Linear Regression

Analytical solution, gradient descent, feature expansion

Machine Learning and Data Mining, 2025

Majid Sohrabi
National Research University Higher School of Economics

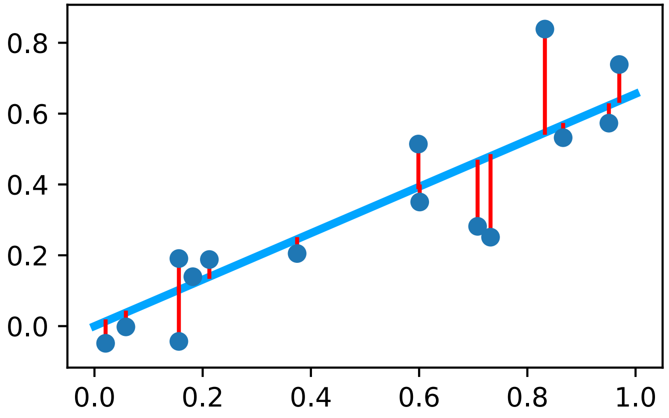
September 3, 2025

Why study linear models?

Linear models in a nutshell

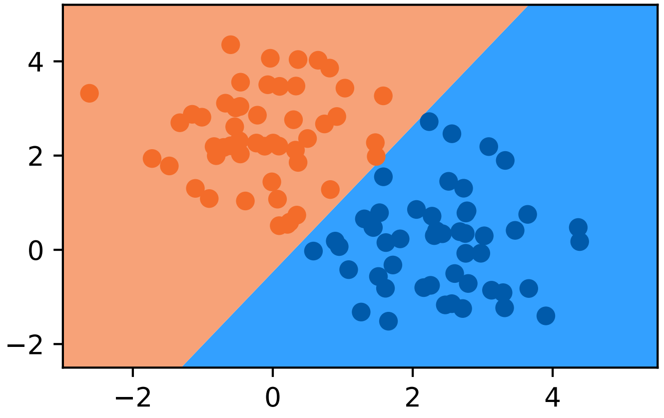
Regression:

$$\hat{f}(x) = \theta^{\mathrm{T}} x$$



Classification:

$$\hat{f}(x) = \mathbb{I}[\theta^{\mathrm{T}} x > 0]$$

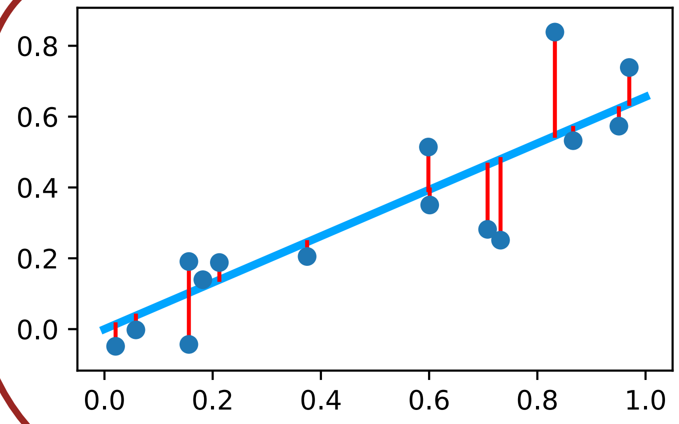


Outputs linear in inputs

Linear models in a nutshell

Regression:

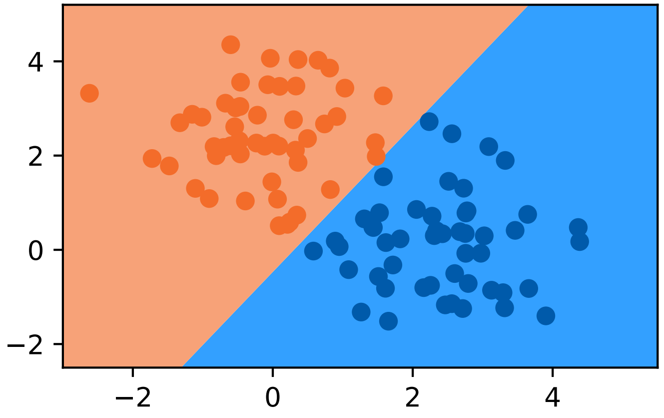
$$\hat{f}(x) = \theta^{\mathrm{T}} x$$



Targets $\in \mathbb{R}$ (or even $\mathbb{R}^m$ in the multidimensional case)

Classification:

$$\hat{f}(x) = \mathbb{I}[\theta^{\mathrm{T}} x > 0]$$

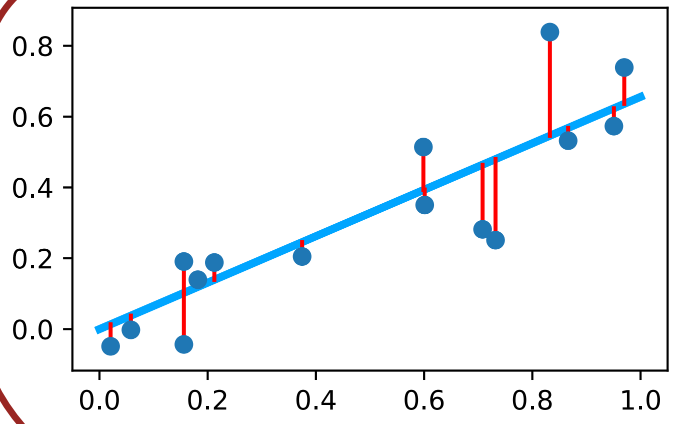


Outputs linear in inputs

Linear models in a nutshell

Regression:

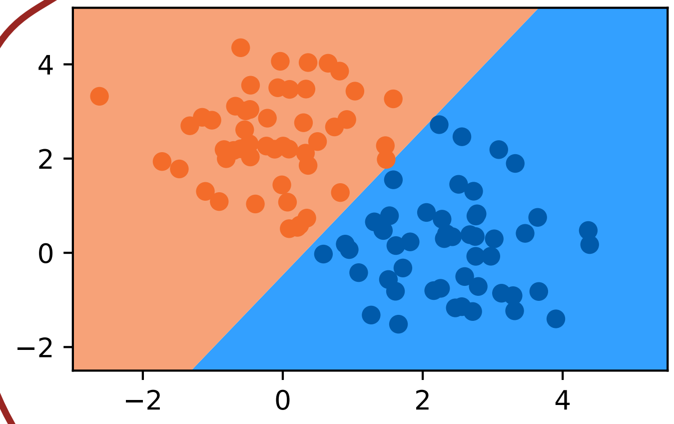
$$\hat{f}(x) = \theta^{\mathrm{T}} x$$



Targets $\in \mathbb{R}$ (or even $\mathbb{R}^m$ in the multidimensional case)

Classification:

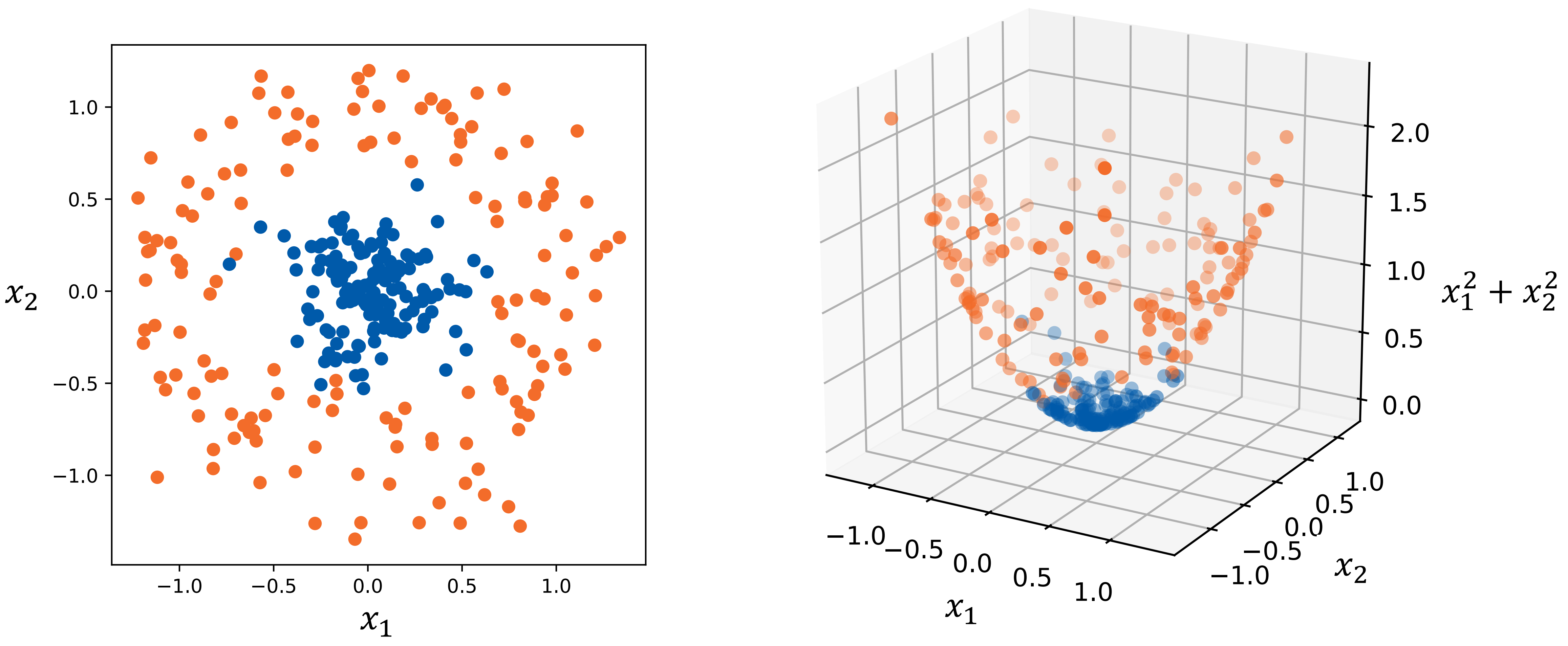
$$\hat{f}(x) = \mathbb{I}[\theta^{\mathrm{T}} x > 0]$$



Targets $\in$ some finite set

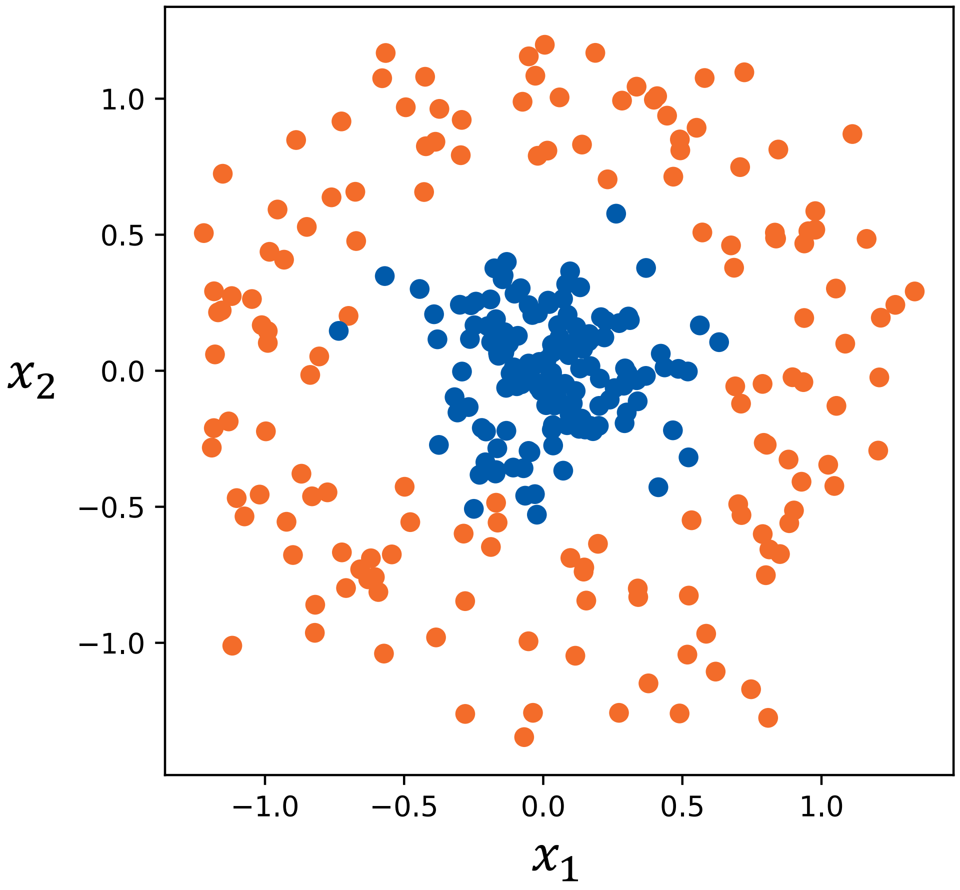
Outputs linear in inputs

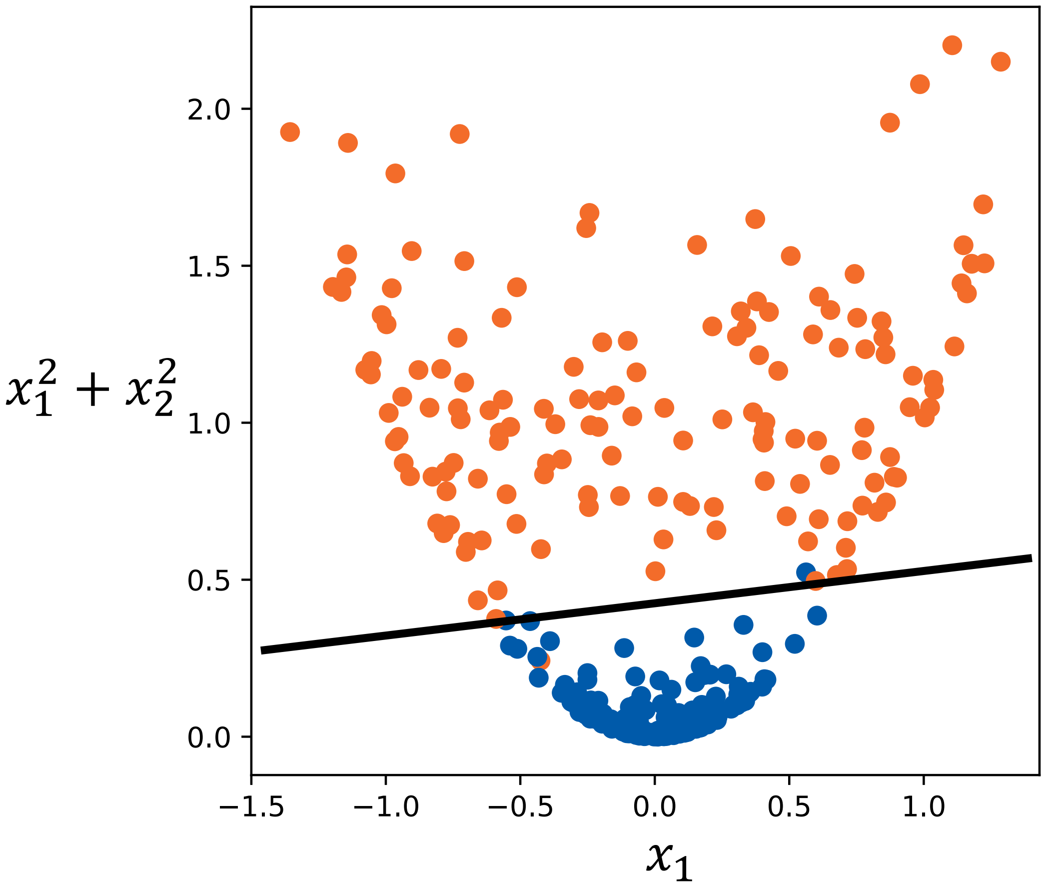
The hidden power



Linearly **inseparable** $\longrightarrow$ **separable** by **transforming the features**

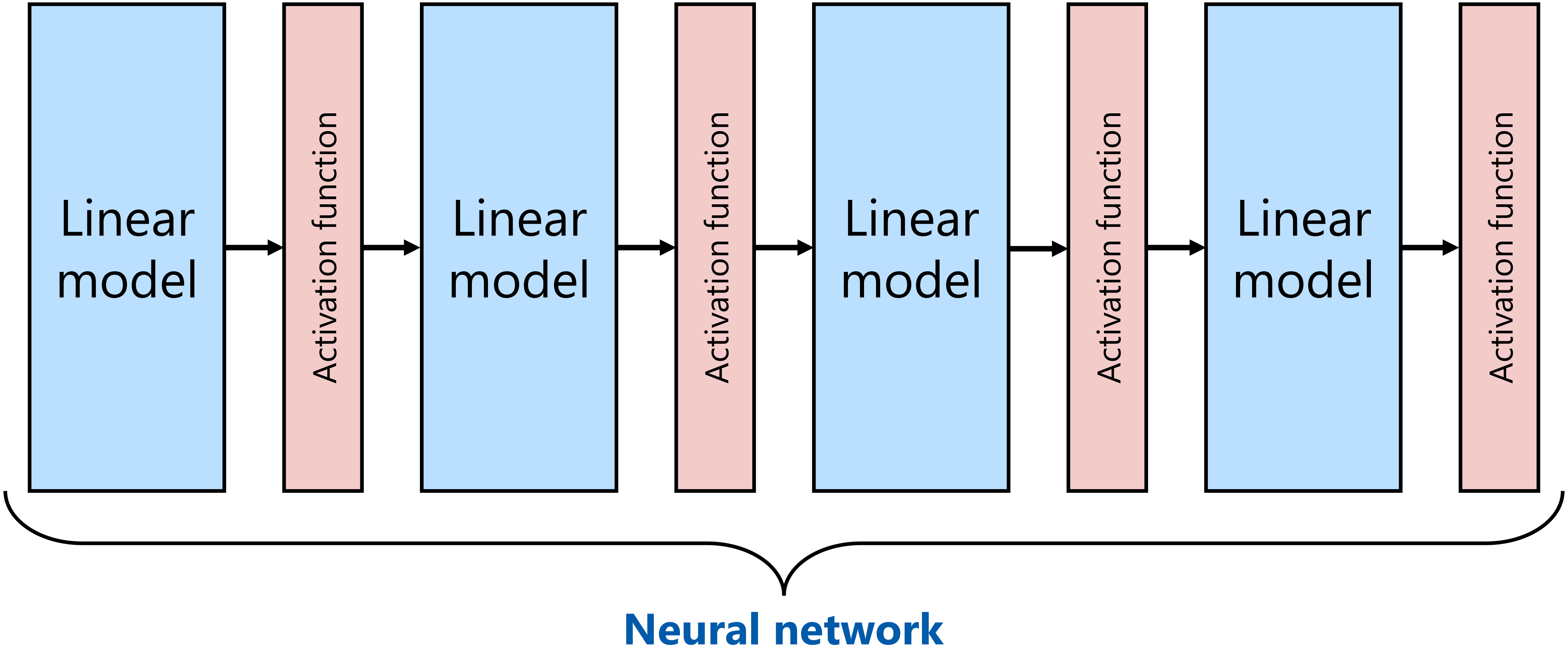
The hidden power





Linearly **inseparable** ⟶ **separable** by **transforming the features**

Building block for deep models



Better intuition for deep neural networks training

Interpretability

One can take a look at the weights corresponding to each of the features separately, e.g.:

Note: this example is not based on real data, but rather taken from the lecturer's imagination

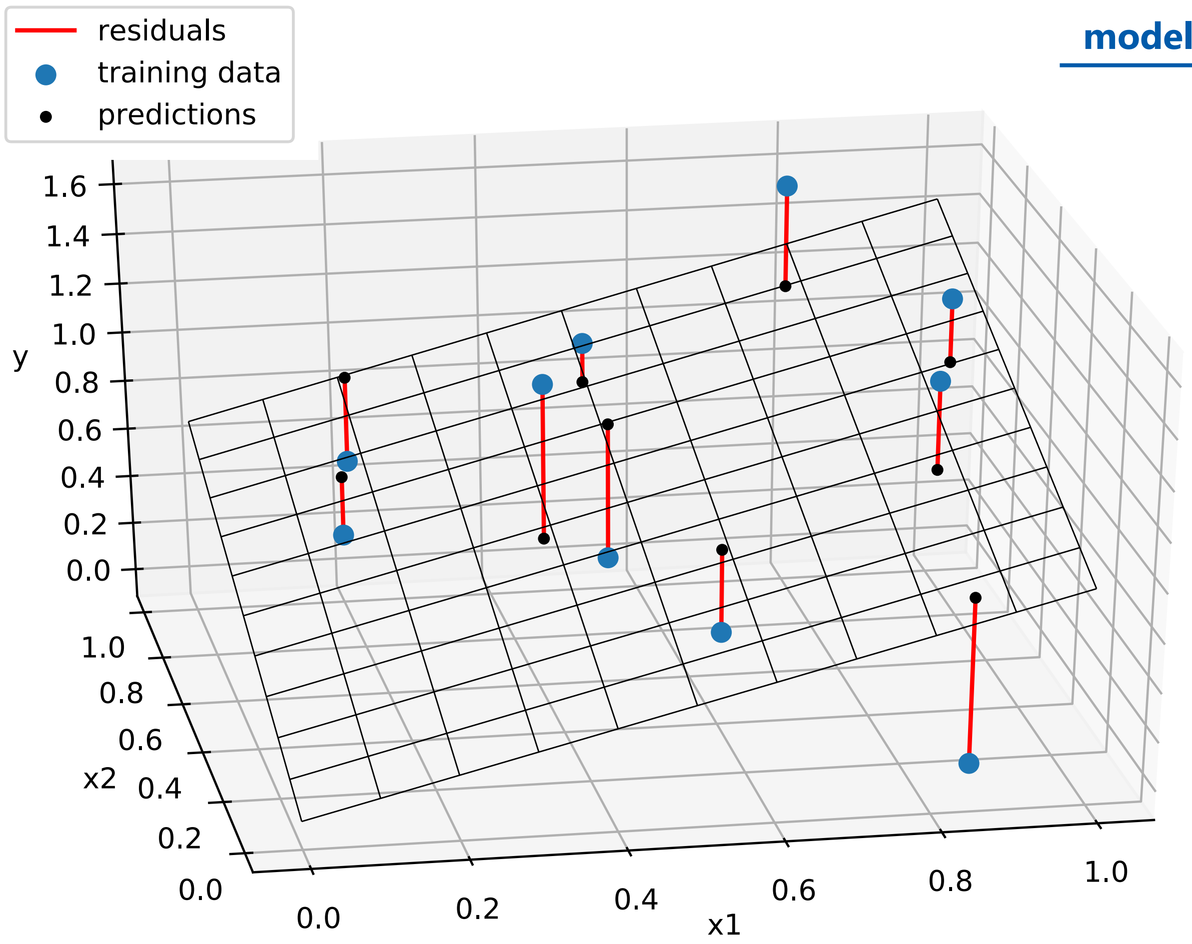
PredictedIncome = 1000 + 10 × **Age** + 1000 × **HasHigherEducation** – 0.1 × **DistanceFromCapital**

Higher weight means more impact on the prediction

- Note: features need to be of same scale if you want to compare their weights

Linear Regression

Linear Regression model



model prediction

$$\hat{f}_\theta(x) = \theta^{\mathrm{T}} x$$

parameters vector

$$\theta \in \mathbb{R}^d$$

features vector

$$x \in \mathcal{X} \subset \mathbb{R}^d$$

$$\underbrace{\frac{1}{N} \sum_{i=1\ldots N} \left(y_i - \hat{f}_\theta(x_i)\right)^2}_{\text{Mean Squared Error (MSE loss)}} \xrightarrow[\theta]{} \min$$

Mean Squared Error
(MSE loss)

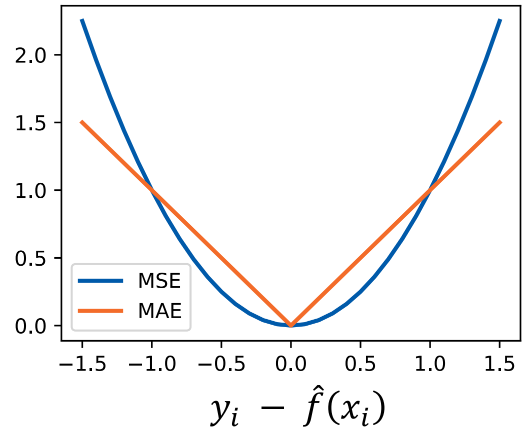
Common loss functions

Mean squared error (MSE): $$\frac{1}{N}\sum_{i=1\ldots N}\left(y_i - \hat{f}_\theta(x_i)\right)^2$$

Common loss functions

Mean squared error (MSE): $$\frac{1}{N}\sum_{i=1\dots N}\left(y_i - \hat{f}_\theta(x_i)\right)^2$$

Mean absolute error (MAE): $$\frac{1}{N}\sum_{i=1\dots N}\left|y_i - \hat{f}_\theta(x_i)\right|$$



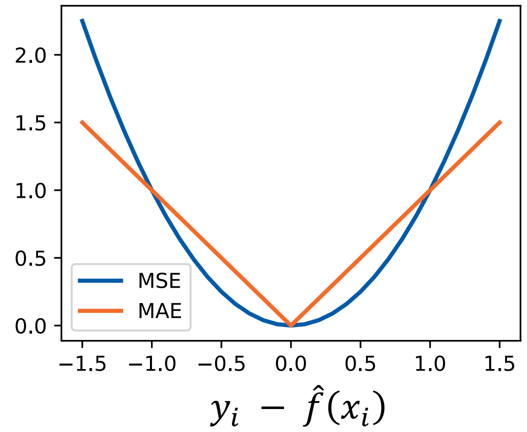
Common loss functions

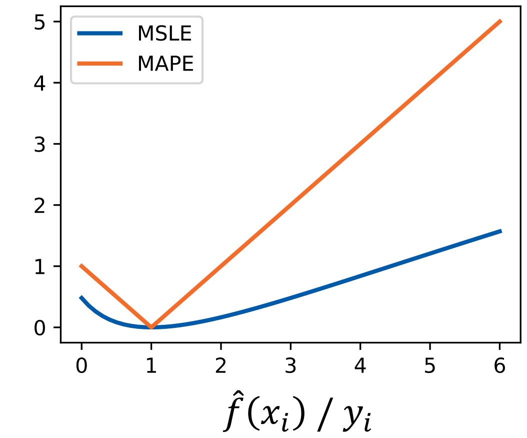
Mean squared error (MSE): $$\frac{1}{N}\sum_{i=1\dots N}\left(y_i - \hat{f}_\theta(x_i)\right)^2$$

Mean absolute error (MAE): $$\frac{1}{N}\sum_{i=1\dots N}\left|y_i - \hat{f}_\theta(x_i)\right|$$

Mean absolute percentage error (MAPE): $$\frac{1}{N}\sum_{i=1\dots N}\left|\frac{y_i - \hat{f}_\theta(x_i)}{y_i}\right|$$

Mean squared logarithmic error (MSLE): $$\frac{1}{N}\sum_{i=1\dots N}\left(\log(y_i+1) - \log\left(\hat{f}_\theta(x_i)+1\right)\right)^2$$





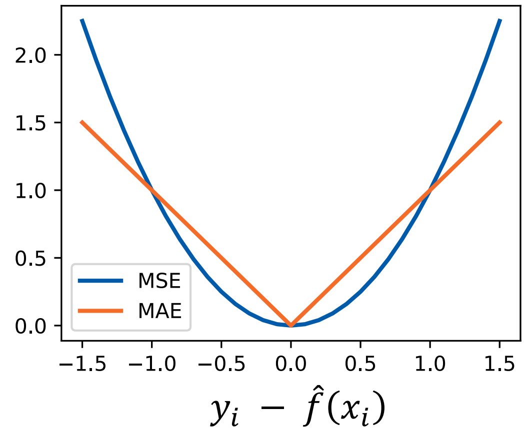
Common loss functions

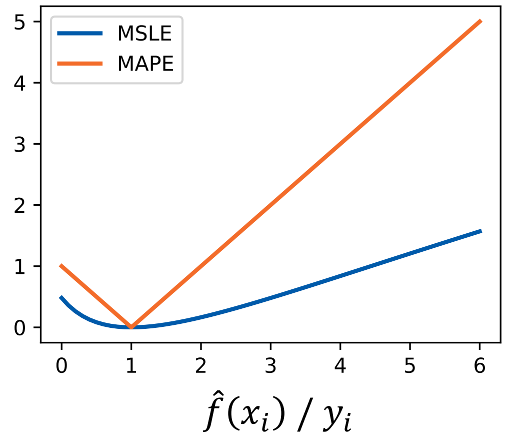
Mean squared error (MSE): $$\frac{1}{N}\sum_{i=1\ldots N}\left(y_i - \hat{f}_\theta(x_i)\right)^2$$

Mean absolute error (MAE): $$\frac{1}{N}\sum_{i=1\ldots N}\left|y_i - \hat{f}_\theta(x_i)\right|$$

Mean absolute percentage error (MAPE): $$\frac{1}{N}\sum_{i=1\ldots N}\left|\frac{y_i - \hat{f}_\theta(x_i)}{y_i}\right|$$

Mean squared logarithmic error (MSLE): $$\frac{1}{N}\sum_{i=1\ldots N}\left(\log(y_i + 1) - \log\left(\hat{f}_\theta(x_i) + 1\right)\right)^2$$





Different **loss functions** also are related to different **assumptions about the data**

Analytical solution

Recall the design matrix:

$$\begin{array}{c} \textit{features} \\ X = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^d \\ x_2^1 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^d \end{bmatrix} \textit{objects} \end{array}$$

Analytical solution

Recall the design matrix:

$$X = \begin{array}{cl} \xrightarrow{\textit{features}} & \\ \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^d \\ x_2^1 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^d \end{bmatrix} & \Big\downarrow \textit{objects} \end{array}$$

We can use it to rewrite the MSE loss:

$$\mathcal{L}_{\mathrm{MSE}} = \frac{1}{N} \sum_{i=1\dots N} \left(y_i - \theta^{\mathrm{T}} x_i\right)^2 = \frac{1}{N} \|y - X\theta\|^2$$

$$y = (y_1, y_2, \dots, y_N)^{\mathrm{T}} \text{ – vector of targets}$$

Analytical solution

$$\mathcal{L}_{\text{MSE}} \sim \|y - X\theta\|^2 \to \min_{\theta}$$

$$\begin{cases} \dfrac{\partial}{\partial\theta}\mathcal{L}_{\text{MSE}} = 0 \\ \dfrac{\partial^2}{\partial\theta\partial\theta^{\text{T}}}\mathcal{L}_{\text{MSE}} > 0\ (\text{pos. def.}) \end{cases}$$

Analytical solution

Working on the 1st derivative*:

$$\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{MSE}} \sim \frac{\partial}{\partial\theta}(y - X\theta)^{\mathrm{T}}(y - X\theta)$$

*some useful info about matrix calculus: https://en.wikipedia.org/wiki/Matrix_calculus#Identities

Analytical solution

Working on the 1st derivative*:

$$\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{MSE}} \sim \frac{\partial}{\partial\theta}(y - X\theta)^{\mathrm{T}}(y - X\theta) = -2X^{\mathrm{T}}(y - X\theta) = 0$$

*some useful info about matrix calculus: https://en.wikipedia.org/wiki/Matrix_calculus#Identities

Analytical solution

Working on the 1st derivative*:

$$\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{MSE}} \sim \frac{\partial}{\partial\theta}(y - X\theta)^{\mathrm{T}}(y - X\theta) = -2X^{\mathrm{T}}(y - X\theta) = 0$$

$$X^{\mathrm{T}}y - X^{\mathrm{T}}X\theta = 0$$

*some useful info about matrix calculus: https://en.wikipedia.org/wiki/Matrix_calculus#Identities

Analytical solution

Working on the 1st derivative*:

$$\frac{\partial}{\partial \theta}\mathcal{L}_{\mathrm{MSE}} \sim \frac{\partial}{\partial \theta}(y - X\theta)^{\mathrm{T}}(y - X\theta) = -2X^{\mathrm{T}}(y - X\theta) = 0$$

$$X^{\mathrm{T}}y - X^{\mathrm{T}}X\theta = 0$$

Solution:

$$\theta = \left(X^{\mathrm{T}}X\right)^{-1}X^{\mathrm{T}}y$$

*some useful info about matrix calculus: https://en.wikipedia.org/wiki/Matrix_calculus#Identities

Analytical solution

Working on the 1st derivative*:

$$\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{MSE}} \sim \frac{\partial}{\partial\theta}(y - X\theta)^{\mathrm{T}}(y - X\theta) = -2X^{\mathrm{T}}(y - X\theta) = 0$$

$$X^{\mathrm{T}}y - X^{\mathrm{T}}X\theta = 0$$

Solution:

$$\theta = \left(X^{\mathrm{T}}X\right)^{-1}X^{\mathrm{T}}y$$

Note that this matrix needs to be invertible

*some useful info about matrix calculus: https://en.wikipedia.org/wiki/Matrix_calculus#Identities

Analytical solution

2nd derivative:

$$\frac{\partial^2}{\partial\theta\partial\theta^{\mathrm{T}}}\mathcal{L}_{\mathrm{MSE}} \sim 2X^{\mathrm{T}}X$$

Analytical solution

2nd derivative:

$$\frac{\partial^2}{\partial\theta\partial\theta^{\mathrm{T}}}\mathcal{L}_{\mathrm{MSE}} \sim 2X^{\mathrm{T}}X$$

This needs to be **positive definite**

Analytical solution

For some non-zero vector v:

$$v^{\mathrm{T}}X^{\mathrm{T}}Xv = (Xv)^{\mathrm{T}}(Xv) = \|Xv\|^2 \geq 0$$

2nd derivative:

$$\frac{\partial^2}{\partial\theta\partial\theta^{\mathrm{T}}}\mathcal{L}_{\mathrm{MSE}} \sim 2X^{\mathrm{T}}X$$

This needs to be **positive definite**

Analytical solution

For some non-zero vector v:

$$v^{\mathrm{T}} X^{\mathrm{T}} X v = (Xv)^{\mathrm{T}}(Xv) = \|Xv\|^2 \geq 0$$

$$\neq 0$$

when columns of X are linearly independent

2nd derivative:

$$\frac{\partial^2}{\partial\theta\partial\theta^{\mathrm{T}}}\mathcal{L}_{\mathrm{MSE}} \sim 2X^{\mathrm{T}}X$$

This needs to be **positive definite**

Analytical solution

For some non-zero vector v:

$$v^{\mathrm{T}} X^{\mathrm{T}} X v = (Xv)^{\mathrm{T}}(Xv) = \|Xv\|^2 \geq 0$$

$$\neq 0$$

when columns of X are linearly independent

2nd derivative:

$$\frac{\partial^2}{\partial\theta\partial\theta^{\mathrm{T}}} \mathcal{L}_{\mathrm{MSE}} \sim 2X^{\mathrm{T}}X$$

This needs to be **positive definite**

True when all the features (columns of the design matrix) are **linearly**

Analytical solution

For some non-zero vector v:
$v^{\mathrm{T}}X^{\mathrm{T}}Xv = (Xv)^{\mathrm{T}}(Xv) = \|Xv\|^2 \geq 0$
$\neq 0$
when columns of X are linearly independent

2nd derivative:

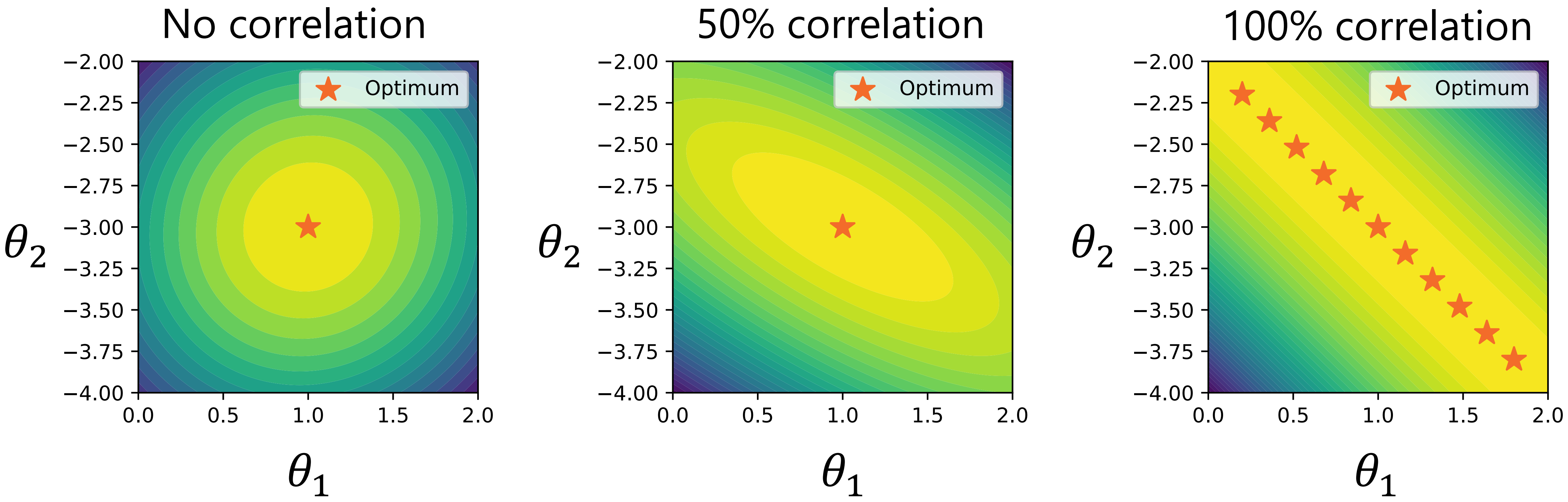
$$\frac{\partial^2}{\partial\theta\partial\theta^{\mathrm{T}}}\mathcal{L}_{\mathrm{MSE}} \sim 2X^{\mathrm{T}}X$$

This needs to be **positive definite**

True when all the features (columns of the design matrix) are **linearly independent**

This also makes $X^T X$ invertible

Feature correlations matter!



MSE level maps

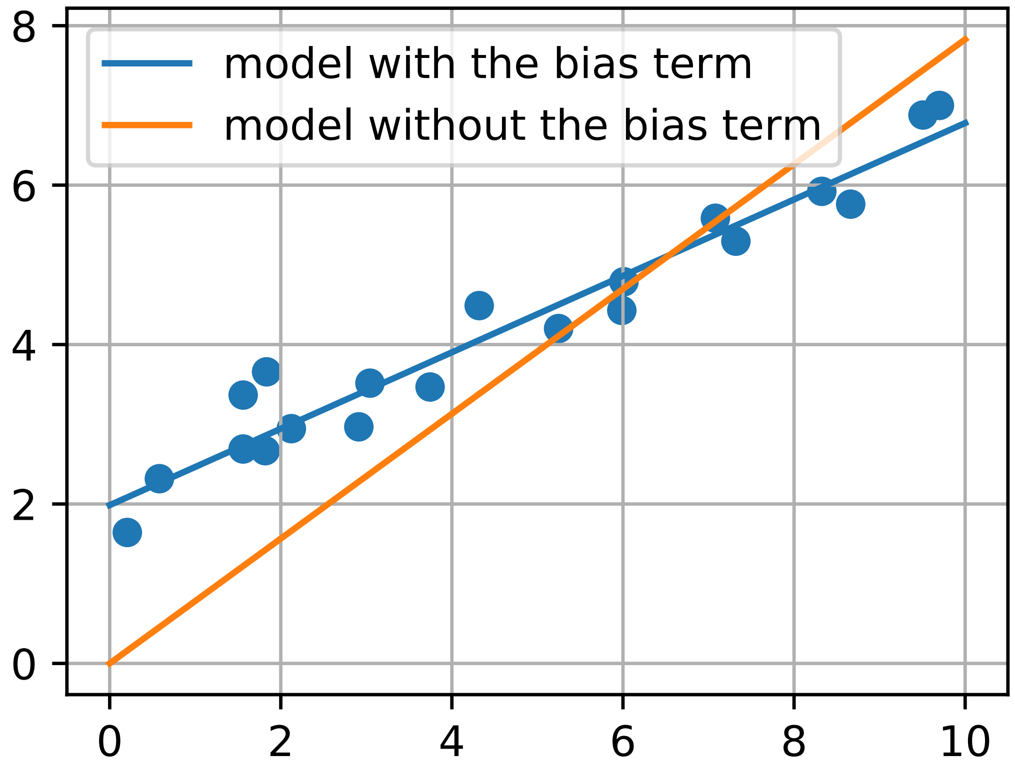
Majid Sohrabi, NRU HSE

Bias term

a.k.a. intercept term

$$\hat{f}_\theta(x) = \theta^{\mathrm{T}} x + \theta_0$$

$$\theta \in \mathbb{R}^d$$
$$\theta_0 \in \mathbb{R}$$
$$x \in \mathcal{X} \subset \mathbb{R}^d$$



No need to redo the math – just add a **constant feature** to the design matrix:

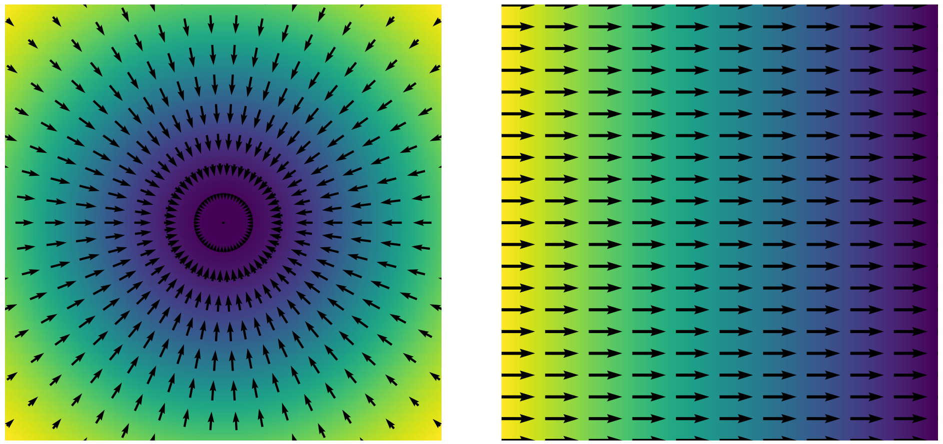
$$X = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^d \\ x_2^1 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^d \end{bmatrix} \longrightarrow X = \begin{bmatrix} 1 & x_1^1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2^1 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N^1 & x_N^2 & \cdots & x_N^d \end{bmatrix}$$

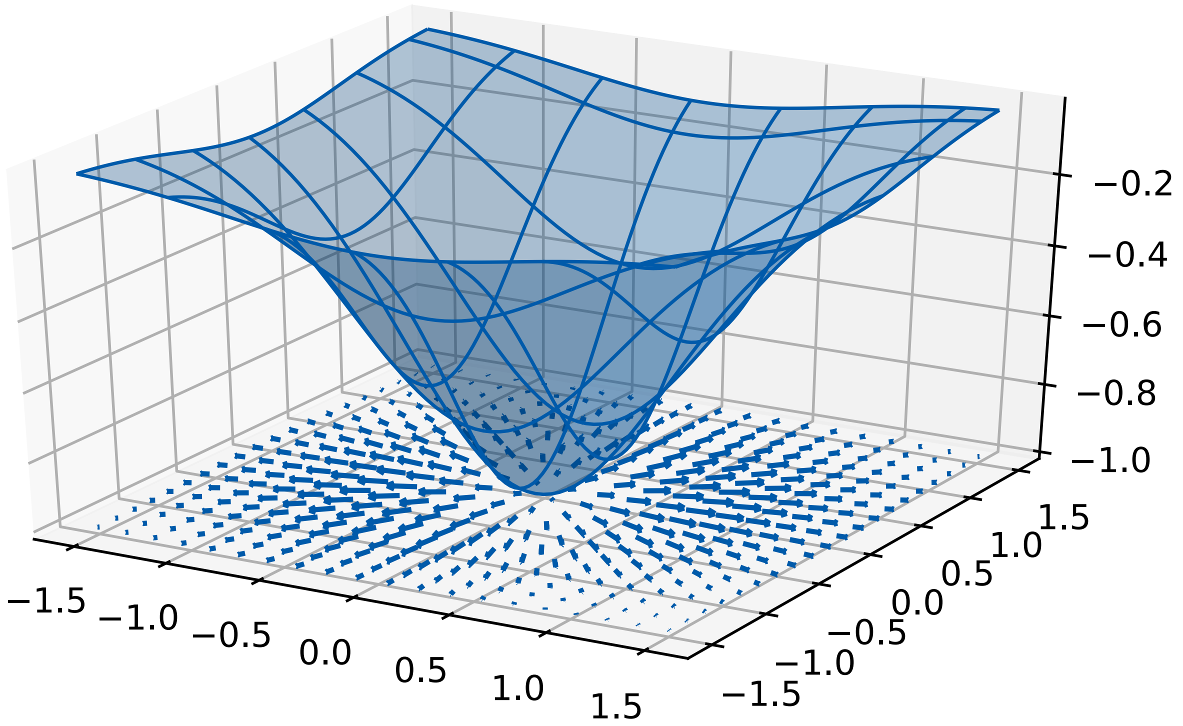
Numerical & Stochastic Optimization

Gradient

Gradient: $\nabla_x f(x) \equiv \left(\frac{\partial f(x)}{\partial x_1}, \dots, \frac{\partial f(x)}{\partial x_d}\right)$

Points towards **steepest function increase**





Gradient Descent Optimization

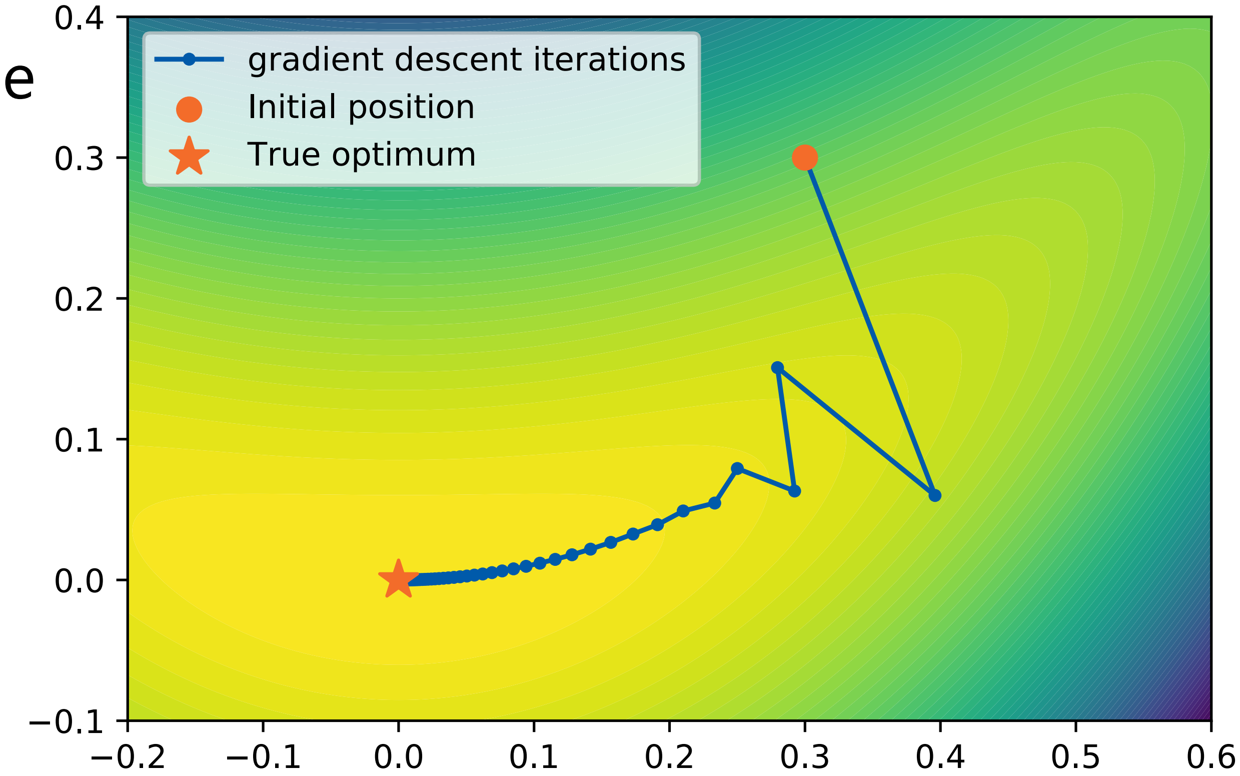
Can optimize functions starting at some initial point $x^{(0)}$ and moving opposite to the gradient:

$$x^{(k)} \longleftarrow x^{(k-1)} - \alpha \nabla_x f\left(x^{(k-1)}\right)$$

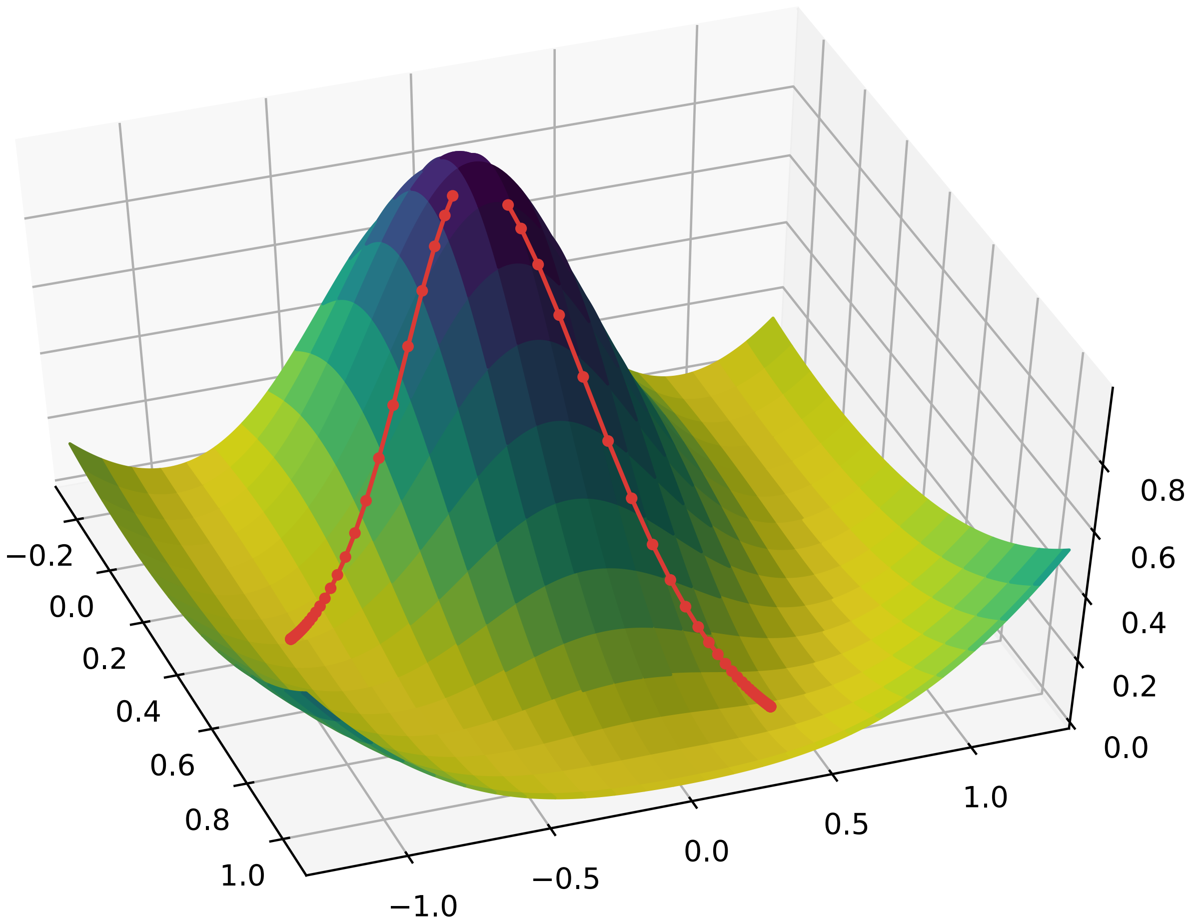
with $\alpha \in \mathbb{R},\ \alpha > 0$ – learning rate.

For smooth **convex** functions with a single minimum x^*:

$$f\left(x^{(k)}\right) - f(x^*) = \mathcal{O}\left(\frac{1}{k}\right)$$



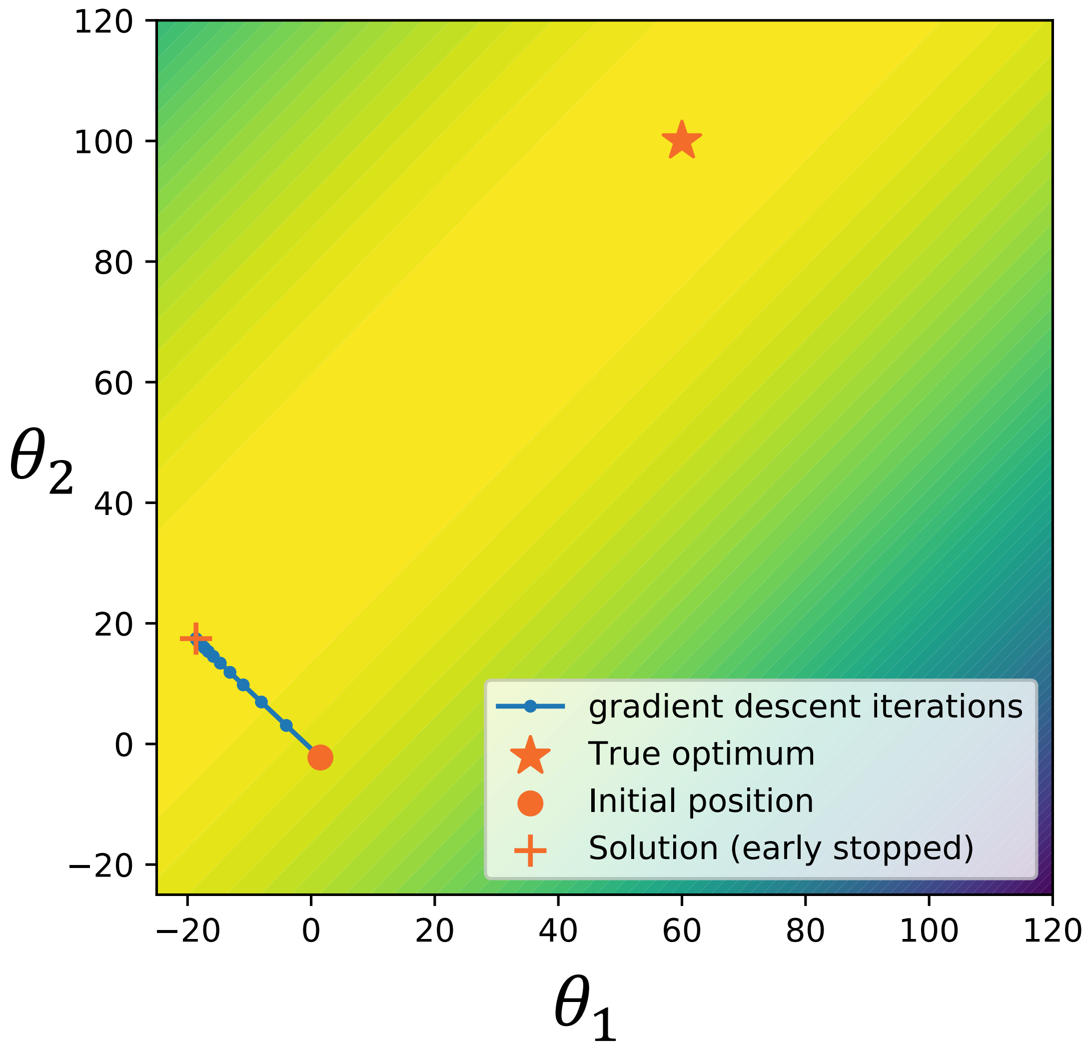
Gradient descent for non-convex functions



May get to a minimum which is not global

Result depends on the starting point

Gradient descent as means for regularisation



Large parameter values typically mean overfitting

You may avoid this problem by initializing parameters with **small values** and **early stopping** the gradient descent

Stochastic Gradient Descent (SGD)

In machine learning we optimize loss functions which are typically averages over objects:

$$L = \frac{1}{N} \sum_{i = 1 \ldots N} \mathcal{L}\left(y_i, \widehat{f_\theta}(x_i)\right)$$

Stochastic Gradient Descent (SGD)

In machine learning we optimize loss functions which are typically averages over objects:

$$L = \frac{1}{N} \sum_{i = 1 \dots N} \mathcal{L}\left(y_i, \widehat{f_\theta}(x_i)\right)$$

For large N, gradient descent is computationally inefficient and may be unfeasible in terms of memory consumption

Stochastic Gradient Descent (SGD)

In machine learning we optimize loss functions which are typically averages over objects:

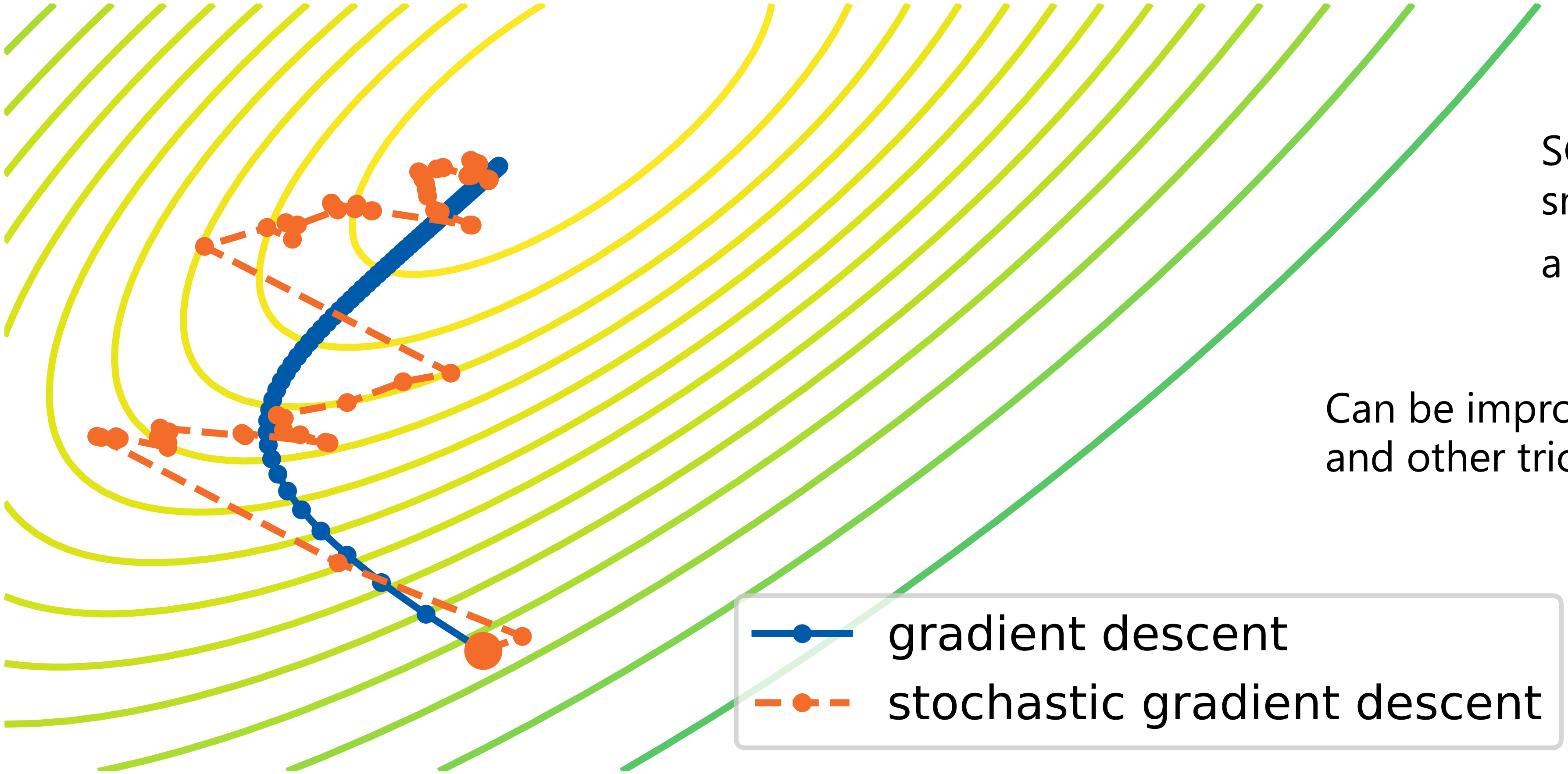
$$L = \frac{1}{N} \sum_{i = 1 \ldots N} \mathcal{L}\left(y_i, \widehat{f_\theta}(x_i)\right)$$

For large N, gradient descent is computationally inefficient and may be unfeasible in terms of memory consumption

Alternative:

- At each step k pick $l_k \in \{1, \ldots, N\}$ at random
- Optimize: $\theta^{(k)} \longleftarrow \theta^{(k-1)} - \alpha \nabla_\theta \mathcal{L}\left(y_{l_k}, \widehat{f_\theta}(x_{l_k})\right)\Big|_{\theta = \theta^{(k-1)}}$

Stochastic Gradient Descent (SGD)



SGD convergence rate for smooth **convex** functions with a single minimum: $\mathcal{O}\left(\frac{1}{\sqrt{k}}\right)$

Can be improved by batching and other tricks

Feature Expansion

Feature expansion

One can perform **feature transformations** with any function $\Phi: \mathbb{R}^d \to \mathbb{R}^{d'}$

$$X = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^d \\ x_2^1 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^d \end{bmatrix} \longrightarrow \Phi(X) = \begin{bmatrix} \Phi^1(x_1^1, \dots, x_1^d) & \cdots & \Phi^{d'}(x_1^1, \dots, x_1^d) \\ \Phi^1(x_2^1, \dots, x_2^d) & \cdots & \Phi^{d'}(x_2^1, \dots, x_2^d) \\ \vdots & \ddots & \vdots \\ \Phi^1(x_N^1, \dots, x_N^d) & \cdots & \Phi^{d'}(x_N^1, \dots, x_N^d) \end{bmatrix}$$

Feature expansion

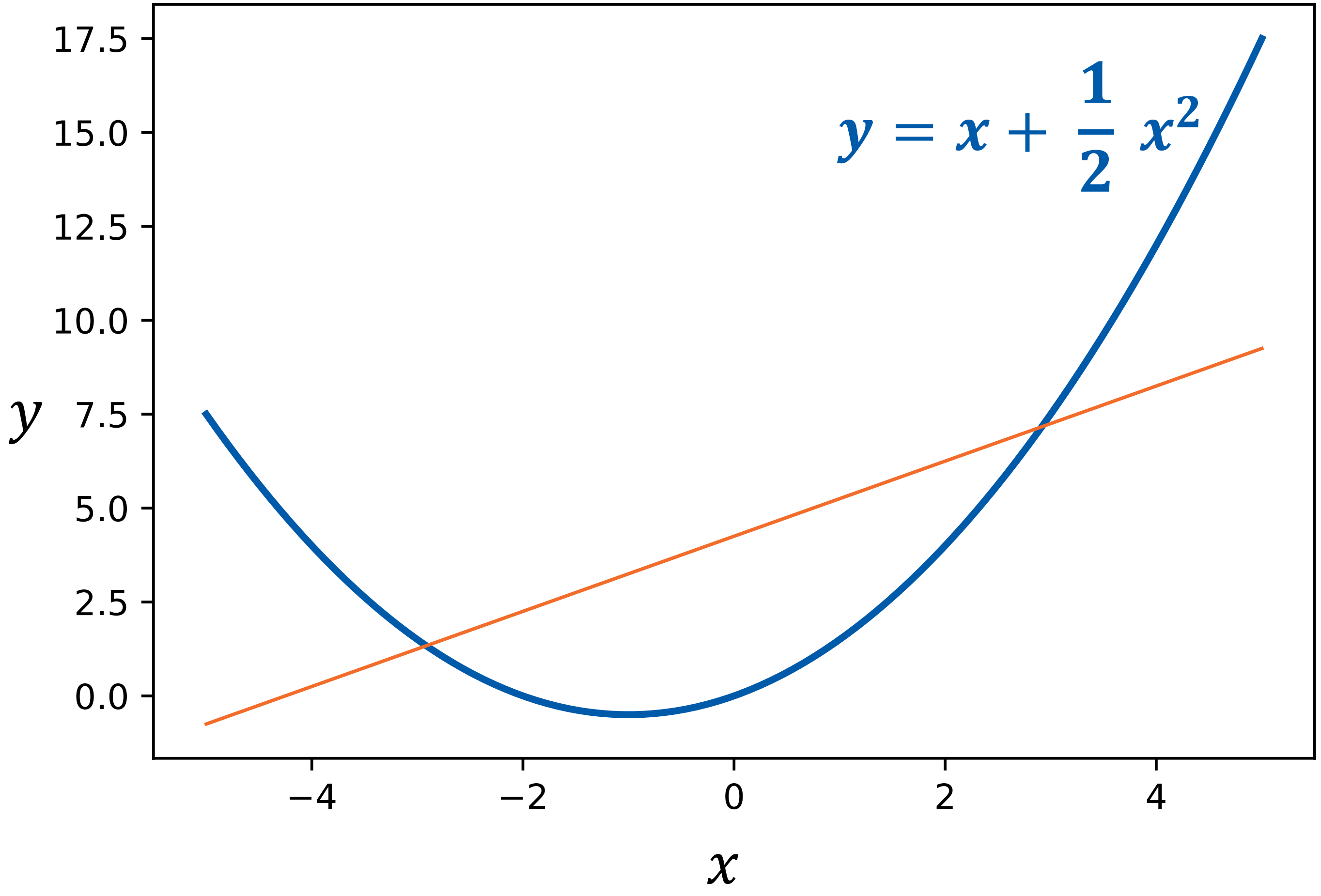
One can perform **feature transformations** with any function $\Phi\colon \mathbb{R}^d \to \mathbb{R}^{d'}$

$$X = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^d \\ x_2^1 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \ddots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^d \end{bmatrix} \longrightarrow \Phi(X) = \begin{bmatrix} \Phi^1(x_1^1, \dots, x_1^d) & \cdots & \Phi^{d'}(x_1^1, \dots, x_1^d) \\ \Phi^1(x_2^1, \dots, x_2^d) & \cdots & \Phi^{d'}(x_2^1, \dots, x_2^d) \\ \vdots & \ddots & \vdots \\ \Phi^1(x_N^1, \dots, x_N^d) & \cdots & \Phi^{d'}(x_N^1, \dots, x_N^d) \end{bmatrix}$$

Finding the best function Φ is called **feature engineering**

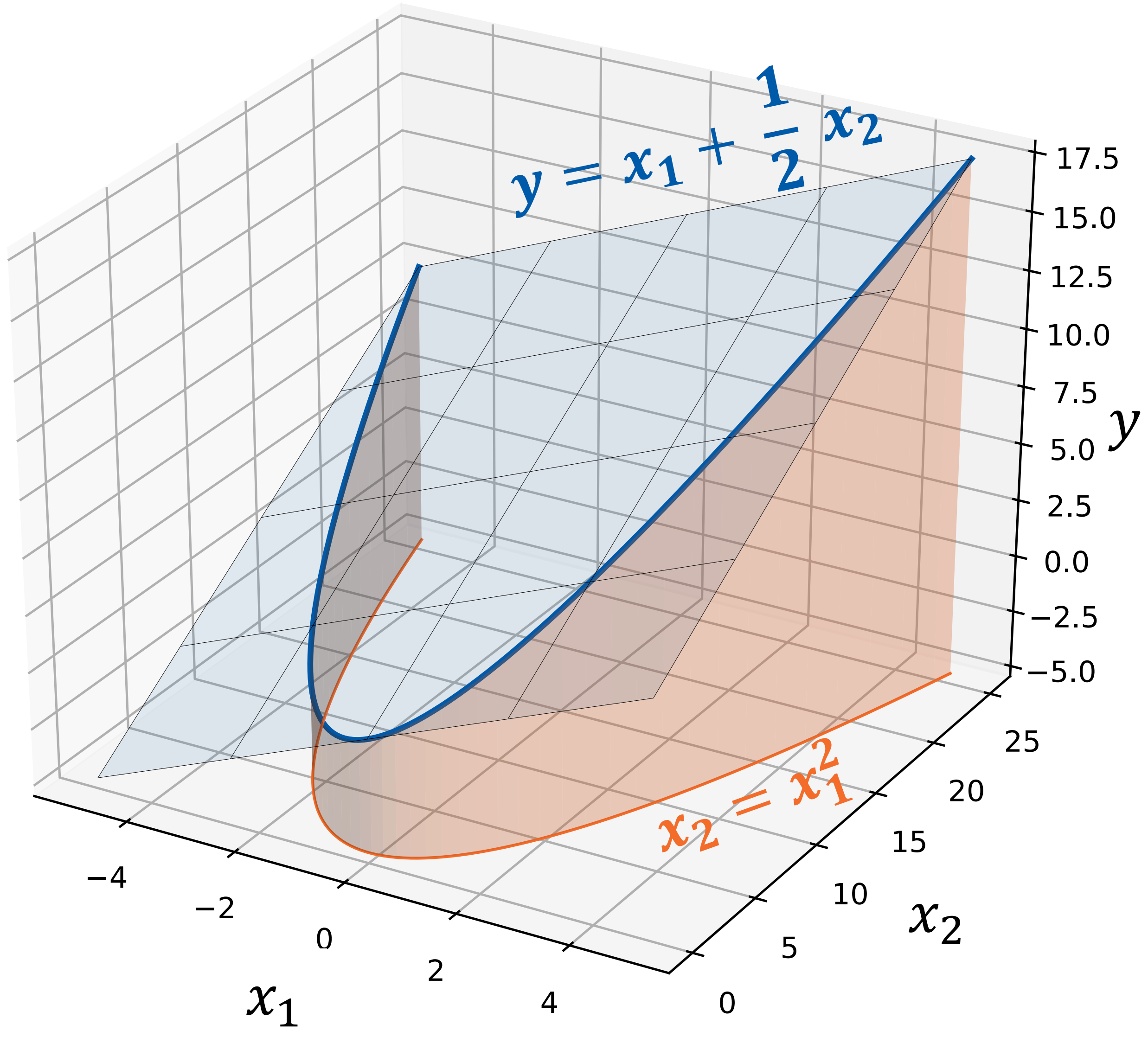
- It is an important part of machine learning and requires deep understanding of the underlying problem and the data

Example: polynomial features



Can't be solved with the only linear feature (x)

Example: polynomial features



Introducing another feature does the job:

$$(x_1, x_2) \equiv (x, x^2)$$

Now our estimate is:

$$\hat{f}(x) = \theta_1 x + \theta_2 x^2$$

Polynomial features of degree p (general case)

For the original features:

$$\left(x_i^1, x_i^2, \ldots, x_i^d \right)$$

introduce all unique multiplicative combinations of the form:

$$\left(x_i^{k_1}\right)^{p_1} \cdot \left(x_i^{k_2}\right)^{p_2} \cdot \ldots \cdot \left(x_i^{k_m}\right)^{p_m}$$

with $p_1 + p_2 + \ldots + p_m \leq p$

Example: degree 3 polynomial features

For the original features (a, b, c):

$$(1, a, b, c, a^2, ab, ac, b^2, bc, c^2, a^3, a^2b, a^2c, ab^2, abc, ac^2, b^3, b^2c, bc^2, c^3)$$

Summary

Understanding linear models gives **useful insights** into more complicated machine learning algorithms and optimization

Summary

Understanding linear models gives **useful insights** into more complicated machine learning algorithms and optimization

Linear Regression with MSE loss allows for **analytical solution**

Summary

Understanding linear models gives **useful insights** into more complicated machine learning algorithms and optimization

Linear Regression with MSE loss allows for **analytical solution**

The stability of the solution depends on the **feature correlations**

Summary

- Understanding linear models gives **useful insights** into more complicated machine learning algorithms and optimization
- Linear Regression with MSE loss allows for **analytical solution**
- The stability of the solution depends on the **feature correlations**
- Linear models can be optimized with gradient descent and stochastic gradient descent

Summary

Understanding linear models gives **useful insights** into more complicated machine learning algorithms and optimization

Linear Regression with MSE loss allows for **analytical solution**

The stability of the solution depends on the **feature correlations**

Linear models can be optimized with gradient descent and stochastic gradient descent

- In some cases this can **regularize** the solution

Feature transformations allow for very powerful use of the linear models

Summary

Understanding linear models gives **useful insights** into more complicated machine learning algorithms and optimization

Linear Regression with MSE loss allows for **analytical solution**

The stability of the solution depends on the **feature correlations**

Linear models can be optimized with gradient descent and stochastic gradient descent

- In some cases this can **regularize** the solution

Feature transformations allow for very powerful use of the linear models

Food for thought: how does polynomial feature expansion affect the complexity of the model?

Thank you!

msohrabi@hse.ru

Majid Sohrabi