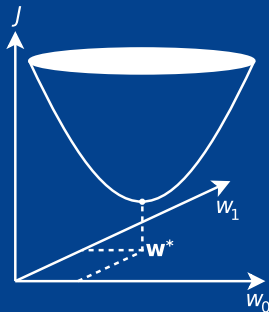




## LESSON 5: Regression og SGD (and Training)

CARSTEN EIE FRIGAARD

AUTUMN 2022



# L05: Regression og SGD (and Training)

## Agenda

- ▶ Training a linear regression model,
  - ▶ (and intro to GD)
- ▶ Cost function in closed-form vs. numerical solutions.
  - ▶ Opgave: [L05/linear\\_regression\\_1.ipynb](#)
  - ▶ Opgave: [L05/linear\\_regression\\_2.ipynb](#) [OPTIONAL]
- ▶ Gradient Descent (GD),
  - ▶ Learning rates,
  - ▶ Batch Gradient Descent (GD),
  - ▶ Stochastic Gradient Descent (SGD),
  - ▶ Mini-batch Gradient Descent.
  - ▶ Opgave: [L05/gradient\\_descent.ipynb](#)

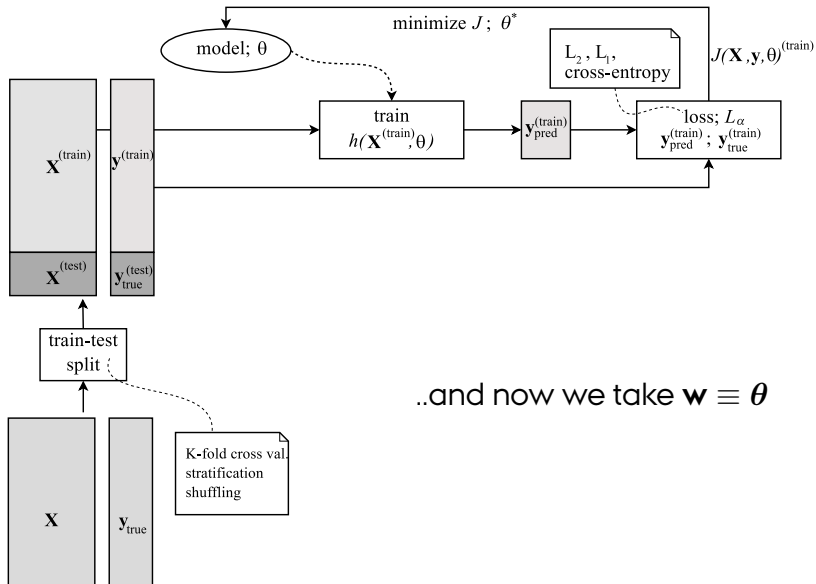
# TRAINING A LINEAR REGRESSOR

---



# Training in General

Training is minimization of  $J$  (optimization)



# Training a Linear Regressor

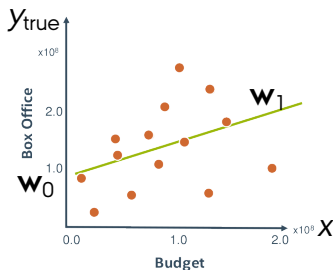
## Linear Regression: In one dimension

The well know linear equation

$$y(x) = \alpha x + \beta$$

or changing some of the symbol names, so that  $h(\mathbf{x}; \mathbf{w})$  means the **predicted** value from  $\mathbf{x}$  for a parameter set  $\mathbf{w}$ , via the hypothesis function

$$h(x; \mathbf{w}) \stackrel{1D}{=} w_0 + w_1 x$$



Question: how do we find the  $\mathbf{w}_n$ 's?

# Training a Linear Regressor

## Linear Regression: Hypothesis Function in $N$ -dimensions

For 1-D:

$$h(x^{(i)}; w) = w_0 + w_1 x^{(i)}$$

The same for  $N$ -D:

$$\begin{aligned} h(\mathbf{x}^{(i)}; \mathbf{w}) &= \mathbf{w}^\top \begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix} \\ &= w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_d x_d^{(i)} \end{aligned}$$

and to ease notation we always prepend  $\mathbf{x}$  with 1:

$$\begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix} \mapsto \mathbf{x}^{(i)}, \quad \text{by convention in the following...}$$

yielding the vector form of the hypothesis function

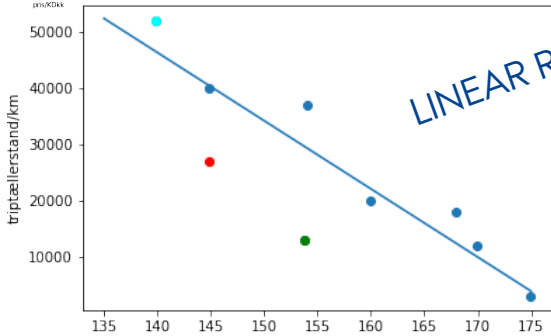
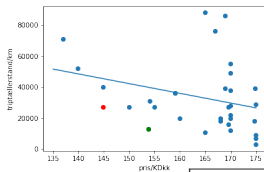
$$h(\mathbf{x}^{(i)}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}^{(i)}$$

# Training a Linear Regressor

Case study: a new car..

Scrape [bilbasen.dk], for particular e-Golf.

Features: total tripdistance/km and price/KDkk..



# Training a Linear Regressor

## Linear Regression: Loss or Objective Function

Individual loss, via a square difference ( $L = \mathcal{L}_2^2$ )

$$\begin{aligned} L^{(i)} &= ||y_{\text{pred}}^{(i)} - y^{(i)}||_2^2 \\ &= (h(\mathbf{x}^{(i)}; \mathbf{w}) - y^{(i)})^2 \\ &= (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2 \end{aligned} \quad y \equiv y_{\text{true}} \text{ in the following}$$

and to minimize all the  $L^{(i)}$  losses (or indirectly also the MSE or RMSE) is to minimize the sum of all the individual costs, via the total cost function  $J$

$$\begin{aligned} \text{MSE}(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= \frac{1}{n} \sum_{i=1}^n L^{(i)} \\ &= \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2 \\ &= \frac{1}{n} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \end{aligned}$$

Ignoring constant factors, this yields our linear regression cost function

$$J = \frac{1}{2} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \propto \text{MSE}$$



# Training a Linear Regressor

## Minimizing the Linear Regression: The `argmin` concept

Our linear regression cost function was

$$J(\mathbf{X}, \mathbf{y}; \mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

and training amounts to finding a value of  $\mathbf{w}$ , that minimizes  $J$ . This is denoted as

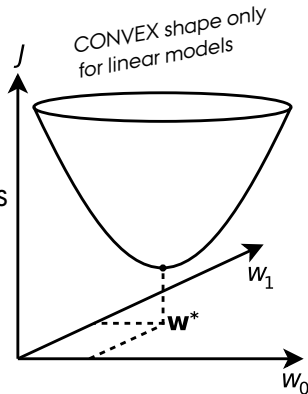
$$\begin{aligned}\mathbf{w}^* &= \operatorname{argmin}_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) \\ &= \operatorname{argmin}_{\mathbf{w}} \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2\end{aligned}$$

and by minima, we naturally hope for

- ▶ the global minimum

thought for non-linear models this cannot be guaranteed, hitting some

- ▶ local minimum



# COST FUNCTION MINIMIZATION IN CLOSED-FORM

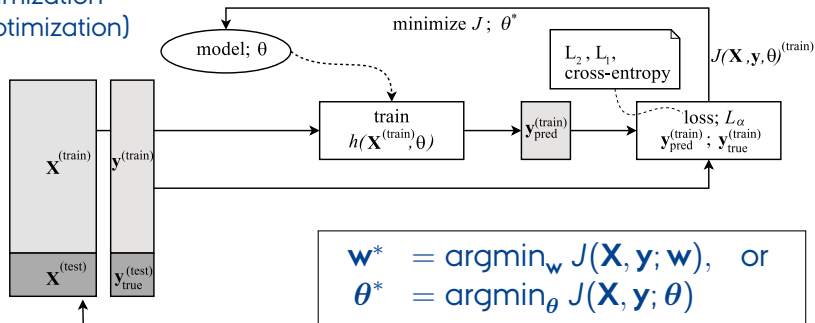
---

The Closed-form Linear-Least-Squares Solution

$$(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

# Training in General

Minimization  
(optimization)



Methods:

- Analytically: Closed-form solution

Exercise: L05/linear\_regression\_1.ipynb

Exercise: L05/linear\_regression\_2.ipynb

- Numerically: Gradient Descent

Exercise: L05/gradient\_descent.ipynb

# Exercise: L05/linear\_regression\_1.ipynb

## Training: The Closed-form Linear-Least-Squares Solution

To solve for  $\mathbf{w}^*$  in closed form, we find the gradient of  $J$  with respect to  $\mathbf{w}$

$$\nabla_{\mathbf{w}} J = \left[ \frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \dots, \frac{\partial J}{\partial w_m} \right]^\top$$

Taking the partial derivery  $\partial/\partial_{\mathbf{w}}$  of the  $J$  via the gradient (nabla) operator (*with a large amount of matrix algebra*)

$$\begin{aligned} \nabla_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0 \\ 0 &= \mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y} \end{aligned}$$

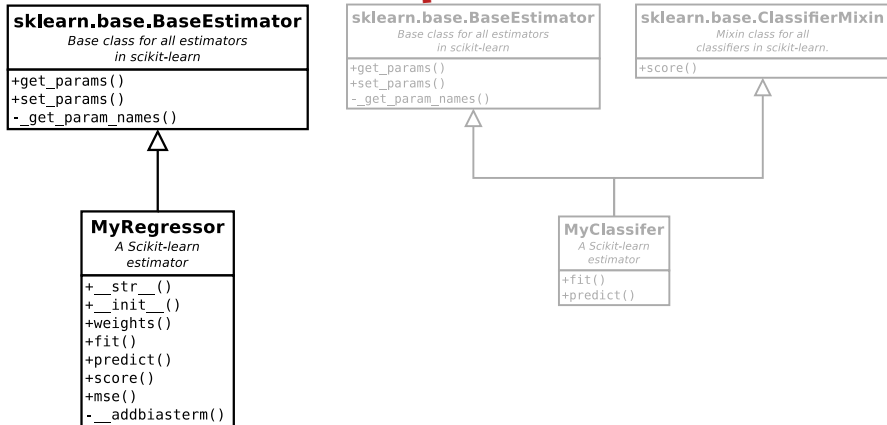
with a *small amount of matrix algegra*, this gives the **normal equation**

$$\begin{aligned} \mathbf{w}^* &= \operatorname{argmin}_{\mathbf{w}} \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \\ &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}, \quad \text{the normal eq.} \end{aligned}$$

# Exercise: L05/linear\_regression\_2.ipynb

Python class: `MyRegressor`

[OPTIONAL Exercise]



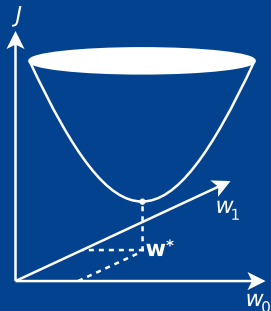
Exercise: create a linear regressor, inheriting from `BaseEstimator` and implement `score()` and `mse()`.

NOTE: no inhering from `ClassifierMixin`.

# COST FUNCTION MINIMIZATION VIA NUMERICAL SOLUTIONS

---

Gradient Descent



# (Full) Batch Gradient Descent (GD)

The nabla matrix differentiation,  $\nabla_{\mathbf{w}}$ , and the learning rate,  $\eta$

$$J(\mathbf{X}, \mathbf{y}; \mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \propto \text{MSE}(\mathbf{X}, \mathbf{y}; \mathbf{w})$$

$$\nabla_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) = \frac{1}{n} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}),$$

only when  $J \propto \text{MSE}$

$$\mathbf{w}^{\text{next step}} = \mathbf{w} - \eta \nabla_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w})$$

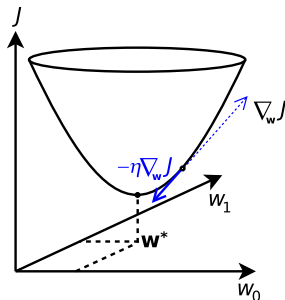
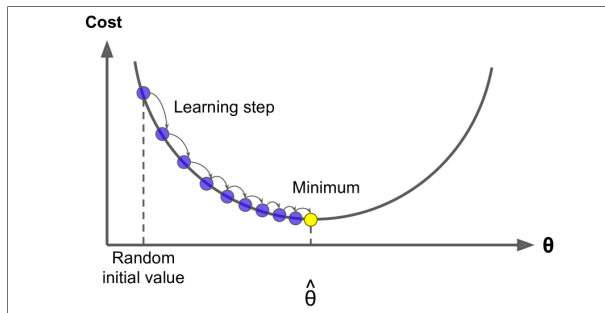


Figure 4-3. Gradient Descent

# Gradient Descent (GD)

## GD pitfalls

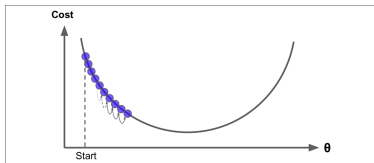


Figure 4-4. Learning rate too small

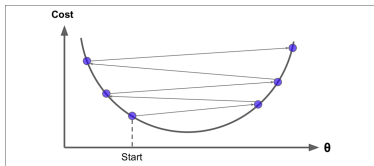


Figure 4-5. Learning rate too large

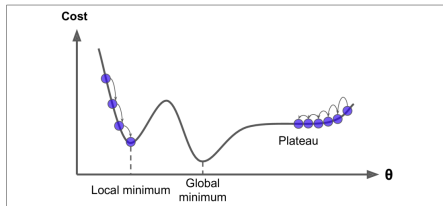


Figure 4-6. Gradient descent pitfalls

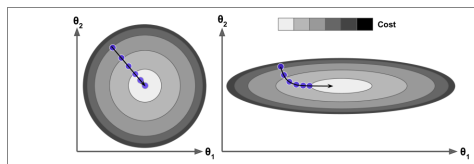
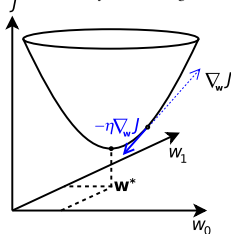


Figure 4-7. Gradient Descent with and without feature scaling





# Stochastic Gradient Descent (SGD)

$\mathbf{X}_{\text{SGD}} \Leftarrow$  one random sample  $\mathbf{x}^{(i)}$ 's from  $\mathbf{X}$

and this lowers the cost of calculating the gradient in each iteration

$$\nabla_{\mathbf{w}} J_{\text{SGD}}(\mathbf{X}_{\text{SGD}}, \mathbf{y}; \mathbf{w}) = \frac{1}{n} \mathbf{x}_{\text{SGD}}^{\top} (\mathbf{X}_{\text{SGD}} \mathbf{w} - \mathbf{y})$$

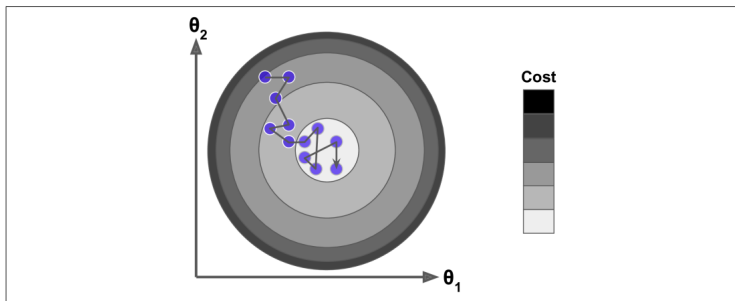


Figure 4-9. Stochastic Gradient Descent

# Mini-batch (stochastic) Gradient Descent (SGD)

$\mathbf{X}_{\text{mini}} \Leftarrow$  a set of random samples  $\mathbf{x}^{(i)}$ 's from  $\mathbf{X}$

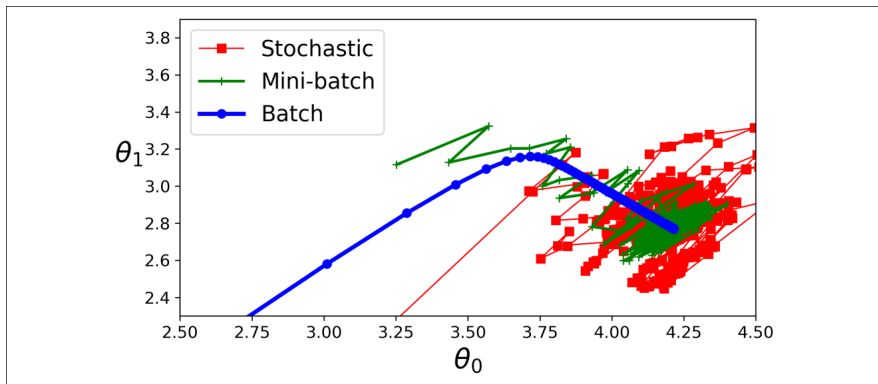


Figure 4-11. Gradient Descent paths in parameter space