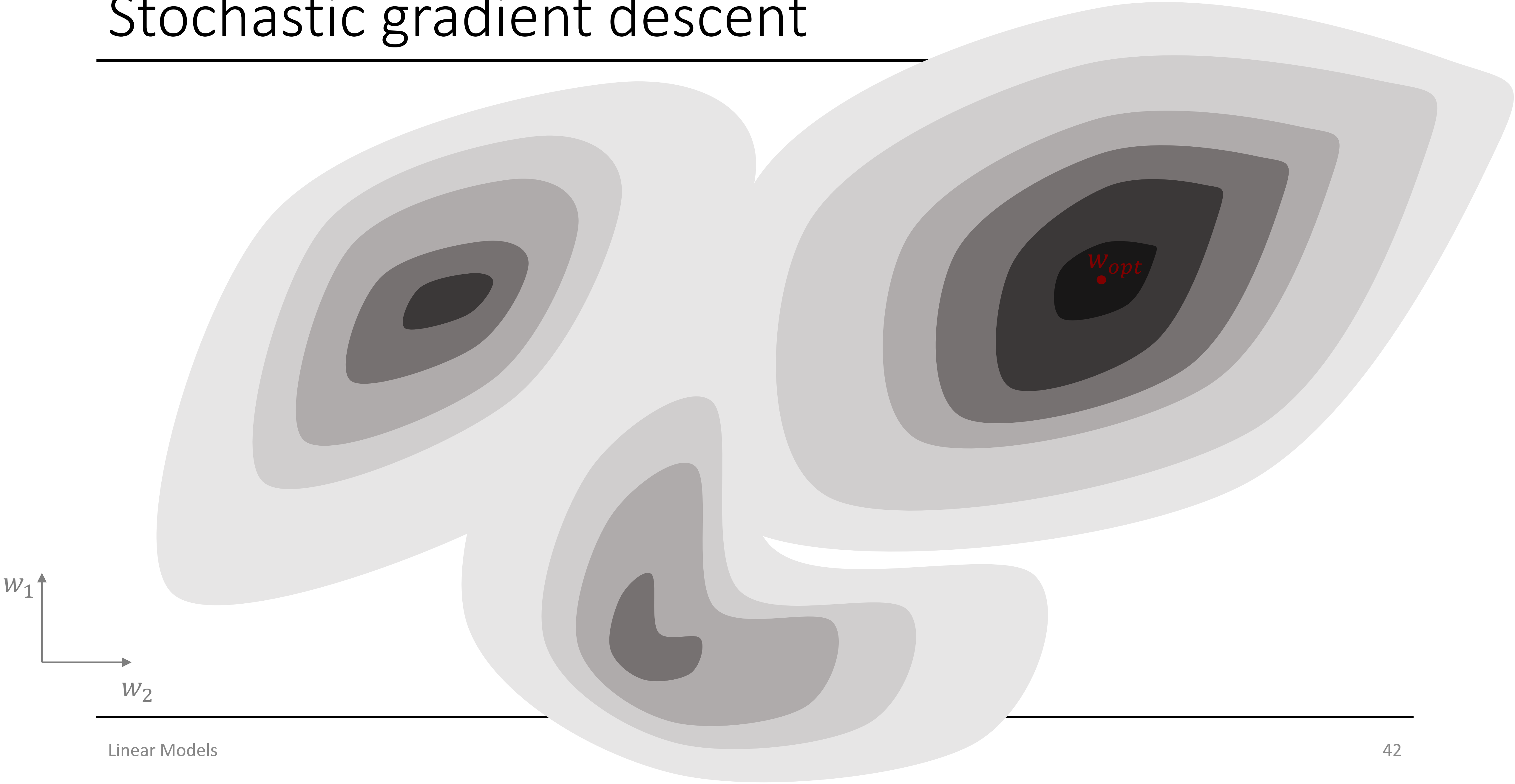
- Single point gradients are not necessarily equal to zero as the full gradient may be. Thus this also helps to escape local minima.

- Initialization plays crucial role!
  - maybe perform multiple initializations

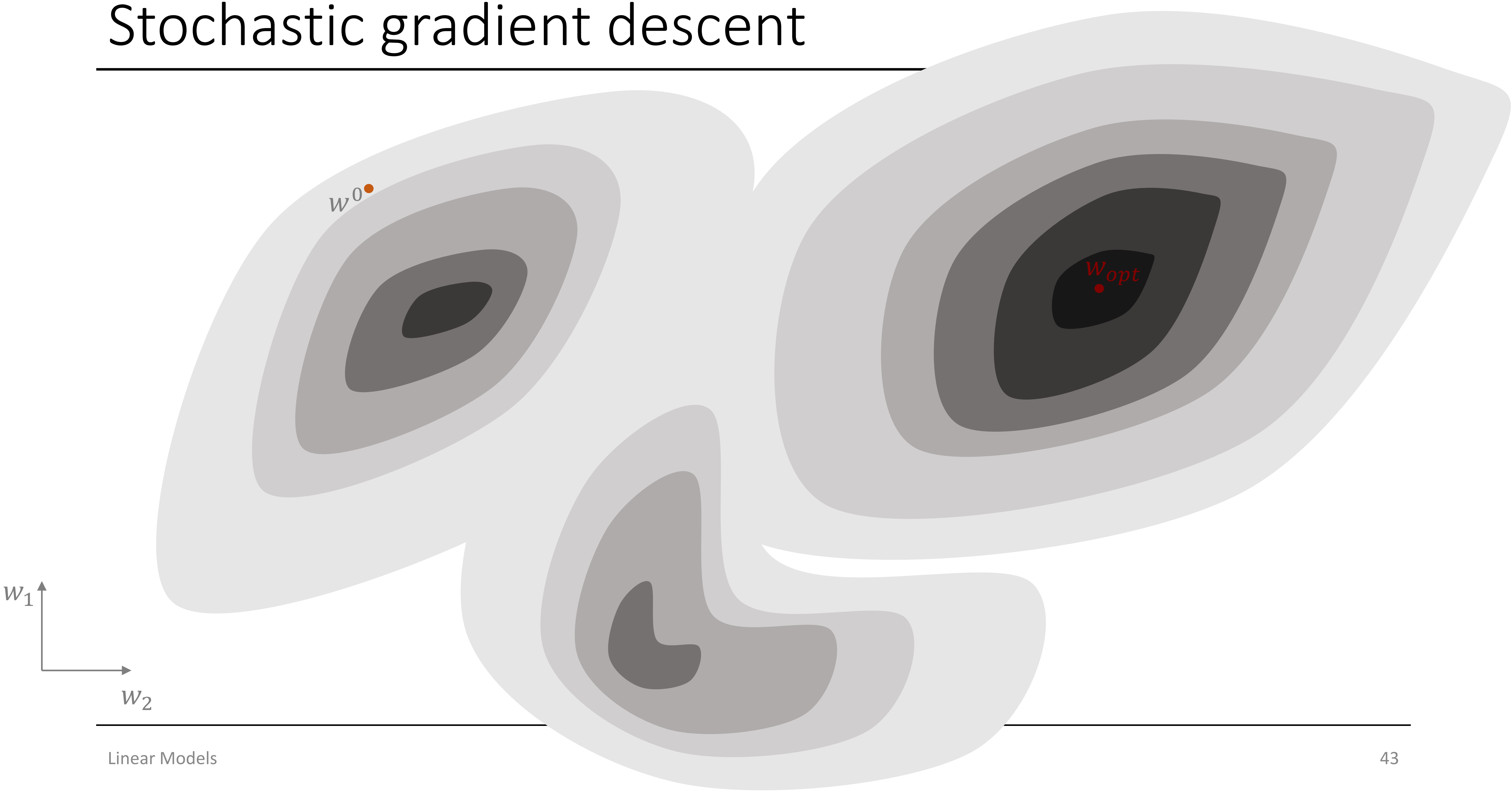
Stochastic gradient descent



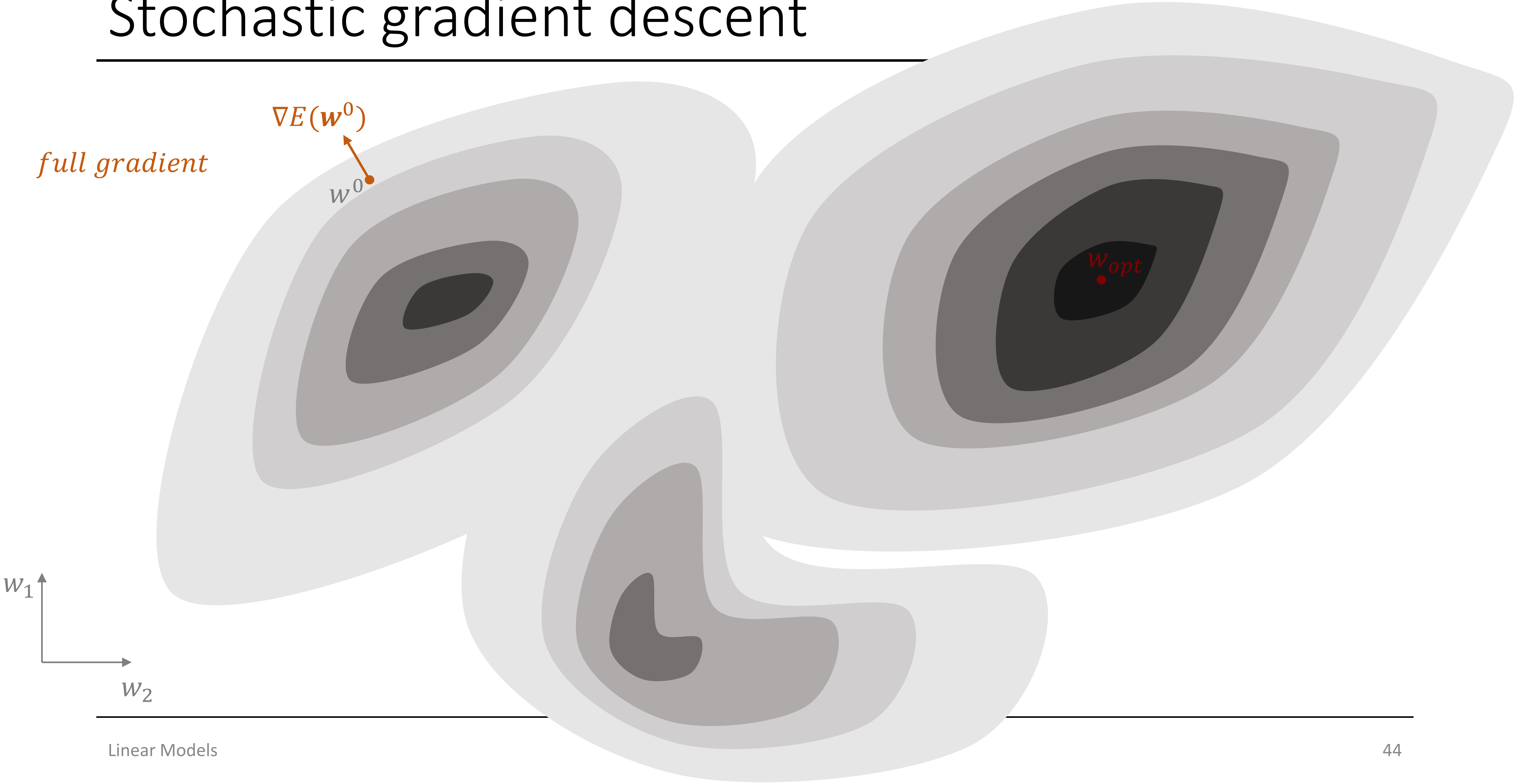
Stochastic gradient descent



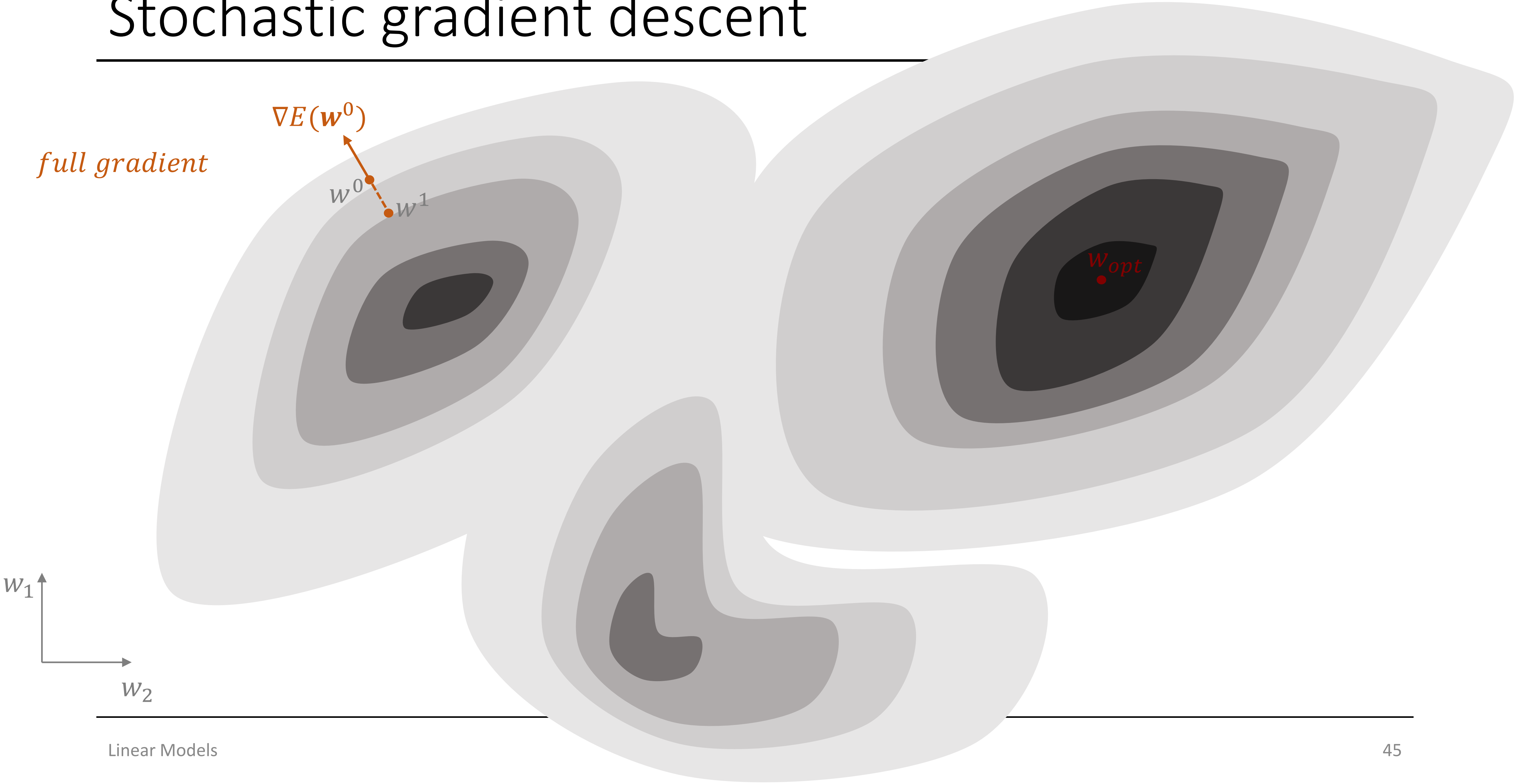
Stochastic gradient descent



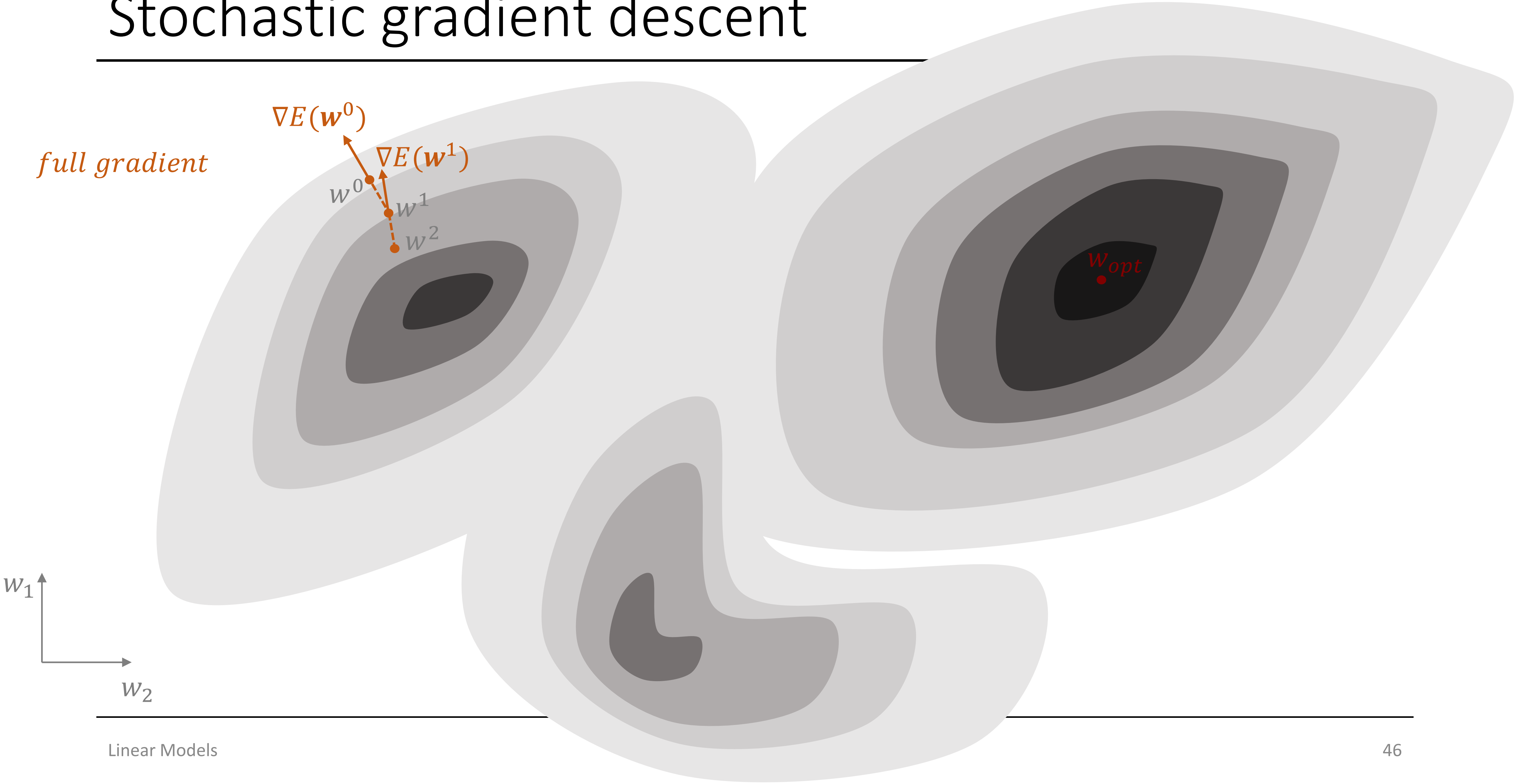
Stochastic gradient descent



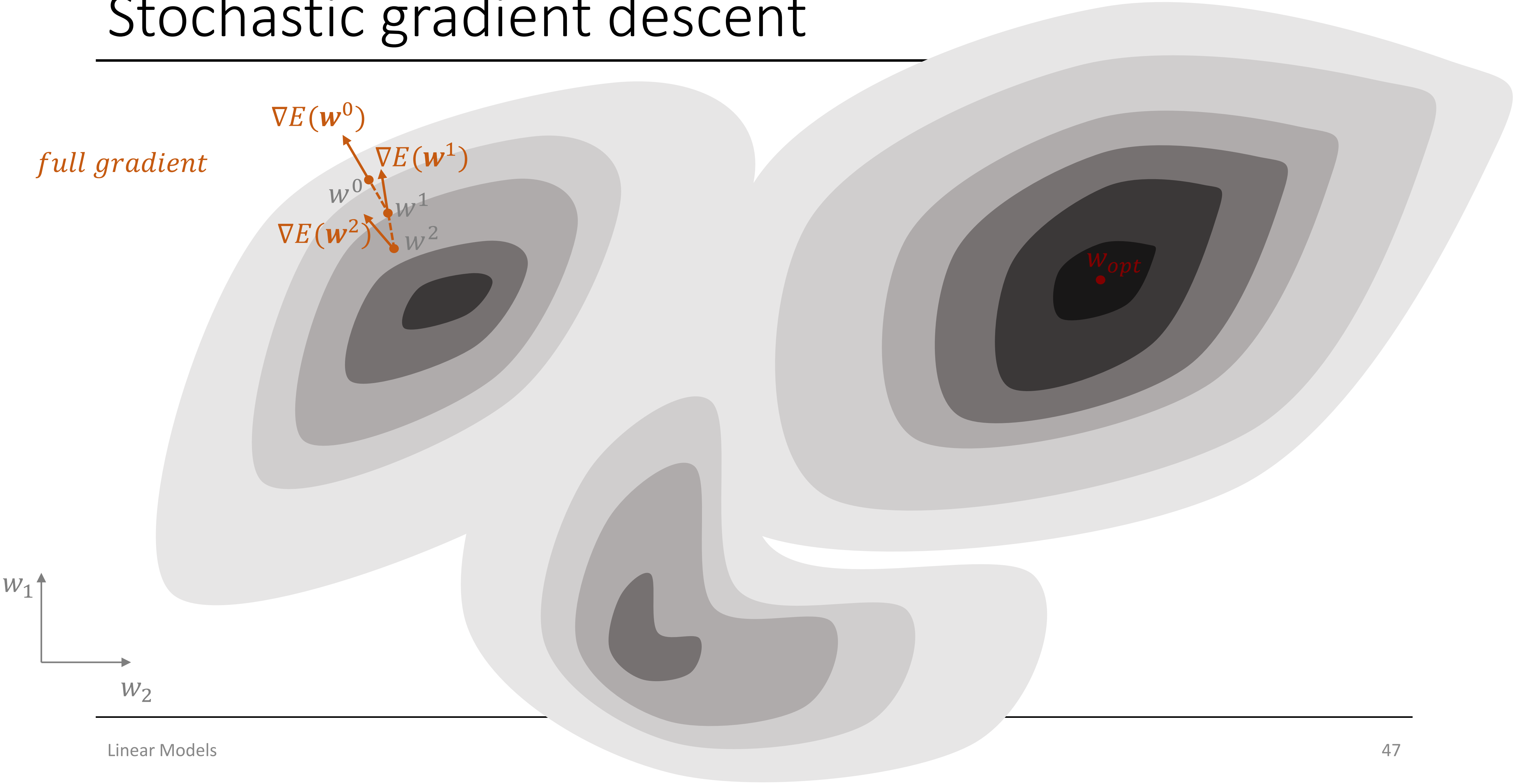
Stochastic gradient descent



Stochastic gradient descent

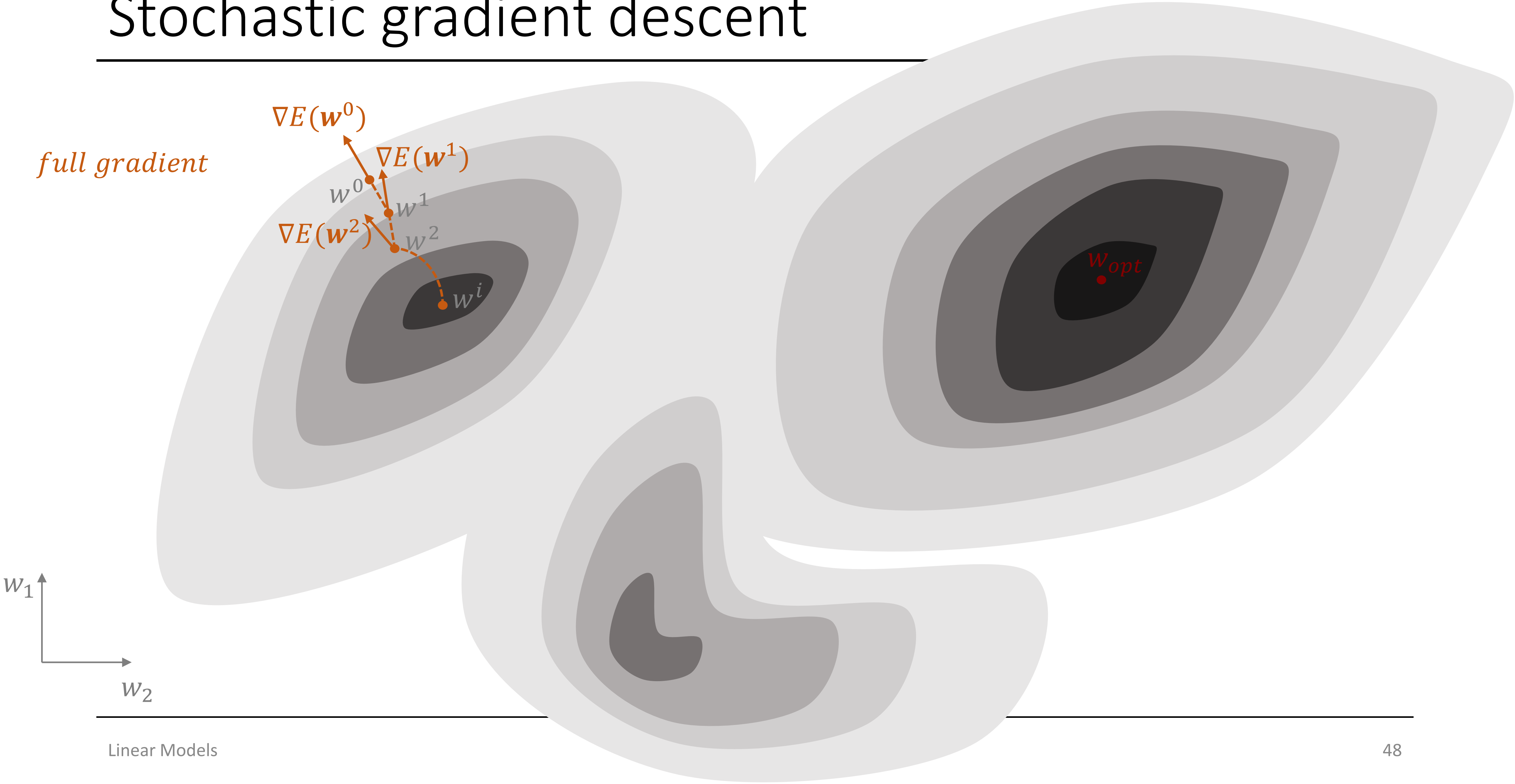


Stochastic gradient descent

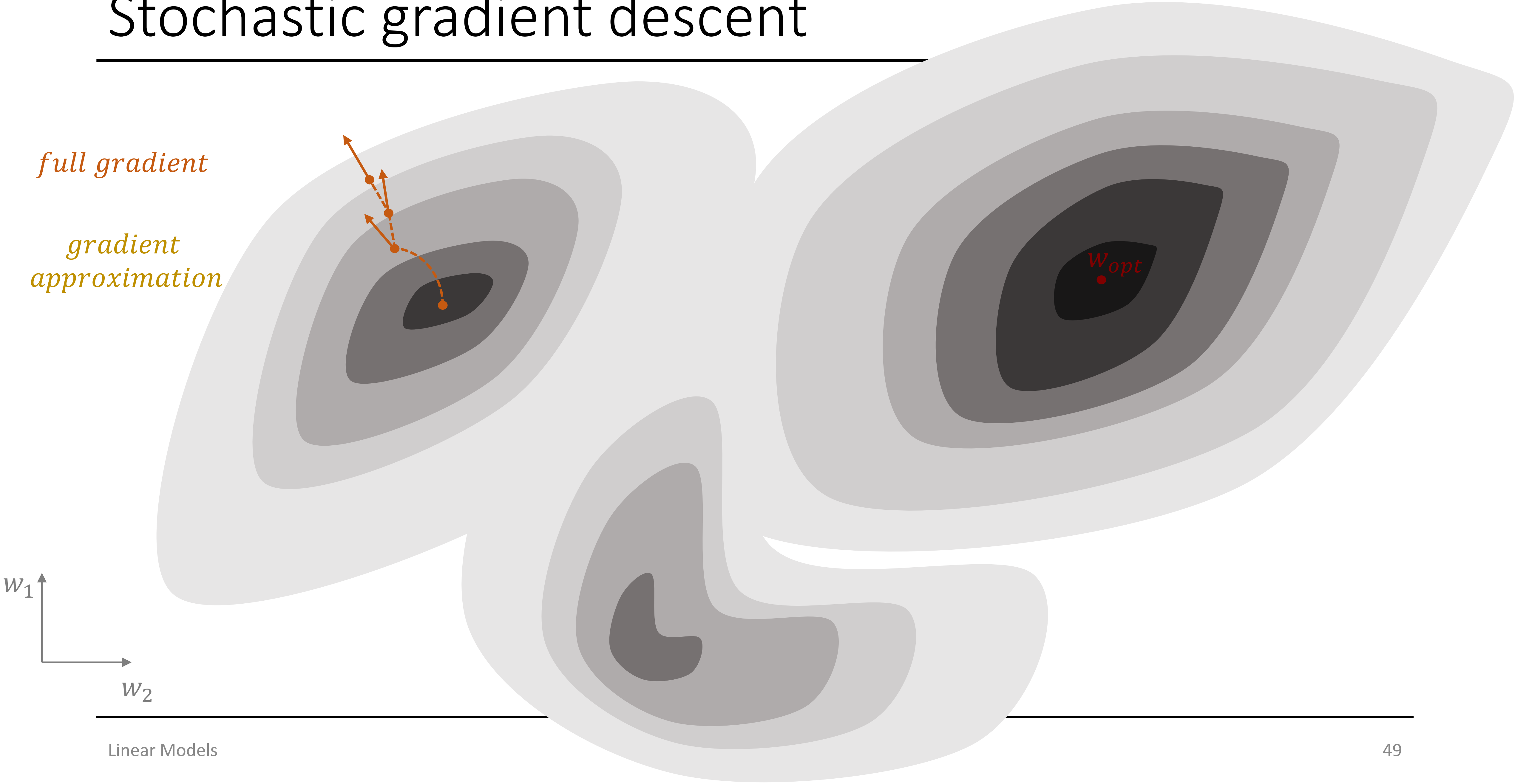


Stochastic gradient descent

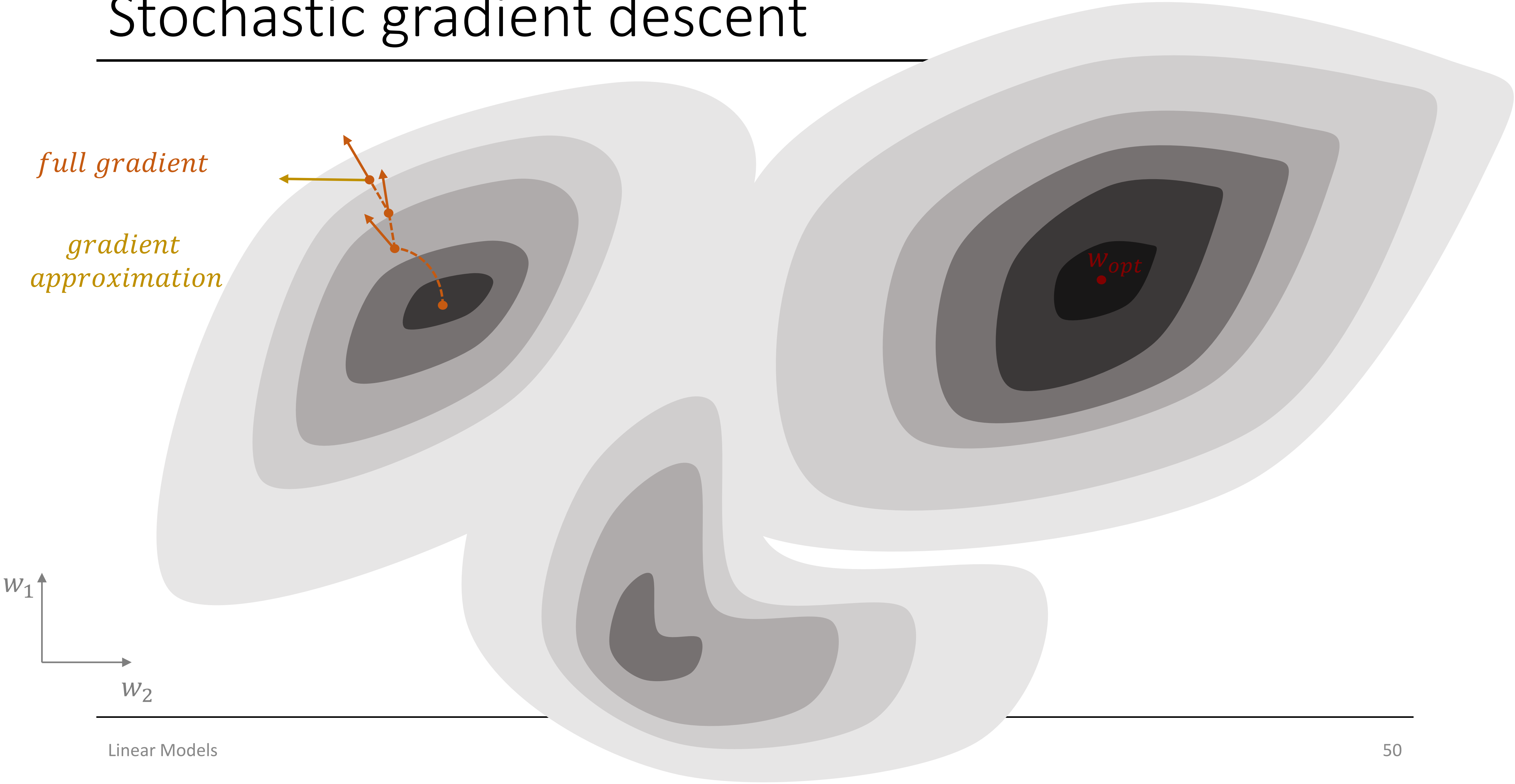
full gradient



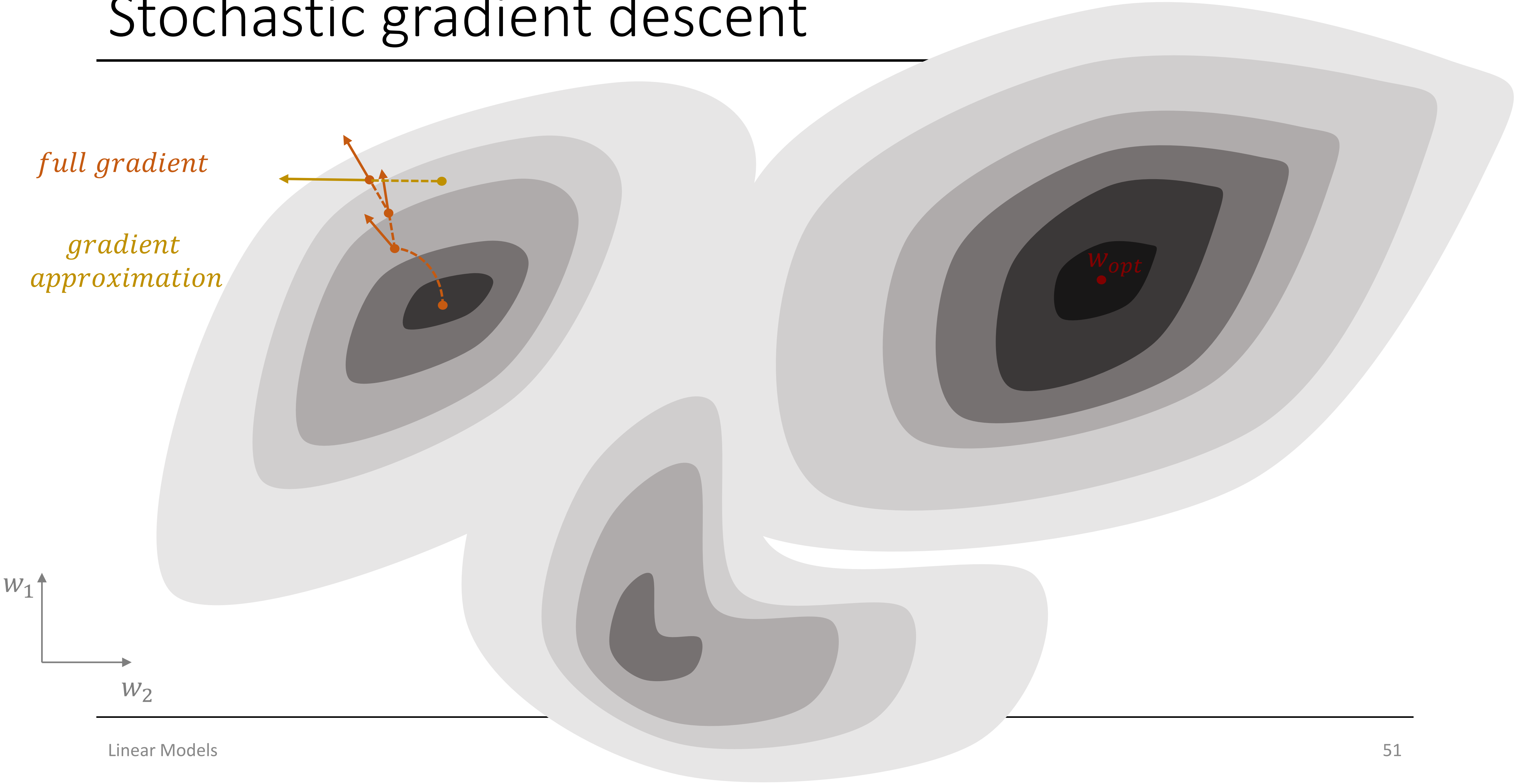
Stochastic gradient descent



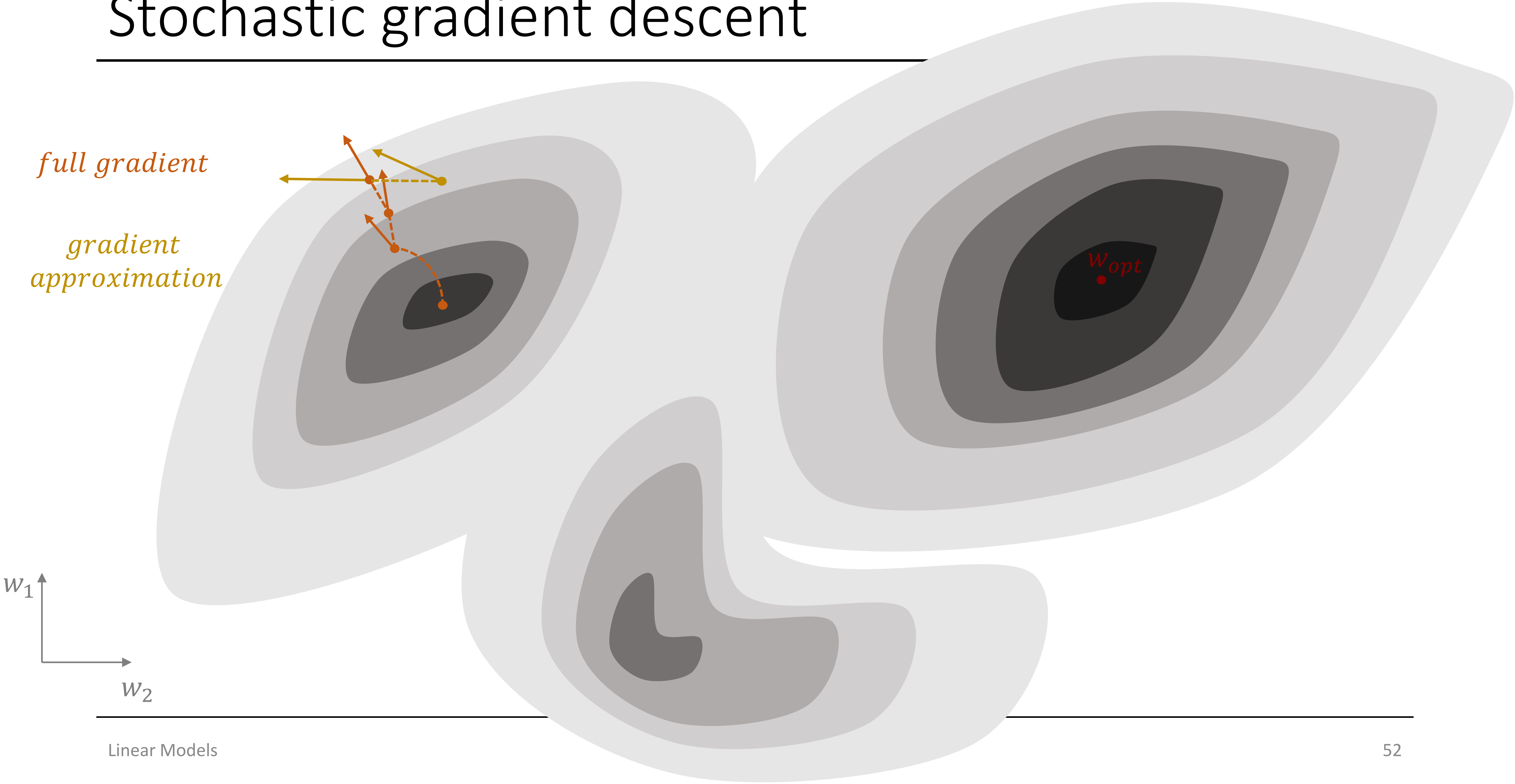
Stochastic gradient descent



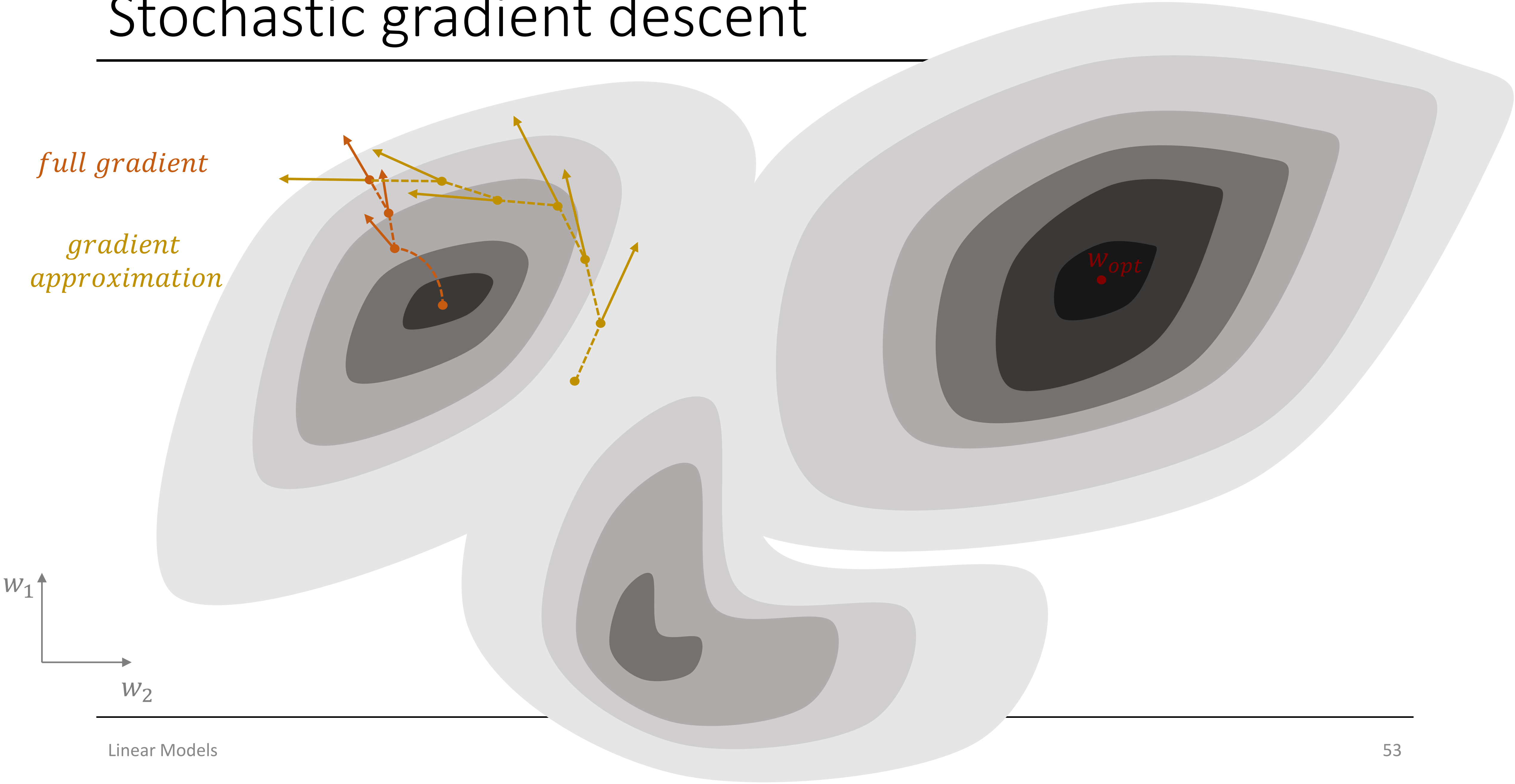
Stochastic gradient descent



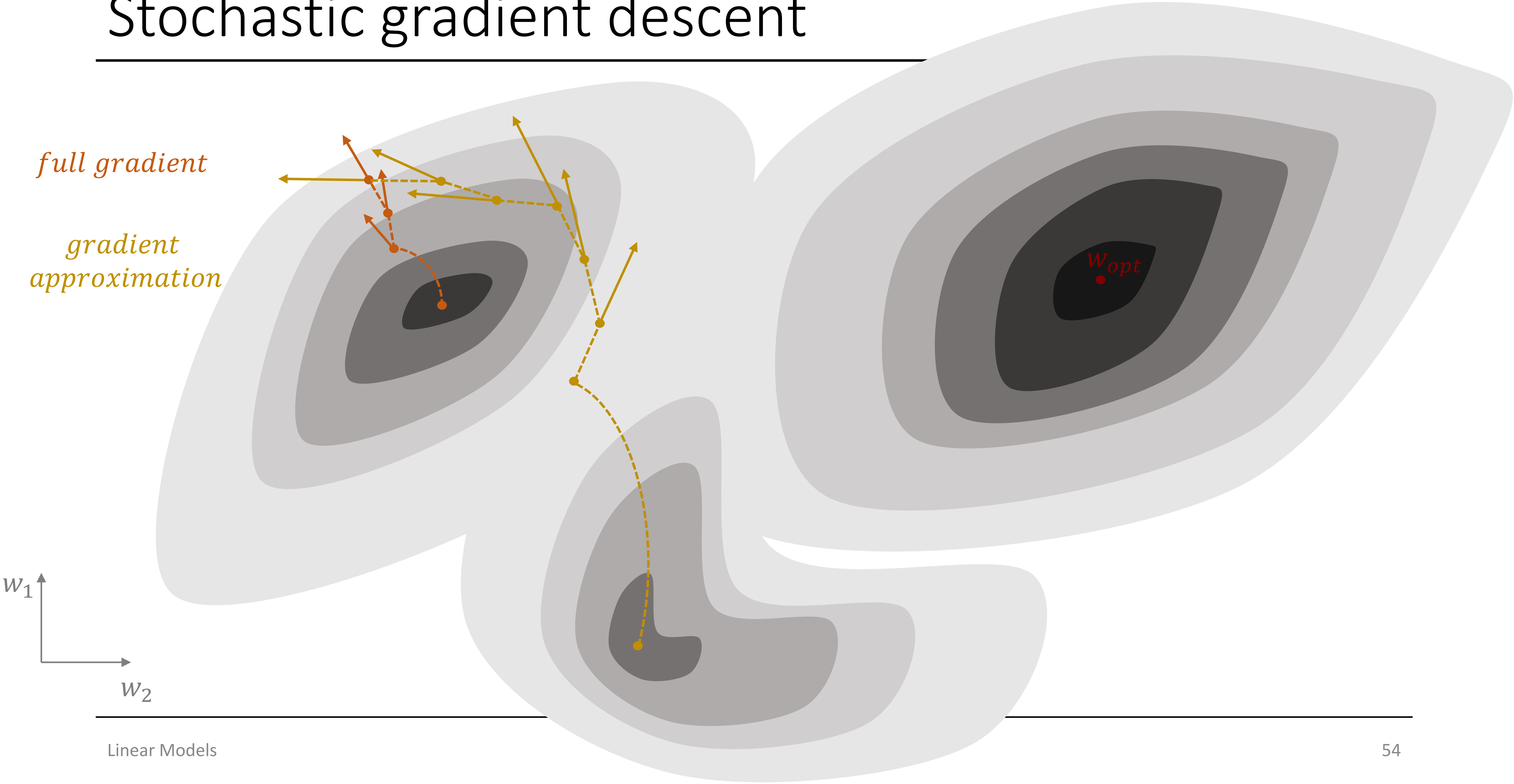
Stochastic gradient descent



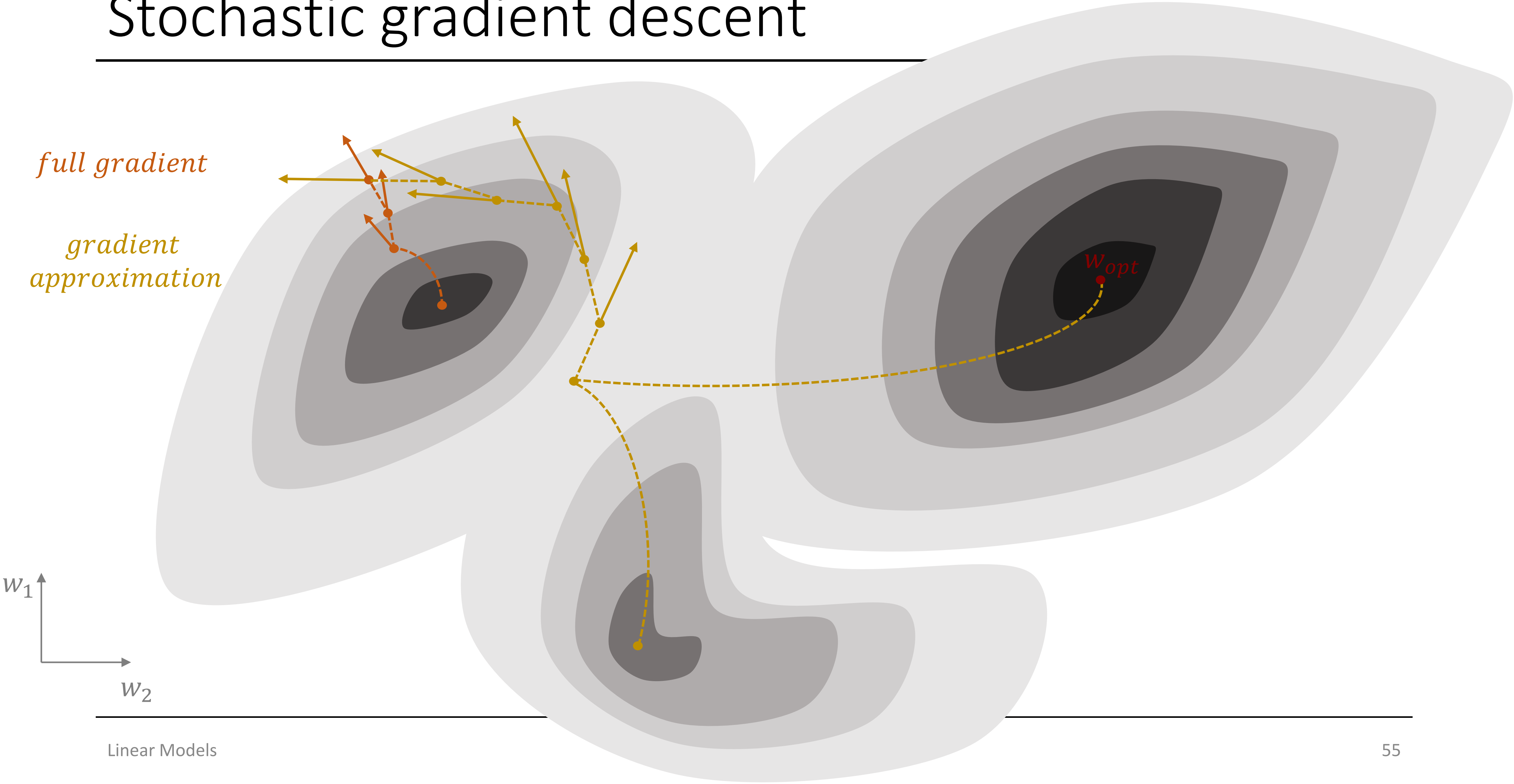
Stochastic gradient descent



Stochastic gradient descent



Stochastic gradient descent



Perceptron – Training : SGD

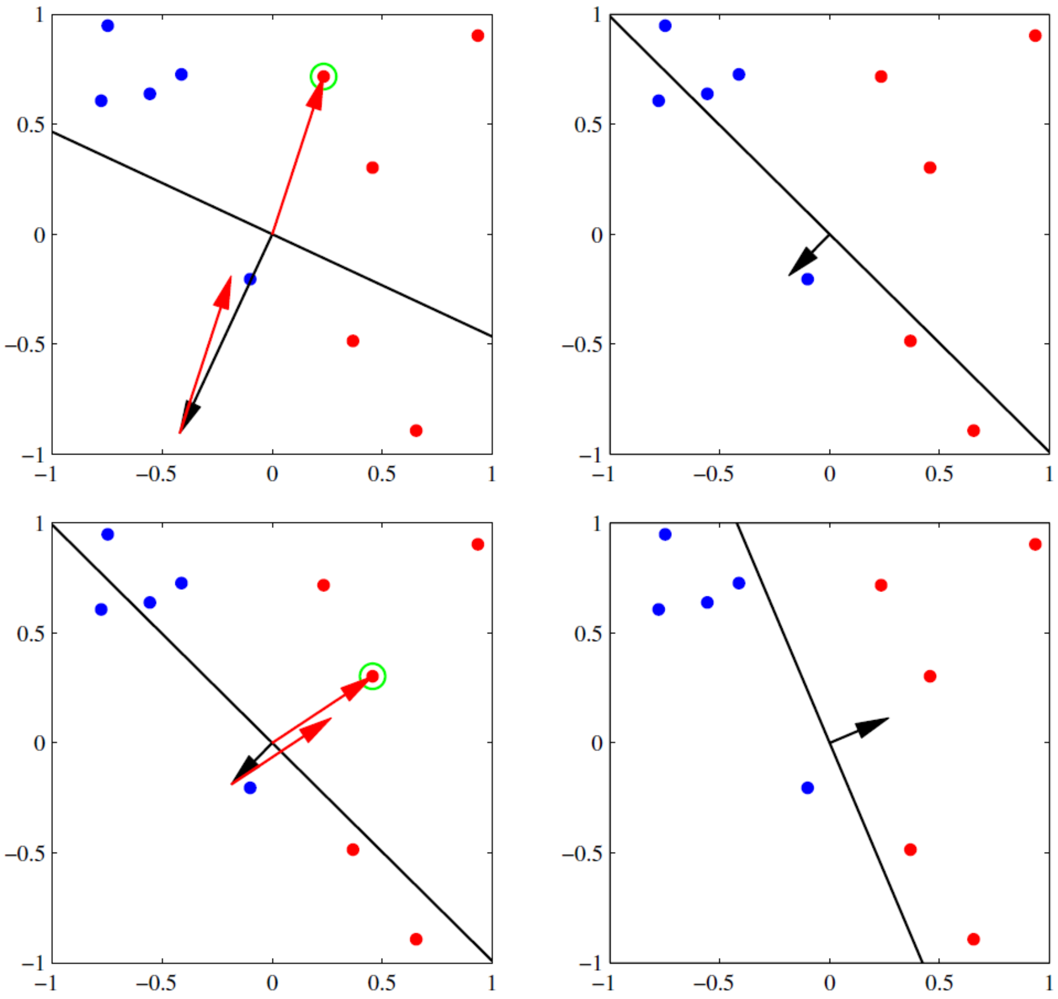
- Perceptron criterion:

$$E(\boldsymbol{w}) = -\sum_{n\in\mathcal{M}} \boldsymbol{w}^t \boldsymbol{x}_n t_n = -\sum_{n\in\mathcal{M}} E_n(\boldsymbol{w})$$

- Sequential learning:
  - Single sample SGD:

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^i - \eta \nabla E_n(\boldsymbol{w}^i)$$

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^i + \eta \boldsymbol{x}_n t_n$$



Bishop

Least Mean Squares (LMS) algorithm

- If we replace the perceptron criterion with a least squares one we get:

$$E_{LS}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - \boldsymbol{w}^t\boldsymbol{x})^2$$

- If we estimate the corresponding derivative:

$$\frac{\partial E_{LS}(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_{n=1}^{N}\frac{\partial \frac{1}{2}(t_n - \boldsymbol{w}^t\boldsymbol{x})^2}{\partial \boldsymbol{w}} = \sum_{n=1}^{N} -(t_n - \boldsymbol{w}^t\boldsymbol{x})\boldsymbol{x}^t$$

- We get the respective Sequential Learning algorithm (SGD):

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^i + \eta\left(t_n - \left(\boldsymbol{w}^i\right)^t\boldsymbol{x}\right)\boldsymbol{x}^t$$

LMS algorithm - Notes

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} + \eta\left(t_n - \left(\boldsymbol{w}^{i}\right)^{t}\boldsymbol{x}\right)\boldsymbol{x}^{t}$$

- LMS is a two-class algorithm as perceptron is.

LMS algorithm - Notes

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} + \eta\left(t_n - \left(\boldsymbol{w}^{i}\right)^{t}\boldsymbol{x}\right)\boldsymbol{x}^{t}$$

- LMS is a two-class algorithm as perceptron is.

- Remember that for the least squares solution that we previously investigated (multiclass problem) we got:

$$\boldsymbol{W}_{LS} = (\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{T}\boldsymbol{T} = \boldsymbol{X}^{\dagger}\boldsymbol{T}$$

LMS algorithm - Notes

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} + \eta\left(t_n - \left(\boldsymbol{w}^{i}\right)^{t}\boldsymbol{x}\right)\boldsymbol{x}^{t}$$

- LMS is a two-class algorithm as perceptron is.

- Remember that for the least squares solution that we previously investigated (multiclass problem) we got:

$$\boldsymbol{W}_{LS} = (\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{T}\boldsymbol{T} = \boldsymbol{X}^{\dagger}\boldsymbol{T}$$

- Matrix inverse is a computationally difficult to get. ($\mathcal{O}(N^3)$).

- With LMS we can overcome this problem and process data in minibatches or on a single sample base (not all data are necessary to be available at once!)

LMS algorithm - Notes

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} + \eta\left(t_n - \left(\boldsymbol{w}^{i}\right)^{t}\boldsymbol{x}\right)\boldsymbol{x}^{t}$$

- LMS is a two-class algorithm as perceptron is.

- Remember that for the least squares solution that we previously investigated (multiclass problem) we got:

$$\boldsymbol{W}_{LS} = (\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{T}\boldsymbol{T} = \boldsymbol{X}^{\dagger}\boldsymbol{T}$$

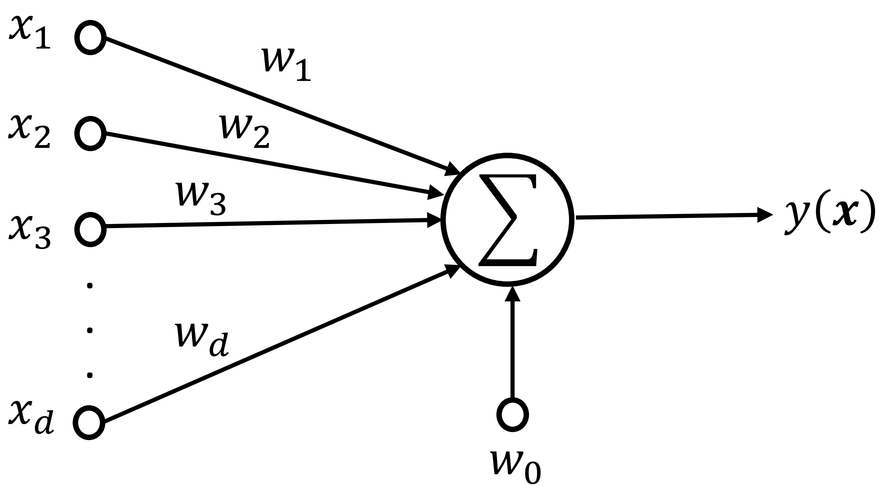
- Matrix inverse is a computationally difficult to get. ($\mathcal{O}(N^3)$).

- With LMS we can overcome this problem and process data in minibatches or on a single sample base (not all data are necessary to be available at once!)

- LMS algorithm is also called Widrow-Hoff algorithm.

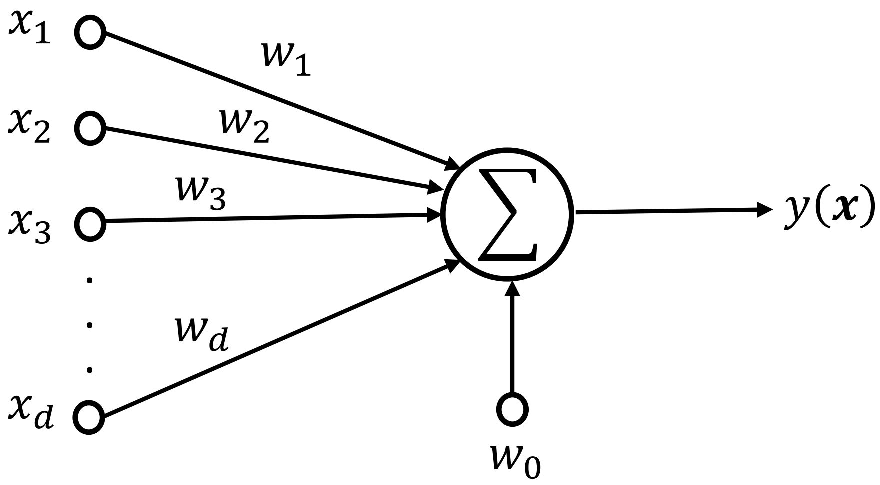
Neuron

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

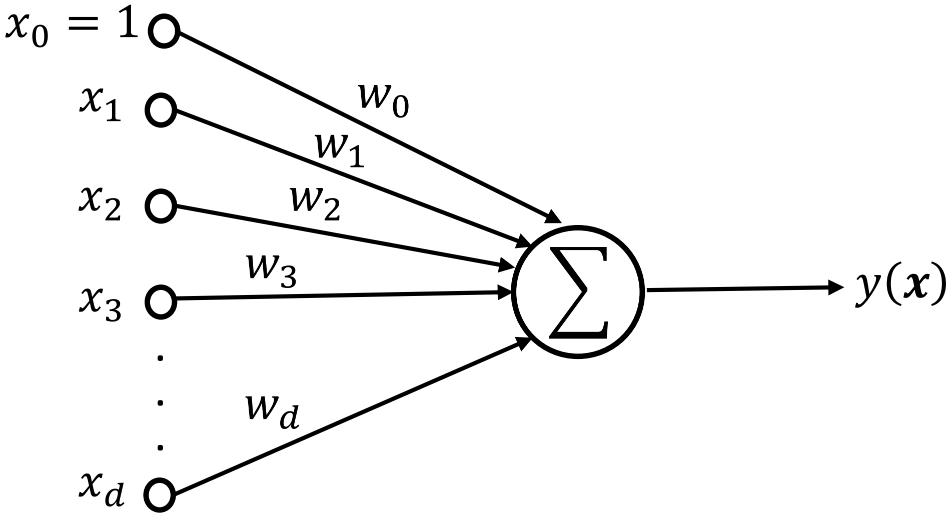


Neuron

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

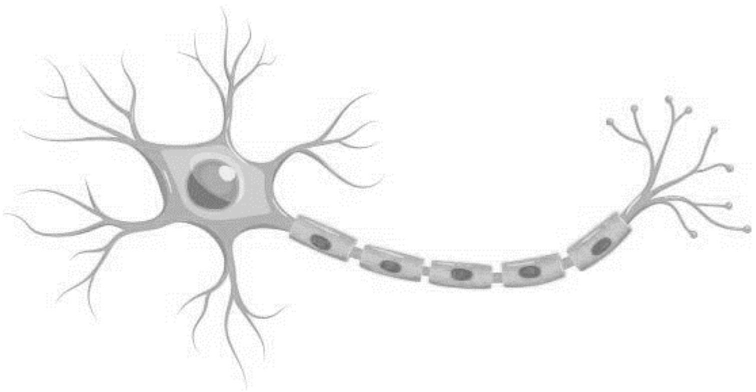


$$y(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x}$$



Neuron

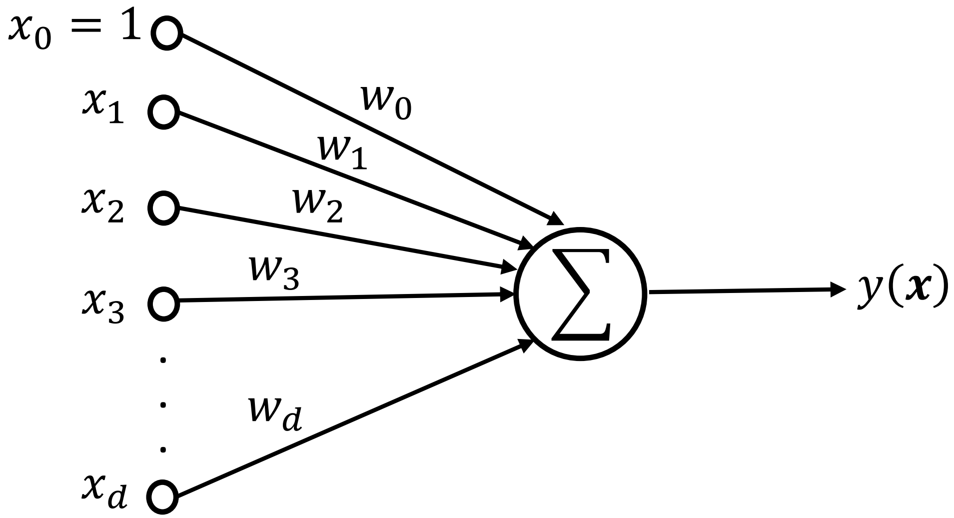
$$y(\boldsymbol{x}) = \boldsymbol{w}^t \boldsymbol{x}$$



- In essence, perceptron and LMS algorithms are two different approaches for the training of the model of a neuron.

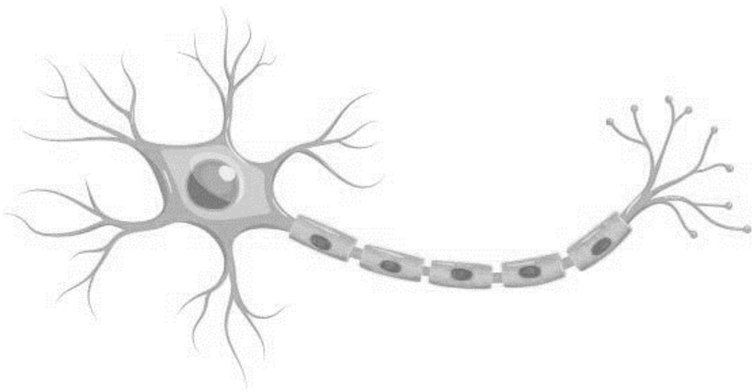
- Perceptron: Rosenblatt (trained with perceptron algorithm).

- Adaline (adaptive linear element): Widrow and Hoff (trained with LMS).



Neuron

$$y(\boldsymbol{x}) = \boldsymbol{w}^t \boldsymbol{x}$$

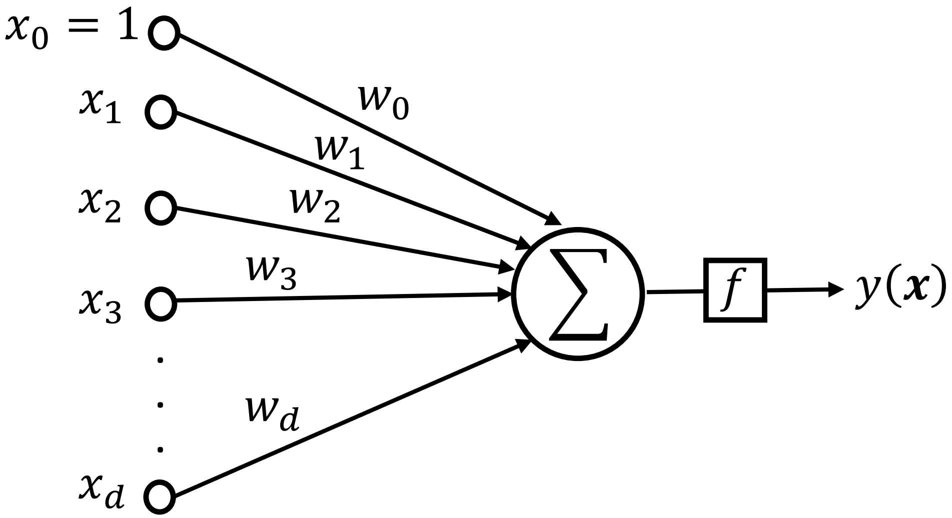


- In essence, perceptron and LMS algorithms are two different approaches for the training of the model of a neuron.

- Perceptron: Rosenblatt (trained with perceptron algorithm).

- Adaline (adaptive linear element): Widrow and Hoff (trained with LMS).

- This model is the fundamental module of Neural Networks using different activation functions.



Classification with basis functions

- So far we have considered models that work directly with the original input vector $\boldsymbol{x}$, e.g., $y(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x}$.

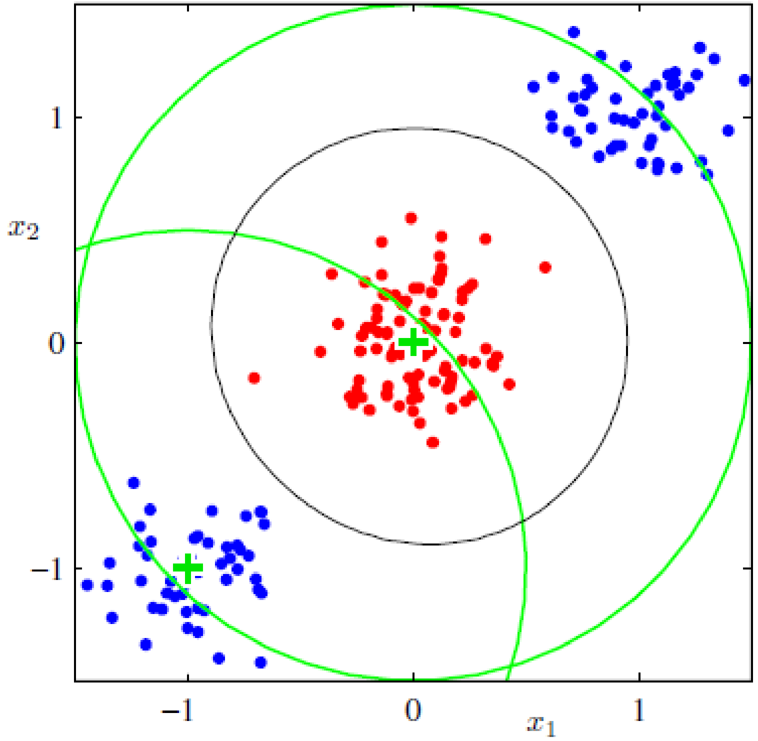
- Nevertheless, all of the algorithms are also applicable when we first non-linearly transform the input using a vector of basis functions $\boldsymbol{\varphi}(\boldsymbol{x})$, e.g.:

$$y(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{\varphi}(\boldsymbol{x})$$

- With this trick classes that are linearly separable in the feature space $\boldsymbol{\varphi}(\boldsymbol{x})$ do not need to be linearly separable in the observation space $\boldsymbol{x}$.

- **Note:** Basis function are fixed, not learned! They need to be selected carefully. They correspond also to sources of inductive bias.

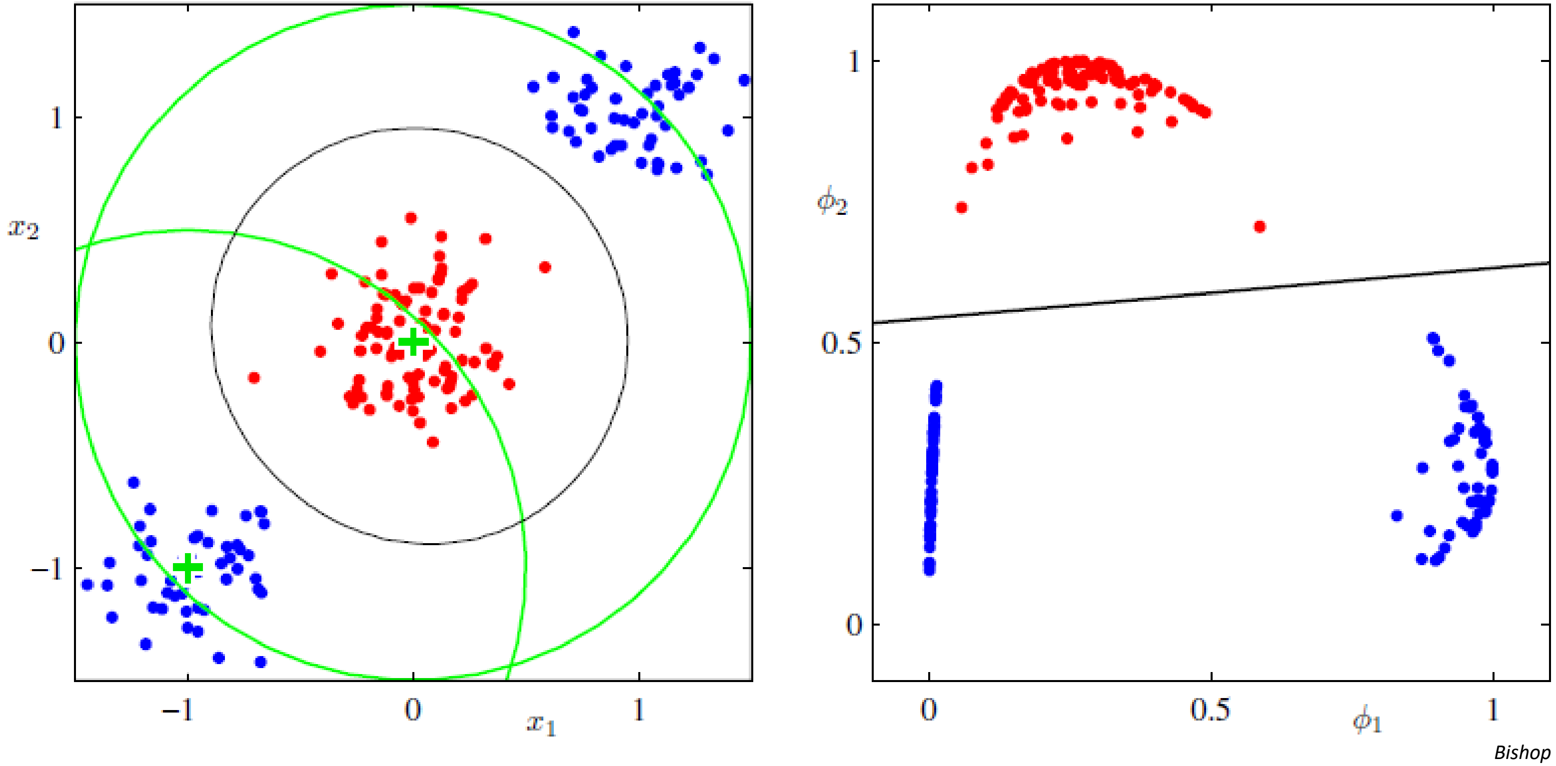
Classification with basis functions - Example



Left: original input space (x_1, x_2)

$$\varphi_1(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_1)^T(\boldsymbol{x}-\boldsymbol{\mu}_1)\right) \text{ and } \varphi_2(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_2)^T(\boldsymbol{x}-\boldsymbol{\mu}_2)\right)$$

Classification with basis functions - Example



Left: original input space (x_1, x_2) , **Right:** space of two gaussian basis functions with centers shown by the green crosses:

$$\varphi_1(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_1)^T(\boldsymbol{x} - \boldsymbol{\mu}_1\right) \text{ and } \varphi_2(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_2)^T(\boldsymbol{x} - \boldsymbol{\mu}_2\right)$$

Classification strategies (revisited)

- Probabilistic Generative Models
  - class-conditional densities $p(\boldsymbol{x}|\omega_i)$
  - prior class probabilities $P(\omega_i)$
  - Via Bayes we get posterior class probabilities $P(\omega_i|\boldsymbol{x})$

- Linear Discriminant functions
  - Direct mapping of input to target $t = y(\boldsymbol{x}, \boldsymbol{w})$

- Probabilistic Discriminative Models
  - Determining directly the posterior class probabilities $P(\omega_i|\boldsymbol{x})$

- Logistic Regression

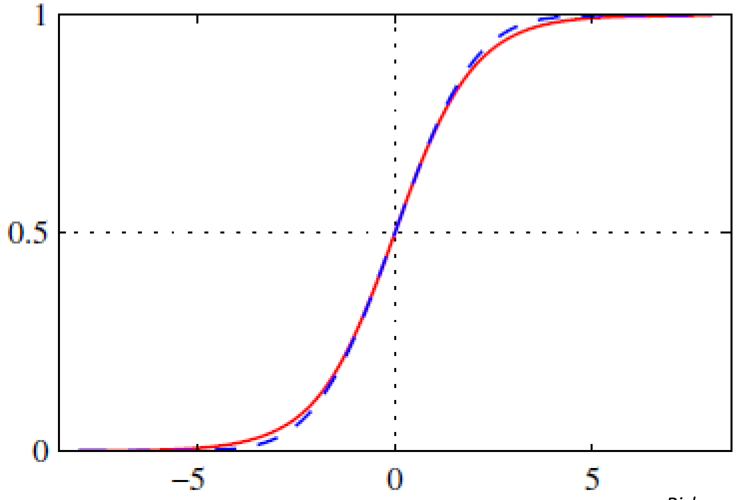
Probabilistic Generative Model (revisited)

- For two classes:

$$P(\omega_1|x) = \frac{p(x|\omega_1)P(\omega_1)}{p(x)} = \frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_1)P(\omega_1) + p(x|\omega_2)P(\omega_2)}$$

we can rewrite this equation as:

$$P(\omega_1|x) = \frac{1}{1+\exp(-a)} = \sigma(a)$$



Bishop

where $a = \ln \frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_2)P(\omega_2)}$ and $\sigma(a)$ is the *logistic sigmoid* function.

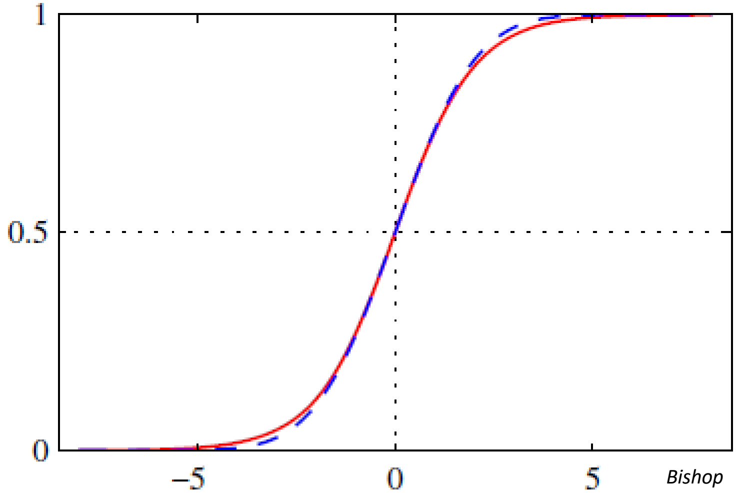
Probabilistic Generative Model (revisited)

- For number of classes $c > 2$:

$$P(\omega_i|x) = \frac{p(x|\omega_i)P(\omega_i)}{p(x)} = \frac{p(x|\omega_i)P(\omega_i)}{\sum_j p\left(x\,\middle|\omega_j\right)P(\omega_j)}$$

we can rewrite this equation as:

$$P(\omega_i|x) = \frac{\exp(a_i)}{\sum_j \exp(a_j)}$$



where $a_i = \ln\big(p(x|\omega_i)P(\omega_i)\big)$. This is known as the *normalized exponential* or the *softmax function.*

Probabilistic Generative Model (revisited)

- Recall that for, e.g., a two-class problem:

$$P(\omega_1|x) = \frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_1)P(\omega_1) + p(x|\omega_2)P(\omega_2)} = \sigma\left(\ln \frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_2)P(\omega_2)}\right)$$

Probabilistic Generative Model (revisited)

- Recall that for, e.g., a two-class problem:

$$P(\omega_1|x) = \frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_1)P(\omega_1) + p(x|\omega_2)P(\omega_2)} = \sigma\left(\ln\frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_2)P(\omega_2)}\right)$$

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$$

$$g(\boldsymbol{x}) = P(\omega_1\,|\boldsymbol{x}) - P(\omega_2\,|\boldsymbol{x})$$

$$g(\boldsymbol{x}) = \ln\frac{p(\boldsymbol{x}|\omega_1)}{p(\boldsymbol{x}|\omega_2)} + \ln\frac{P(\omega_1)}{P(\omega_2)}$$

slide 38 of Bayes Theory lecture

Probabilistic Generative Model (revisited)

- Recall that for, e.g., a two-class problem:

$$P(\omega_1|x) = \frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_1)P(\omega_1) + p(x|\omega_2)P(\omega_2)} = \sigma\left(\ln\frac{p(x|\omega_1)P(\omega_1)}{p(x|\omega_2)P(\omega_2)}\right)$$

and if we adopt Gaussian class conditional densities with same covariance matrix across classes we end up with:

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$$

$$g(\boldsymbol{x}) = P(\omega_1\,|\boldsymbol{x}) - P(\omega_2\,|\boldsymbol{x})$$

$$g(\boldsymbol{x}) = \ln\frac{p(\boldsymbol{x}|\omega_1)}{p(\boldsymbol{x}|\omega_2)} + \ln\frac{P(\omega_1)}{P(\omega_2)}$$

slide 38 of Bayes Theory lecture

$$P(\omega_1|x) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0)$$

where

$$\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \text{ and}$$

$$w_0 = -\frac{1}{2}\boldsymbol{\mu}_1\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + ln\frac{P(\omega_1)}{P(\omega_2)}$$

- We can use Maximum Likelihood to estimate $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$ and $\boldsymbol{\Sigma}$.

Logistic Regression

- Thus we have shown that the posterior class probability under certain conditions can be written as a logistic sigmoid acting on a linear function of the feature vector:

$$P(\omega_1|\boldsymbol{x}) = y(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x})$$

with $P(\omega_2|x) = 1 - P(\omega_1|x)$.

- Using basis functions we can rewrite this as:

$$P(\omega_1|\boldsymbol{\varphi}(\boldsymbol{x})) = y(\boldsymbol{\varphi}(\boldsymbol{x})) = \sigma(\boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x}))$$

with $P(\omega_2|\boldsymbol{\varphi}(\boldsymbol{x})) = 1 - P(\omega_1|\boldsymbol{\varphi}(\boldsymbol{x}))$.

- In statistics this model is known as *logistic regression*. However, this is a model for classification **not** regression.

Logistic Regression

$$P(\omega_1|\boldsymbol{\varphi}(\boldsymbol{x})) = y(\boldsymbol{\varphi}(\boldsymbol{x})) = \sigma(\boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x}))$$

with $P(\omega_2|\boldsymbol{\varphi}(\boldsymbol{x})) = 1 - P(\omega_1|\boldsymbol{\varphi}(\boldsymbol{x}))$.

- For a d −dimensional feature space $\boldsymbol{\varphi}$, this model has $d + 1$ adjustable parameters (weight of the linear function).

- If we have tried to fit gaussian class conditional densities we would need to estimate $2d$ parameters for the means and $\mathcal{O}(d^2)$ parameters for the shared covariance matrix!

- Thus, even if the logistic regression parameters scale linearly with d the probabilistic generative model approach scales quadratically with d!

- There is clear advantage to work with the logistic regression model directly!

Logistic Regression–Training(Maximum Likelihood)

$$P(\omega_1|\boldsymbol{\varphi}(\boldsymbol{x})) = y(\boldsymbol{\varphi}(\boldsymbol{x})) = \sigma(\boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x}))$$

- In order to adjust the weights of the logistic regression model we can try Maximum Likelihood.

- We will make use of the fact that:

$$\frac{d\sigma(a)}{da} = \sigma(a)(1 - \sigma(\alpha))$$

- Thus with a training set of N samples of the form $\{\boldsymbol{\varphi}(\boldsymbol{x}_n), t_n\}$ with $t_n \in \{0,1\}$ the likelihood function can be written as:

$$p(\boldsymbol{t}|\boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n},$$

where $y_n = P(\omega_1|\boldsymbol{\varphi}(\boldsymbol{x}_n)) = \sigma(\boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x}_n))$ and $\boldsymbol{t} = (t_1, \dots, t_N)$

Logistic Regression–Training(Maximum Likelihood)

- We can define an error function and instead of maximizing the likelihood:

$$p(\boldsymbol{t}|\boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n},$$

we can minimize the error function which is the negative logarithm of the likelihood:

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{t}|\boldsymbol{w}) = -\sum_{n=1}^{N}[t_n \ln y_n + (1 - t_n) \ln(1 - y_n)]$$

- This error function is called *cross entropy loss* and its gradient is:

$$\nabla E(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - t_n)\,\boldsymbol{\varphi}(\boldsymbol{x}_n)$$

Logistic Regression–Training(SGD)

$$\nabla E(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\varphi}(\boldsymbol{x}_n) = \sum_{n=1}^{N} \nabla E_n(\boldsymbol{w})$$

- Unfortunately, we cannot derive a closed form solution for $\boldsymbol{w}$ as $y(\boldsymbol{\varphi}(\boldsymbol{x})) = \sigma(\boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x}))$ is nonlinear on $\boldsymbol{w}$.

- Nevertheless, $E(\boldsymbol{w})$ is a convex function of $\boldsymbol{w}$ and we could use a sequential learning algorithm for the logistic regression model training.

- In particular with SGD we get:

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - \eta \nabla E_n(\boldsymbol{w}^{i}) \quad \rightarrow \quad \boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - \eta (y_n - t_n)\boldsymbol{\varphi}(\boldsymbol{x}_n)$$

- **Classification:** a test sample is assigned to ω_1 if $P(\omega_1|\boldsymbol{\varphi}(\boldsymbol{x})) = \sigma(\boldsymbol{w}^T\boldsymbol{\varphi}(\boldsymbol{x})) \geq \frac{1}{2}$

Questions?

Pattern Recognition & Machine Learning

Linear Discriminant Functions and Models