linear regression

Lecture 02
23.09.2024

Outline

- Linear regression
- Empirical loss function/cost function
- Optimizing the empirical loss function
- Overfitting/underfitting

Review from last week

Introduced Artificial Intelligence (AI)

Machine learning approach (ML) to AI

supervised learning

$\{x^i, y^i\}_{i=1}^N$

x^i: feature vector

y^i: labels

$f: x \mapsto y$

Find f

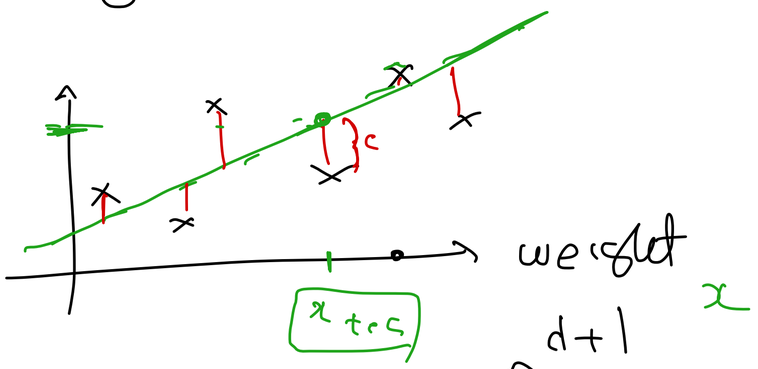
unsupervised learning

$\{x^i\}_{i=1}^N$

ex: pattern recognition such as clustering or dimensionality reduction

Theory - linear predictors

Training data $\{x^i, y^i\}_{i=1}^N$ $x^i \in \mathbb{R}^d$, $y^i \in \mathbb{R}^m$



Linear predictor $f(x) = b + w_1 x_1 + \ldots + w_d x_d$

offset

$(b, w_1, \cdots, w_d) \in \mathbb{R}^{d+1}$ is the parameter

x

Note: often $w = (b, w_1, \cdots, w_d) \in \mathbb{R}^{d+1}$, $x = (1, x_1, \cdots, x_d) \in \mathbb{R}^{d+1}$, $f_w(x) = w^T x = [b, w_1, \ldots, w_d] \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$

Tuning, training, fitting a parameter: finding the best parameter $w \in \mathbb{R}^{d+1}$

Linear Regression

Loss function (also referred to as cost or objective function)

Q: How to choose w_1 and b such that the predicted value $\hat{y}^i = wx^i + b$ is as close as possible of the real value $y^{(i)}$?

Prediction, prediction error

Consider x^i, true label y^i

predicted label $\hat{y}^i = w^T x$

$$e^i = (\hat{y}^i - y^i)$$

Squared prediction error

$$(\hat{y}^i - y^i)^2$$

Mean squared prediction error

$$\frac{1}{N} \sum_{i=1}^{N} (\hat{y}^i - y^i)^2$$

Empirical loss/cost function for training:

$$w = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \in \mathbb{R}^{d+1}$$

$$J(w) = \frac{1}{N} \sum_{i=1}^{N} (w^T x^i - y^i)^2$$

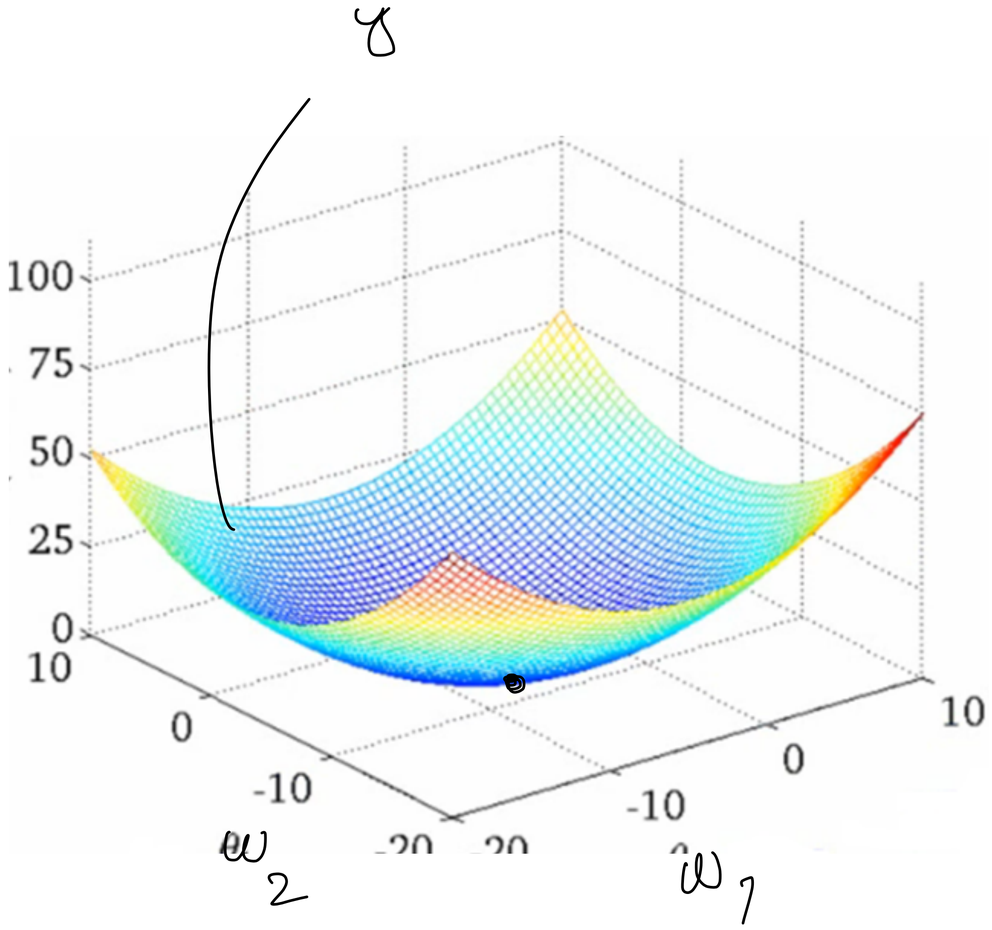
Linear Regression
Optimizing the loss function

Q: How to minimise the loss function?

$$J(w) = \frac{1}{N} \sum_{i=1}^{N} (w^T x^i - y^i)^2$$

$$\nabla J(w) = \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_p} \end{bmatrix}, \text{ for } J : \mathbb{R}^p \to \mathbb{R}$$

$\in \mathbb{R}^p$



Optimization - brief review

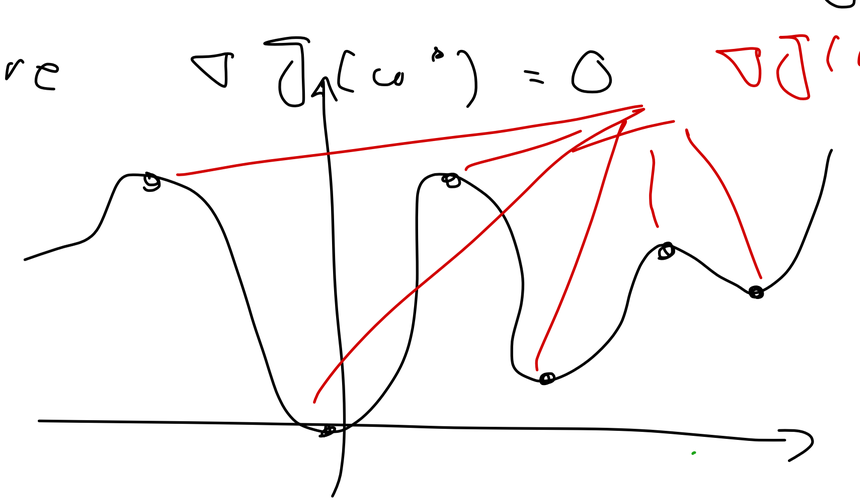
Unconstrained minimisation of differentiable function

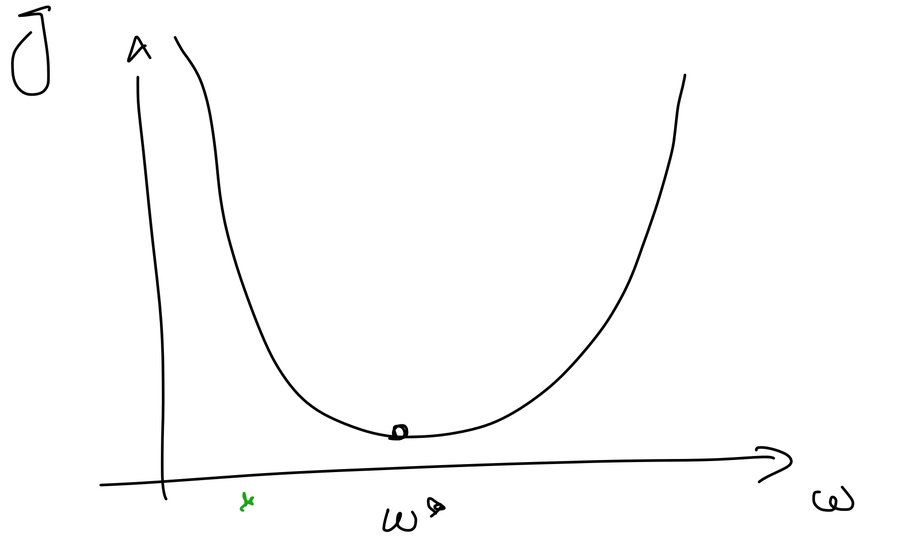
$$J : \mathbb{R}^p \to \mathbb{R}, \quad \min_w J(w)$$

if w^* is a minimizer then $\nabla J(w^*) = 0 \in \mathbb{R}^p$

Necessary optimality condition

first order condition: any w^* minimizer should be a point where $\nabla J(w^*) = 0$





$\nabla J(w^*) = 0$ is not a sufficient condition for w^* to be a minimum

Optimization - convex functions

For convex functions, the first-order condition of optimality is necessary and sufficient

w^* is optimal $\Longleftrightarrow \nabla J(w^*) = 0 \in \mathbb{R}^p$

Characterization of convexity for twice differentiable functions based on the Hessian matrix

$J : \mathbb{R}^p \to \mathbb{R}$ is convex if Hessian of $J(\cdot)$ is

positive semi definite.

Hessian : $H := \nabla^2 J(w) \in \mathbb{R}^{p \times p}$

$H_{ij}(w) = \dfrac{\partial^2 J}{\partial w_i \partial w_j}(w), \quad i, j = 1, \dots, p$

Review - positive (semi) definite matrices

Definition : a matrix $A \in \mathbb{R}^{p \times p}$ is positive (semi) definite if

$\forall\, w \in \mathbb{R}^p, \quad w^T A w \; (\geq 0) > 0$ for any $w \neq 0$.

(semi)

Consider the matrices $A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $A_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, $A_3 = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$

Which are positive (semi) definite?

A_1 : $\begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = w_1^2 + w_2^2$ positive definite

A_2 : $w_1^2 + w_2^2 + w_1 w_2$ $\Longrightarrow$ positive definite

A_3 : $w_1^2 + w_2^2 + 4 w_1 w_2$ $\Longrightarrow$ not positive semi definite

Review - positive (semi) definite matrices

Test for positive (semi)definiteness

$A \in \mathbb{R}^{p \times p}$, $w \in \mathbb{R}^{p \times 1}$

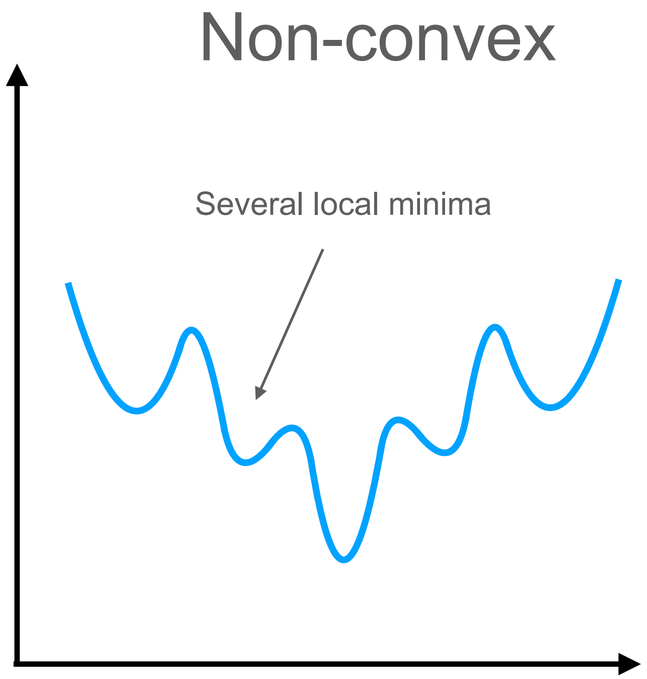
$$w^T A w = w^T A^T w \Rightarrow w^T A w = w^T \left(\frac{A + A^T}{2} \right) w$$

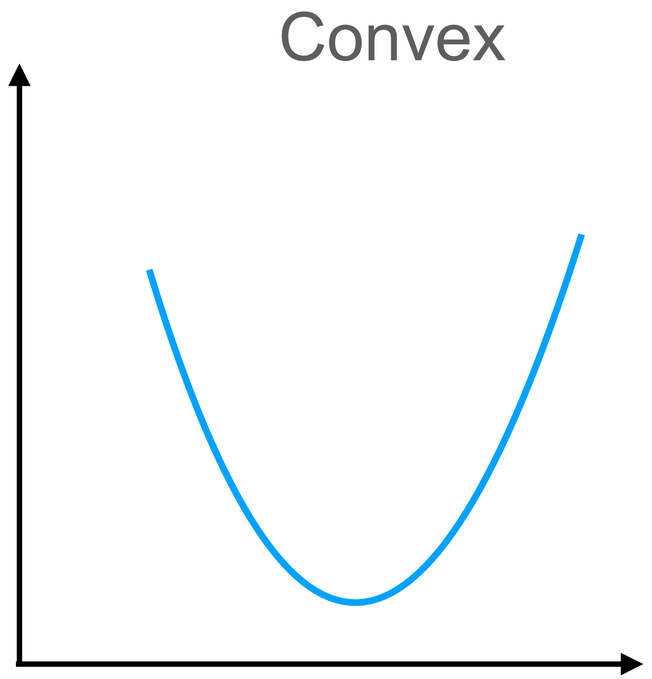
symmetric part of A.

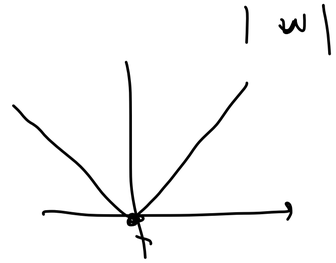
A symmetric matrix M is positive (semi) definite iff all eigenvalues of M are positive (non-negative).

$\Rightarrow$ A is positive (semi) definite $\Leftrightarrow$ $A + A^T$ has positive (non-negative) eigenvalues.

Optimization of convex function







Questions during break:

1) why not take $e^i = |w^T x^i - y^i|$ (absolute value error?)

2) if there are multiple w's such that $\nabla J(w) = 0$, which one gives lower cost? for convex function doesn't matter all give same cost

Review - gradient vector and Hessian matrix

For an affine function

Consider $f : \mathbb{R}^p \to \mathbb{R}$. Suppose $f(w) = a^T w$

$$\nabla f(w) = a \in \mathbb{R}^p = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \\ \vdots \\ \frac{\partial f}{\partial w_p} \end{bmatrix}$$

$$\nabla^2 f(w) = 0 \in \mathbb{R}^{p \times p}$$

Review - gradient vector and Hessian matrix

For a quadratic function

$f(w) = w^T A w \qquad f : \mathbb{R}^p \to \mathbb{R}$

$\nabla f(w) = (A + A^T) w \qquad \in \mathbb{R}^p$

$\nabla^2 f(w) = A + A^T$ (note : if A is symmetric)

$\nabla^2 f(w) = 2A$

(algebra).

See additional notes posted for derivation

Linear Regression - Loss function optimization

Loss function is quadratic in the parameter

$$J(w) = \frac{1}{N}\sum_{i=1}^{N}(w^T x^i - y^i)^2$$

Gradient vector of the loss function

$$\nabla_w J(w) \in \mathbb{R}^{d+1}$$

$$\nabla_w J(w) = \nabla_w \left(\frac{1}{N}\sum_{i=1}^{N}(w^T x^i - y^i)^2\right) = \frac{1}{N}\sum_{i=1}^{N}\nabla_w (w^T x^i - y^i)^2$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left[x^i (w^T x^i - y^i) + x^i (w^T x^i - y^i)\right] =$$

$$= \frac{2}{N}\sum_{i=1}^{N} x^i (w^T x^i - y^i)$$

Linear Regression - gradient of the loss function

$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial b} \\ \frac{\partial J(w)}{\partial w_1} \\ \frac{\partial J(w)}{\partial w_2} \\ \cdots \\ \frac{\partial J(w)}{\partial w_d} \end{pmatrix} = \frac{2}{N} \sum_{i=1}^{N} x^i (w^T x^i - y^i), \quad \text{where } w \in R^{d+1} = \begin{pmatrix} b \\ w_1 \\ \cdots \\ w_d \end{pmatrix} = w_0 \in \mathbb{R}^{d+1}$$

$$x^i = (1, x^i_1, \cdots, x^i_d) \in \mathbb{R}^{d+1}$$

Vector notation for gradient of the loss function

Define data matrix $X \in R^{N\times(d+1)} = \begin{pmatrix} 1 & x_1^1 & x_2^1 & \dots & x_d^1 \\ 1 & x_1^2 & x_2^2 & \dots & x_d^2 \\ \dots & & & & \\ 1 & x_1^N & x_2^N & \dots & x_d^N \end{pmatrix}$, $w \in R^{d+1} = \begin{pmatrix} b \\ w_1 \\ \dots \\ w_d \end{pmatrix}$, $y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix} \in \mathbb{R}^N$

Verify that: $J(w) = \frac{1}{N}\sum_{i=1}^{N}(w^T x^i - y^i)^2$ **Can be equivalently written as :** $J(w) = \frac{1}{N}(Xw - y)^T(Xw - y)$

↳ if question, ask in exercise hour

$J(w) = \frac{1}{N}\left(w^T X^T X w - w^T X^T y + y^T y \right)$

Conclude $\nabla J(w) = \frac{2}{N}X^T(Xw - y)$ $\nabla^2 J(w) = \frac{2}{N}X^T X$

Linear Regression - Loss function optimization

$$\frac{\partial J(w)}{\partial w} = \frac{2}{N} X^T (Xw - y) = 0 \iff w = (X^T X)^{-1} X^T y$$

$$\nabla^2 J(w) = \frac{2}{N} X^T X$$

Verify that the loss function is convex : $M = A^T A$ $\quad w^T A^T A w = \tilde{w}^T \tilde{w}, \quad \tilde{w} = Aw$, ≥ 0

Optimizer

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

Under which conditions $X^T X$ is invertible ?

Linear Regression - Loss function optimization

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

Q: is this computationally efficient for large data sets?

inverse of $\mathbf{X}^T\mathbf{X}$:

- Computational complexity is $O(d^{2.4})$ to $O(d^3)$ depending on the algorithm used..

Also, in many ML approaches we may not be able to find the solution of $\nabla J(w^*) = 0$, to find a good parameter.

Linear Regression
Gradient Descent

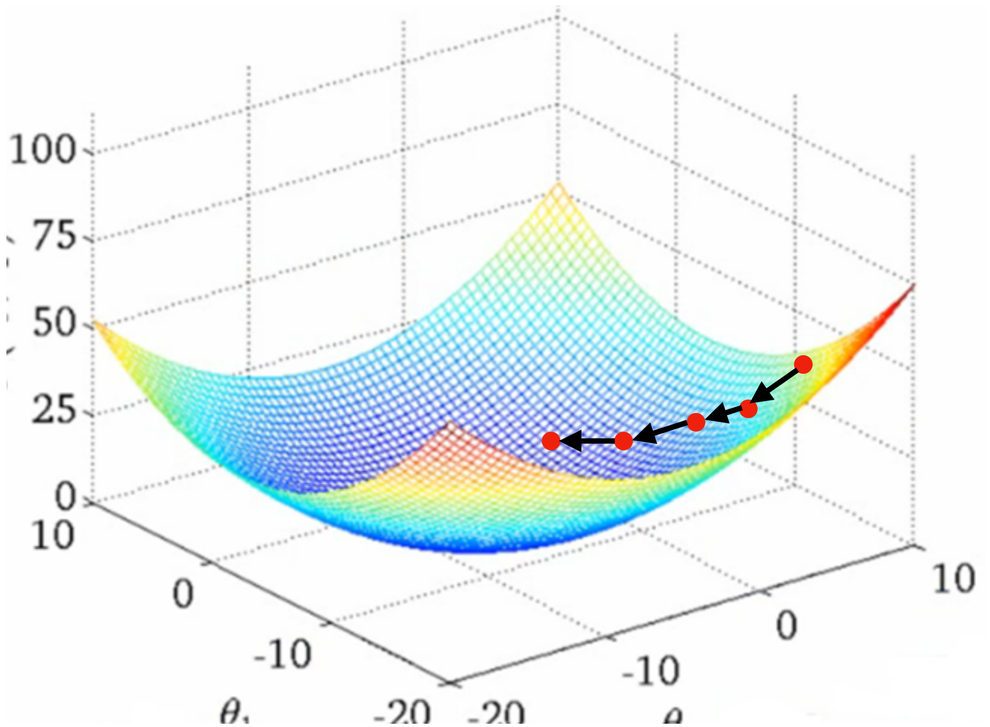
Alternative approach to compute $\arg\min_w J(w) = \frac{1}{N}\sum_{i=1}^{N}(w^T x^i - y^i)^2$

Gradient Descent:

- Calculate the gradient of $J(w)$ at the starting point.
- Iteratively update (w) by taking steps in the direction of the negative gradient

Linear Regression

Optimizing loss function through gradient descent



$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial b} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \frac{\partial J(\mathbf{w})}{\partial w_2} \\ \cdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{pmatrix}$$

The iterative update is performed as follows

- $\mathbf{w}^{\,next\ iteration} = \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$

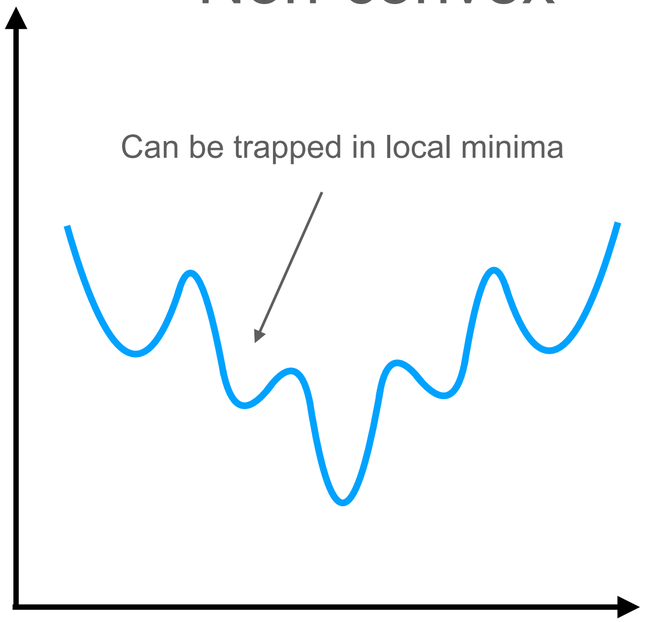
step size / learning rule

$w \in \mathbb{R}^{d+1}$, $\alpha \in \mathbb{R}_+$

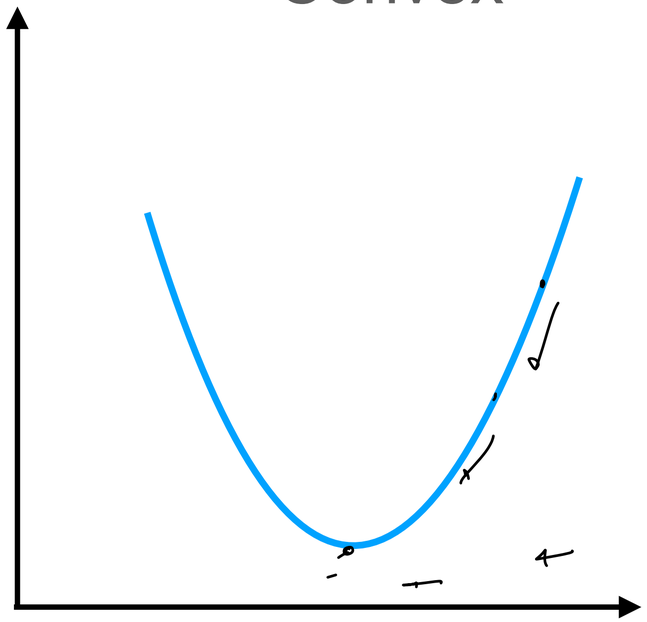
$\nabla_w J(w) \in \mathbb{R}^{d+1}$

Does gradient descent converge?

Non-convex



Convex



for convex function by suitably choosing α, gradient descent converges to w^* (an optimized)

for non-convex function you may get stuck in a local minimum

Linear Regression

Optimizing loss function through gradient descent

$$w(t+1) = w(t) - \alpha \nabla J(w(t))$$

Since loss function of linear of linear regression is convex, there exists a choice of learning rate/step size α so that the iteration converges .

Nonlinear transformations of data

$\{x^i, y^i\}_{i=1}^N$

- Feature map/feature vector $\phi : \mathbb{R}^d \to \mathbb{R}^p$

example: let $x \in \mathbb{R}^2$, $\phi_1(x_1, x_2) = x_1 x_2$, $\phi_2(x_1, x_2) = \sin x_1$

$\phi_3(x_1, x_2) = \cos x_2$

$f_w(x) = b + w_1 \phi_1(x) + w_2 \phi_2(x) + w_3 \phi_3(x)$

$J(w) = \frac{1}{N} \sum_{i=1}^{N} (w^T \phi(x^i) - y^i)^2$, $\phi(x^i) = \begin{bmatrix} 1 \\ \phi_1(x^i) \\ \phi_2(x^i) \\ \phi_3(x^i) \end{bmatrix}$, $w = \begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$

J is still convex quadratic in w.

Exercise: compute $\nabla_w J(w)$, $\nabla_w^2 J(w)$

Feature expansion
Overview

Feature expansion is the process of creating derived features from the input data

- Adds complexity
- Can significantly improve performance

Popular feature expansion techniques:

- **Feature crosses:** Multiply features together
- **Applying a non-linear function to each feature** (*e.g.,* sin, log, polynomials)
- Often requires insight about the problem

Feature expansion
Polynomial feature expansion

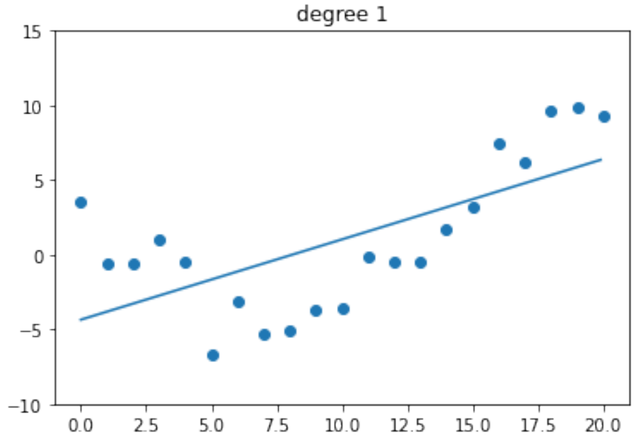
Polynomial feature expansion:

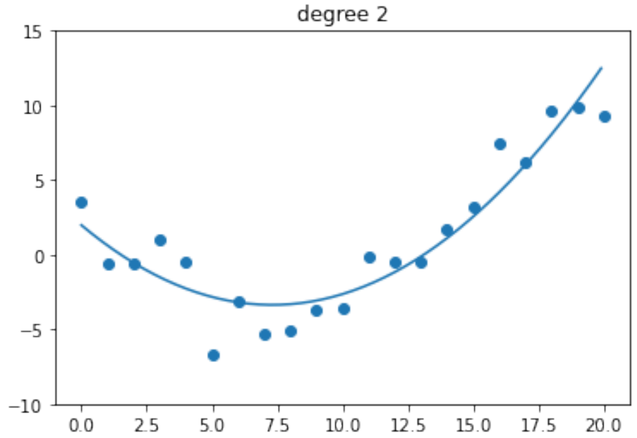
Generate a new feature vector consisting of all polynomial combinations of the features with degree less than or equal to a given degree

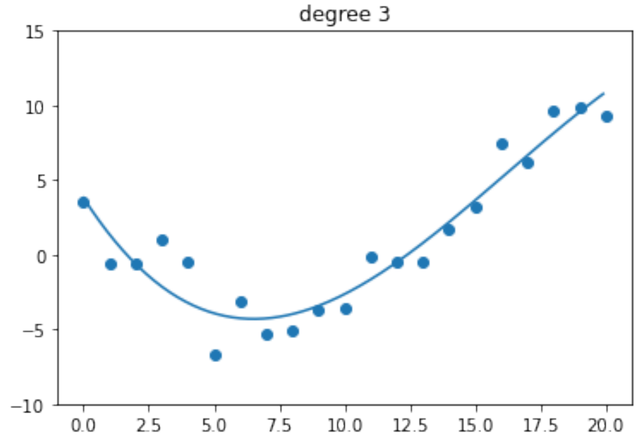
$$\boldsymbol{x} = (x_1, x_2, x_3) \xrightarrow{\text{Second order polynomial feature expansion}} (1,\ x_1,\ x_2,\ x_3,\ x_1^2,\ x_1x_2,\ x_1x_3,\ x_2^2,\ x_2x_3,\ x_3^2)$$

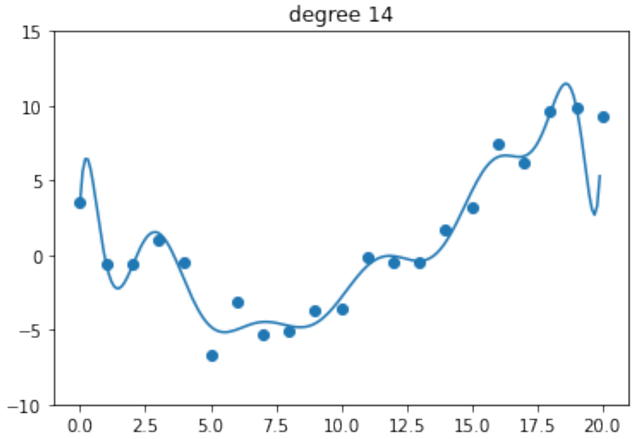
Feature expansion

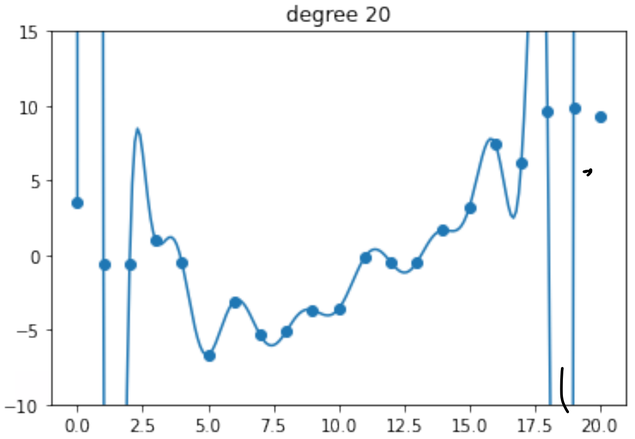
Polynomial feature expansion











Overfitting and underfitting

Train and test set

Goal of supervised ML models: generalise well on new data (based on the patterns learned from known data).

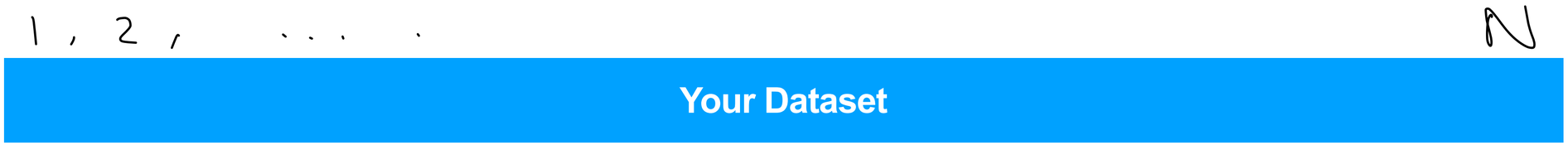
Performance metric:

example $\left(f_w(x)_{test} - y_{test} \right)^2$

$\frac{1}{N} \sum_{i=1}^{N} \left(f_w(x^i) - y^i \right)^2$

Training versus Test performance: how well does the model predict the output of an "unseen" data x_{test}

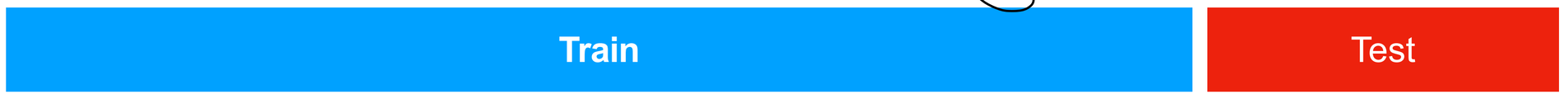
Train and test



Split data into **train** and **test**

80% train

20% test



indices $i_1, i_2, \dots, i_{N_1}$ for train

$i_{N_1+1}, i_{N_1+2}, \dots, i_N$ for test

$\{x_i, y_i\}_{i=1}^{N}$

Later in the course we will see split into 3 sets: train, validate and test sets in order to find best “hyper”-parameters of the model

Underfitting & Overfitting

Goal of supervised ML models: generalise well on new data (based on the patterns learned from known data).

Two situations where it fails:

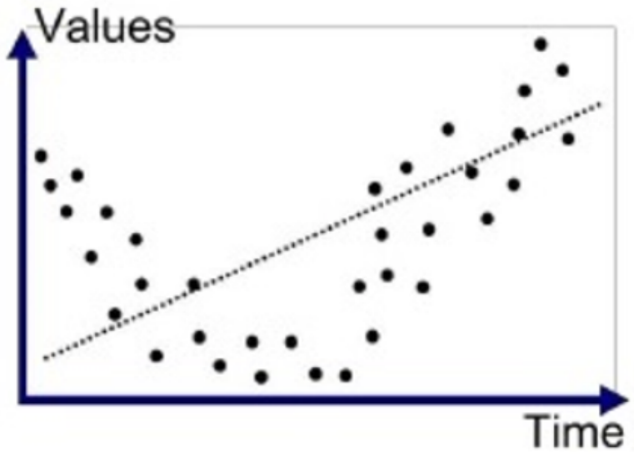
- Underfitting
- Overfitting

Underfitting & Overfitting

Underfitting

Underfitting: the model doesn't fit well on the training data.

Reason: model too simple → can't capture the underlying patterns within the data.



The model (the line) doesn't capture the U shape of the data set.

→ **Solution**: choose a more complex algorithm/model to better fit the data.

Summary

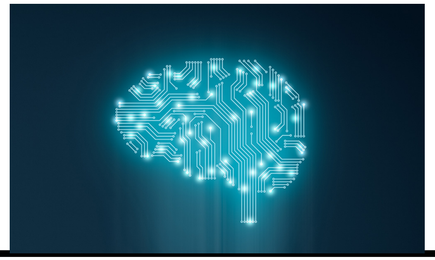
- Linear regression
- Empirical loss function/cost function
- Optimizing the empirical loss function
- Overfitting/underfitting

mohv

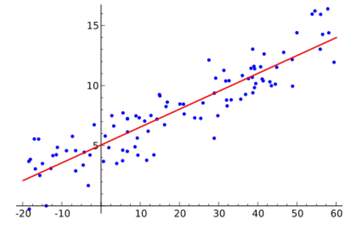
Exercise hour

- This week:
  - Review background on gradient, Hessian and optimization (see notes posted)
  - You will use python to do a linear regression
  - Link to the exercises will be published on moodle

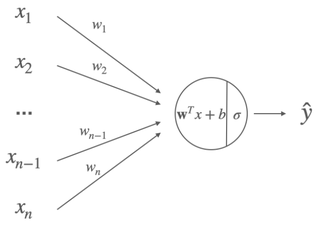
Introduction



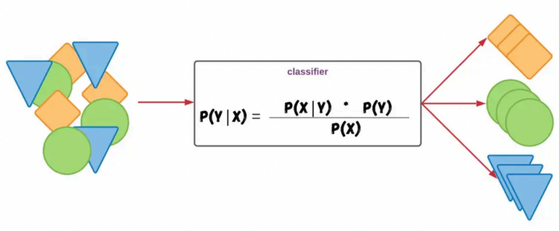
Linear regression



Logistic regression



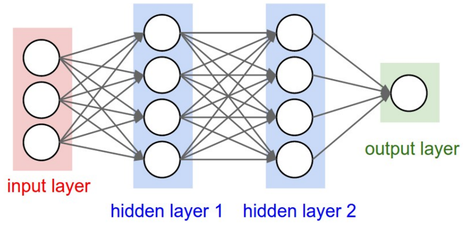
Naive Bayes



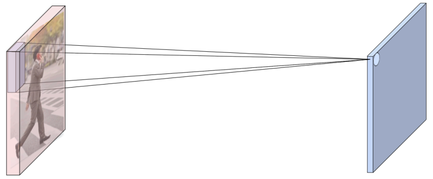
KNN

AI ethics

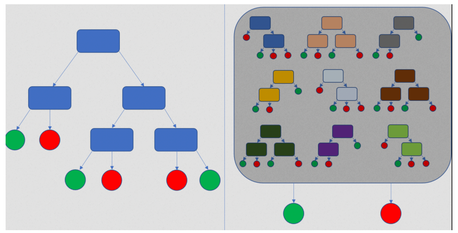
Neural networks



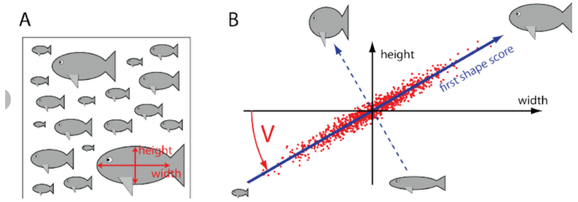
Convolutional neural networks



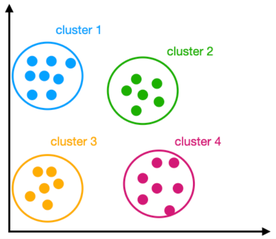
Decision-trees



Dimensionality reduction



Clustering



Reinforcement learning