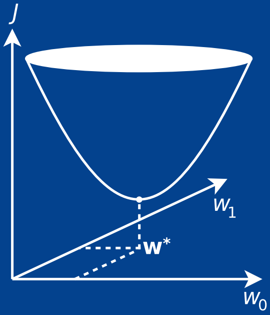
AARHUS UNIVERSITY

LESSON 5: Training (Regression, GD and SGD)

CARSTEN EIE FRIGAARD

SPRING 2025



L05: Training (Regression, GD and SGD)

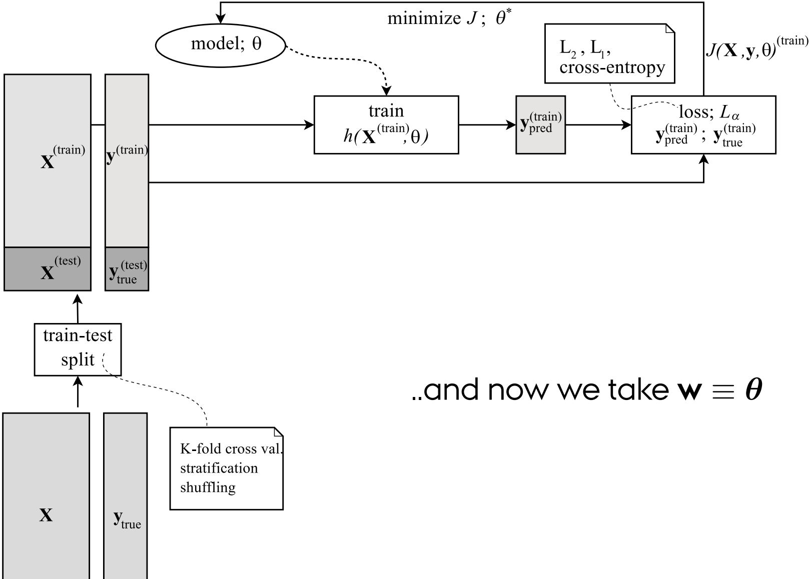
Agenda

- Revisiting 'The Map',
- Training a linear regression model,
  - (and intro to GD)
- Cost function in closed-form vs. numerical solutions.
  - analytical via $\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y}$
  - numerical via $\nabla_{\mathbf{w}} J$
- Gradient Descent (GD),
  - Learning rates,
  - Batch Gradient Descent (GD),
  - Stochastic Gradient Descent (SGD),
  - Mini-batch Gradient Descent.
- Opgave: `L05/train_linear_regression.ipynb`

TRAINING A LINEAR REGRESSOR

Training in General

Training is minimization of *J* (optimization)



minimize J ; θ^*
model; θ
L_2 , L_1, cross-entropy
$J(\mathbf{X},\mathbf{y},\theta)^{(\text{train})}$
train $h(\mathbf{X}^{(\text{train})},\theta)$
$\mathbf{y}_{\text{pred}}^{(\text{train})}$
loss; L_α $\mathbf{y}_{\text{pred}}^{(\text{train})}$; $\mathbf{y}_{\text{true}}^{(\text{train})}$
$\mathbf{X}^{(\text{train})}$
$\mathbf{y}^{(\text{train})}$
$\mathbf{X}^{(\text{test})}$
$\mathbf{y}_{\text{true}}^{(\text{test})}$
train-test split
K-fold cross val. stratification shuffling
$\mathbf{X}$
$\mathbf{y}_{\text{true}}$

..and now we take $\mathbf{w} \equiv \boldsymbol{\theta}$

Training a Linear Regressor

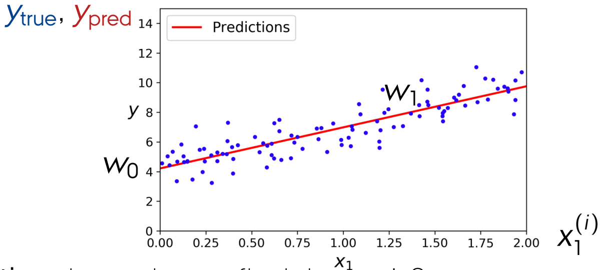
Linear Regression: In one dimension

The well know linear equation

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

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

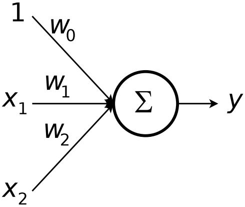
$$h(x^{(i)}; \mathbf{w}) \quad \overset{1D}{=} w_0 + w_1 x_1^{(i)} = y_{\text{pred}}$$



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

Training a Linear Regressor

Linear Regression: Hypotheis Function in N-dimensions



For 1-D:

$$h(x^{(i)}; \mathbf{w}) = w_0 + w_1 x_1^{(i)}$$

The same for N-D:

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

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

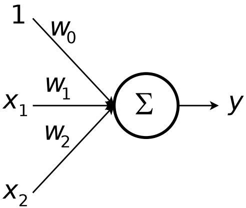
from Scikit-learn, use: `add_dummy_feature()`

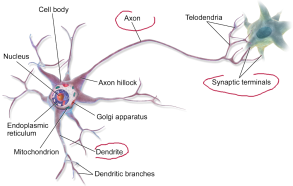
$$\begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix} \mapsto \mathbf{x}^{(i)},$$ by convention in the following...

yielding the vector form of the hypothesis function

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

Training a Linear Regressor





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

Oh, a half-neuron!!

Training a Linear Regressor

Linear Regression: Loss Function (or Cost/Objective Fun.)

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

$$\begin{aligned} L^{(i)} &= ||y_{\text{pred}}^{(i)} - y^{(i)}||_2^2 \\ &= ||h(\mathbf{x}^{(i)};\mathbf{w}) - y^{(i)}||_2^2 \\ &= \left(\mathbf{w}^\top\mathbf{x}^{(i)} - y^{(i)}\right)^2 \end{aligned}$$

$y \equiv y_{\text{true}}$ in the following

only when y is 1-D

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

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

only when y is 1-D

Ignoring constant factors, this yields our linear regression cost function

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

Training a Linear Regressor

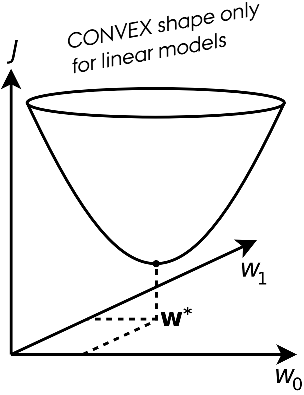
Minimizing the Linear Regression: The `argmin` concept

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

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



and by minima, we naturally hope for

- the global minumum

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

- local minimum

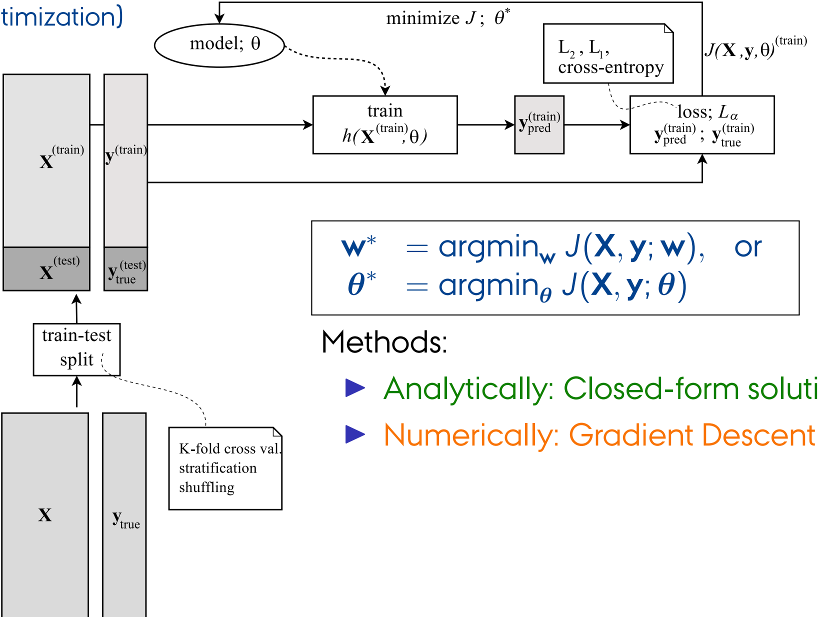
COST FUNCTION MINIMIZATION IN CLOSED-FORM

The Closed-form Linear-Least-Squares Solution

$$\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

Training in General

Minimization (optimization)



minimize J; θ^*
model; θ
L_2, L_1,
cross-entropy
$J(\mathbf{X},\mathbf{y},\theta)^{(\text{train})}$
$\mathbf{X}^{(\text{train})}$
$\mathbf{y}^{(\text{train})}$
train
$h(\mathbf{X}^{(\text{train})},\theta)$
$\mathbf{y}_{\text{pred}}^{(\text{train})}$
loss; L_α
$\mathbf{y}_{\text{pred}}^{(\text{train})}$; $\mathbf{y}_{\text{true}}^{(\text{train})}$
$\mathbf{X}^{(\text{test})}$
$\mathbf{y}_{\text{true}}^{(\text{test})}$
train-test
split
$\mathbf{X}$
$\mathbf{y}_{\text{true}}$
K-fold cross val.
stratification
shuffling

$$
\begin{aligned}
\mathbf{w}^* &= \operatorname{argmin}_{\mathbf{w}} J(\mathbf{X},\mathbf{y};\mathbf{w}), \quad \text{or} \\
\boldsymbol{\theta}^* &= \operatorname{argmin}_{\boldsymbol{\theta}} J(\mathbf{X},\mathbf{y};\boldsymbol{\theta})
\end{aligned}
$$

Methods:

- Analytically: Closed-form solution
- Numerically: Gradient Descent

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 $\mathbf{w}$

$$\nabla_{\mathbf{w}} J = \left[\frac{\partial J}{\partial w_1} \quad \frac{\partial J}{\partial w_2} \quad \cdots \quad \frac{\partial J}{\partial w_d}\right]^\top$$

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

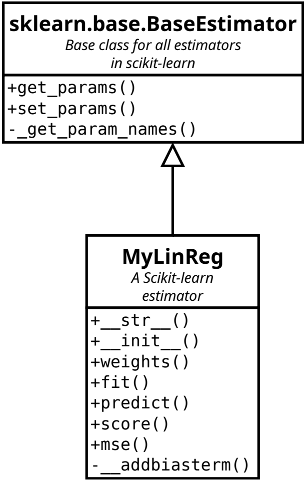
$$\begin{aligned} \nabla_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \ = \ 0 \\ 0 &= \mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y} \end{aligned}$$

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

$$\begin{aligned} \mathbf{w}^* &= argmin_{\mathbf{w}} \ \tfrac{1}{2} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \\ &= \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}, \qquad \textit{the normal eq.} \end{aligned}$$

Exercise: train_linear_regression.ipynb

Python class: MyLinReg



sklearn.base.BaseEstimator

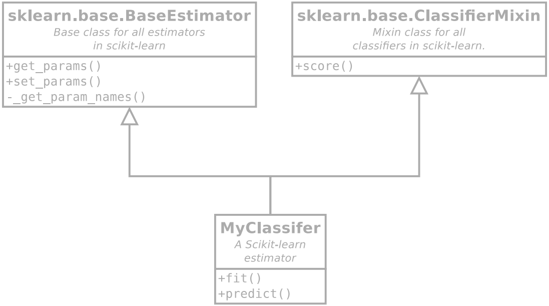
Base class for all estimators in scikit-learn

+get_params()
+set_params()
-_get_param_names()

MyLinReg

A Scikit-learn estimator

+__str__()
+__init__()
+weights()
+fit()
+predict()
+score()
+mse()
-__addbiasterm()



sklearn.base.BaseEstimator

Base class for all estimators in scikit-learn

+get_params()
+set_params()
-_get_param_names()

sklearn.base.ClassifierMixin

Mixin class for all classifiers in scikit-learn.

+score()

MyClassifer

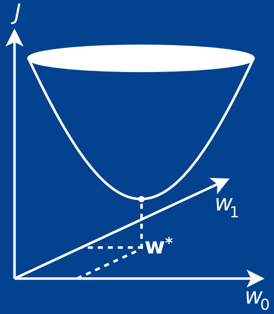
A Scikit-learn estimator

+fit()
+predict()

`Exercise:` *Create a linear regressor, with a Scikit-learn compatible fit-predict interface. You should implement every detail [..]*

COST FUNCTION MINIMIZATION VIA NUMERICAL SOLUTIONS

Gradient Descent



(Full) Batch Gradient Descent (GD)

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

$$
\begin{aligned}
J(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= \tfrac{1}{2}||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \;\propto \mathrm{MSE}(\mathbf{X}, \mathbf{y}; \mathbf{w}) \\
\nabla_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= \tfrac{1}{n}\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}), \\
\mathbf{w}^{\text{next step}} &= \mathbf{w} - \eta \nabla_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w})
\end{aligned}
$$

$1/n$ only when $J = \mathrm{MSE}$

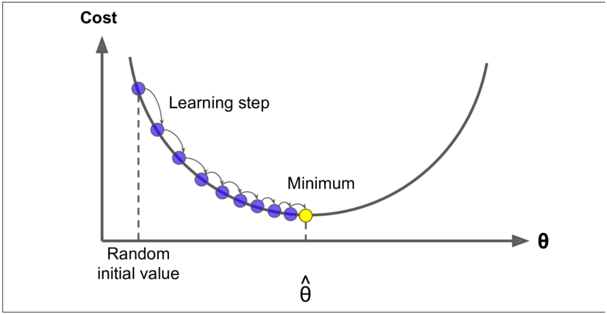
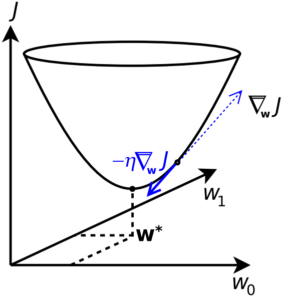


Figure 4-3. Gradient Descent



Gradient Descent (GD)

GD pitfalls

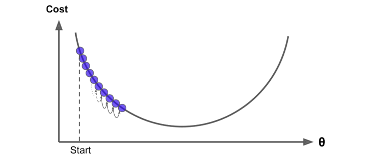


Figure 4-4. Learning rate too small

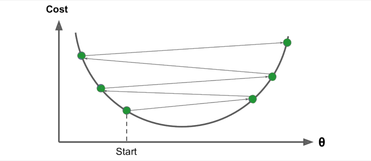


Figure 4-5. Learning rate too large

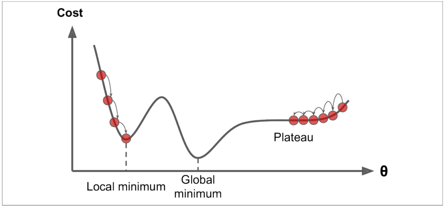


Figure 4-6. Gradient Descent pitfalls

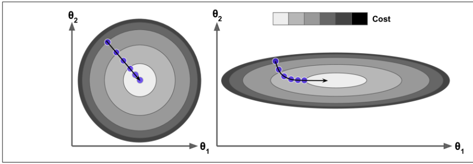
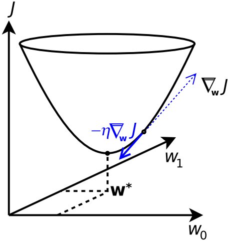


Figure 4-7. Gradient Descent with and without feature scaling



Learning Curve for GD

Plot J for Fig 4-4, 4.5 and 4.6 over 'time' or iteration in the numerical gradient descent algorithm..

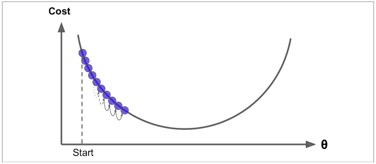


Figure 4-4. Learning rate too small

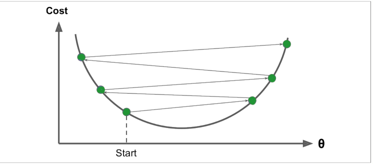


Figure 4-5. Learning rate too large

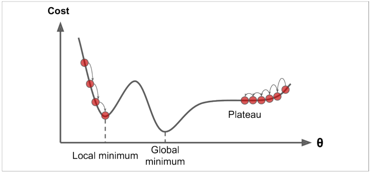
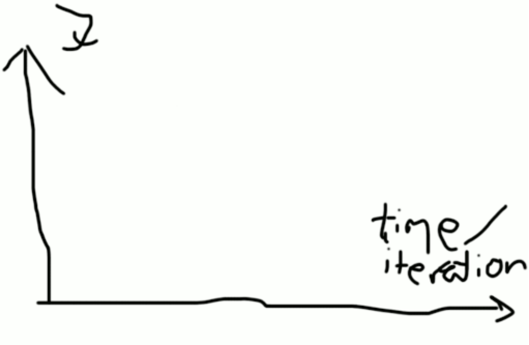


Figure 4-6. Gradient Descent pitfalls



Stochastic Gradient Descent (SGD)

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

and this lowers the computational effort of calculating the gradient in each iteration

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

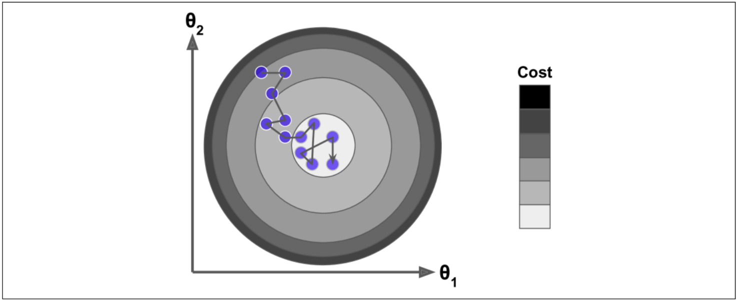


Figure 4-9. Stochastic Gradient Descent

Mini-batch (stochastic) Gradient Descent (SGD)

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

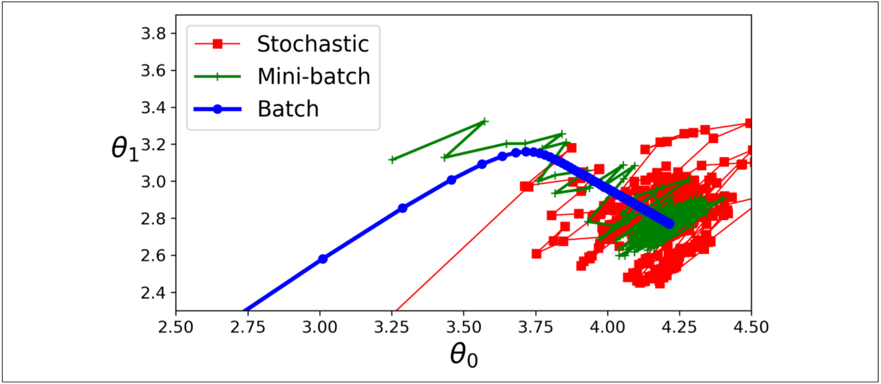


Figure 4-11. Gradient Descent paths in parameter space