Machine Learning for Physics and Astronomy

Juan Rojo

VU Amsterdam & Theory group, Nikhef

Natuur- en Sterrenkunde BSc (Joint Degree), Honours Track

Lecture 7, 12/10/2020

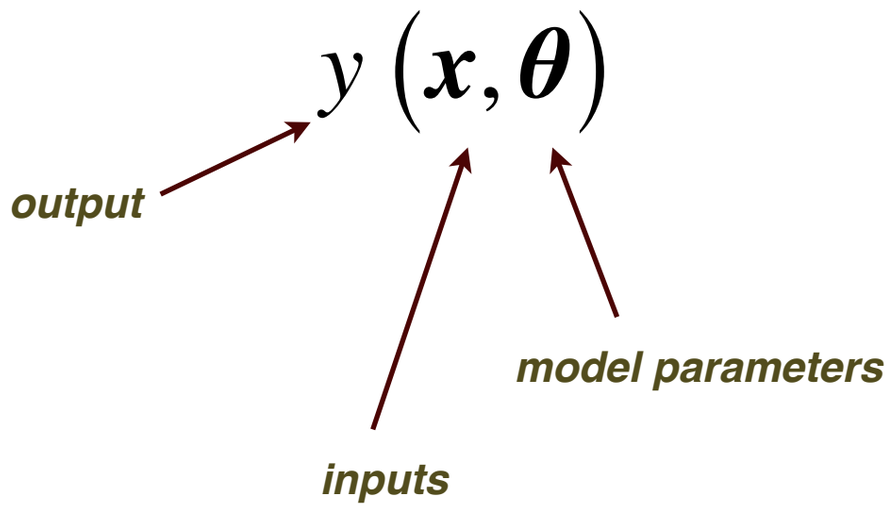
Today's lecture

- Kernel methods and the Dual Representation
- Support Vector Machines
- Gaussian Processes
- Variational Inference
- Guest lecture by **Dr. Tristan Bereau** on applications of machine learning in condensed matter

Kernel Methods

Kernel Methods

most of the ML models we have considered for regression and classification are based on the **parametric mapping** (linear or non-linear) between inputs and outputs



after the training, the input examples are **discarded** and not relevant for subsequent **predictions**: the information contained in the original training examples is now **entirely contained** on the model parameters

Kernel Methods

most of the ML models we have considered for regression and classification are based on the **parametric mapping** (linear or non-linear) between inputs and outputs

$$y\left(\boldsymbol{x}, \boldsymbol{\theta}\right)$$

info on training samples entirely contained in model params

were the **training examples are discarded** after the model parameters (or their posterior distribution) have been determined

here we introduce models where the training dataset is also used in the **prediction phase**

Feature space mapping

A key concept of **kernel methods** is that of a **feature space mapping**

This is a transformation, in general non-linear, between the **input data space *X*** and a **feature space *F*** of the same or different dimensions:

$$F = \{\phi(x) : x \in X\}$$

Example of linear feature-space transformation

x	$\phi(x)$
Height	*Height+Weight*
Weight	*Weight-Height*
Age	*Age + 2*Weight*

Example of non-linear feature-space transformation

x	$\phi(x)$
Height	*Height/Weight*
Weight	*Weight+Height*
Age	*Age * Weight*

such mapping can be chosen **to increase the efficiency** of the ML model training

note that the dimensions of feature space can be much larger than of data space

Feature space mapping

most of the ML models we have considered for regression and classification are based on the **parametric mapping** (linear or non-linear) between inputs and outputs

$$y\left(\boldsymbol{x}, \boldsymbol{\theta}\right)$$

info on training samples entirely contained in model params

were the **training examples are discarded** after the model parameters (or their posterior distribution) have been determined

here we introduce models where the training dataset is also used in the **prediction phase**

the dimensions of the **feature space mapping** are in general different from those of input space

$$\boldsymbol{x} \longrightarrow \boldsymbol{\phi}\left(\boldsymbol{x}\right)$$

Height

Weight

Age

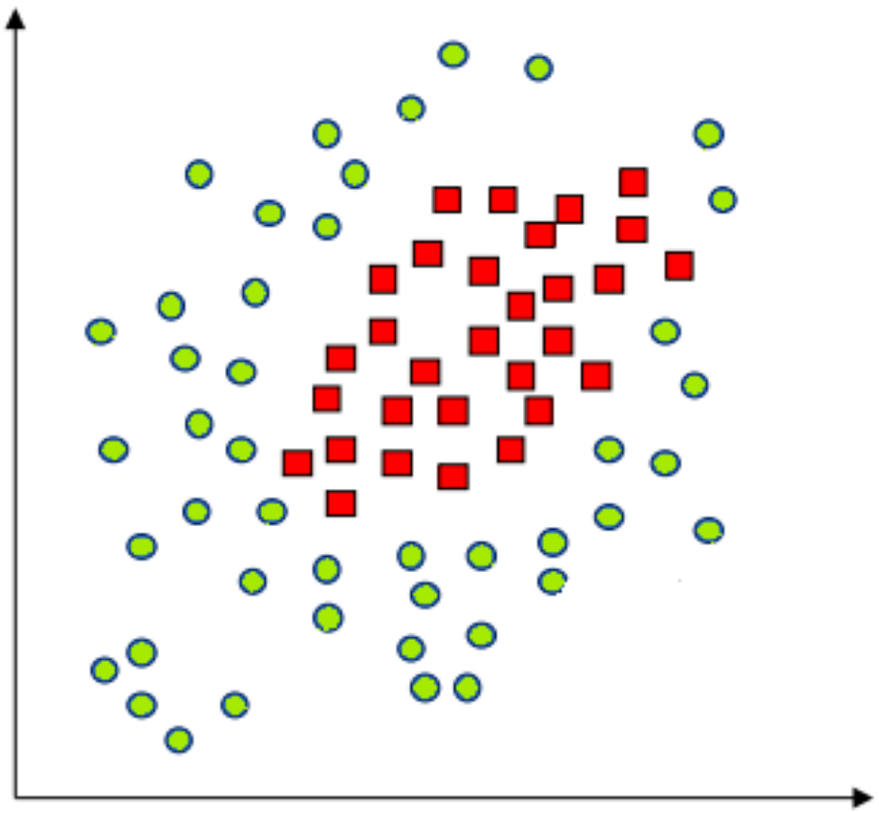
Height/Weight

Weight + Height

Age * Weight

Height * Weight+Age

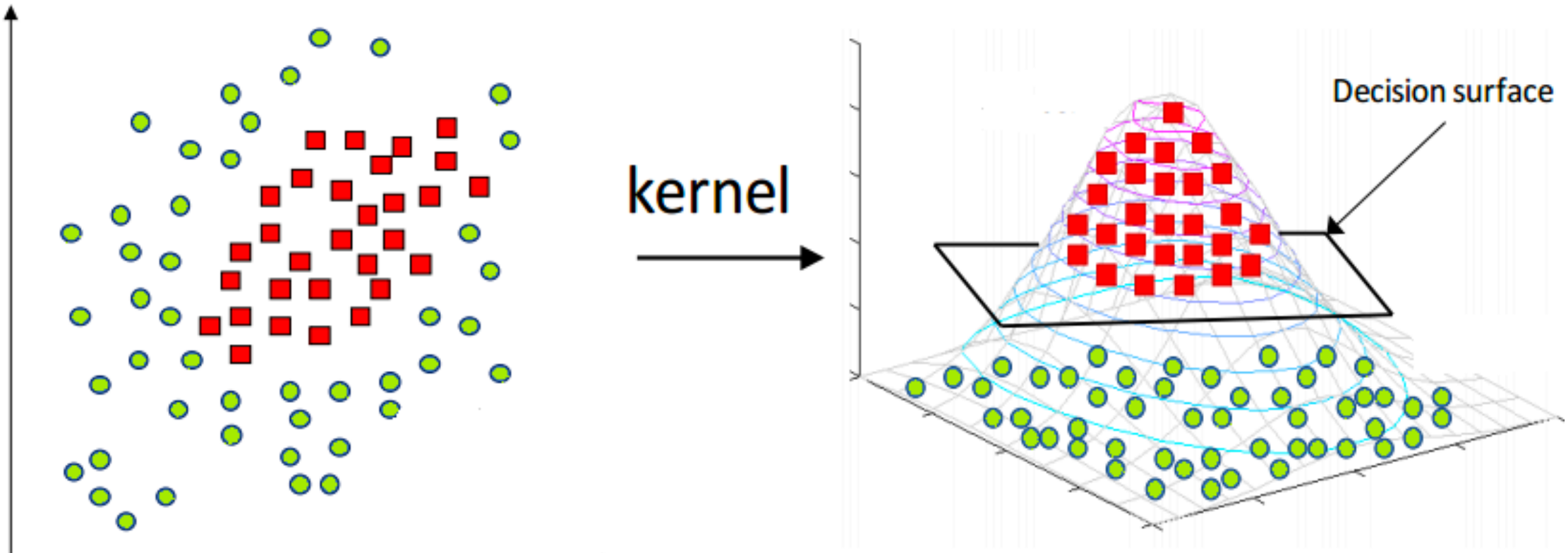
Feature space mapping



Data Space (p=2):

linear classification impossible

Feature space mapping



Data Space (p=2):

linear classification impossible

Feature space (p=3):

linear classification possible!

Kernel Methods

the key ingredient of **kernel methods** is the **kernel function** evaluated at the training points

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$$

inner product in feature space

kernel function (scalar)

feature space mapping

the simplest kernel is the **linear kernel** (identity mapping in feature space)

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}' = \sum_{k=1}^{p} x_i x_i'$$

sum over p,

dimension of input data space

Kernel Methods

in kernel methods, the training dataset is also used in the **prediction phase**

The advantages of kernel methods arise when choice special, non-linear kernels

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$$

for example, assume a two-dimensional problems (***p=2***) with the following feature mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1 x_2, x_1 - x_2\right)$$

then the corresponding kernel will be given by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}') = x_1 x_2 x_1' x_2 + (x_1 - x_2)(x_1' - x_2')$$

as we will see, kernel methods exploit the **kernel trick,** whereby they can work directly at the level of kernels **without the need to specify the feature space mapping**

Kernel Methods

Why working with kernels is advantageous?

Assume we work in a data space of **dimension p=3,** but classification requires a feature space of **dimension p=9**

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2\right)^T$$

$$\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{y}) = \sum_{i,j=1}^{3} x_i x_j y_i y_j$$

scales as $O(n^2)$

Kernel Methods

Why working with kernels is advantageous?

Assume we work in a data space of **dimension p=3,** but classification requires a feature space of **dimension p=9**

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2\right)^T$$

$$\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{y}) = \sum_{i,j=1}^{3} x_ix_jy_iy_j$$

scales as O(n²)

The same result can be obtained working with an appropriate **kernel**

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y})^2 = \left(x_1y_1 + x_2y_2 + x_3y_3\right)^2 = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{y})$$

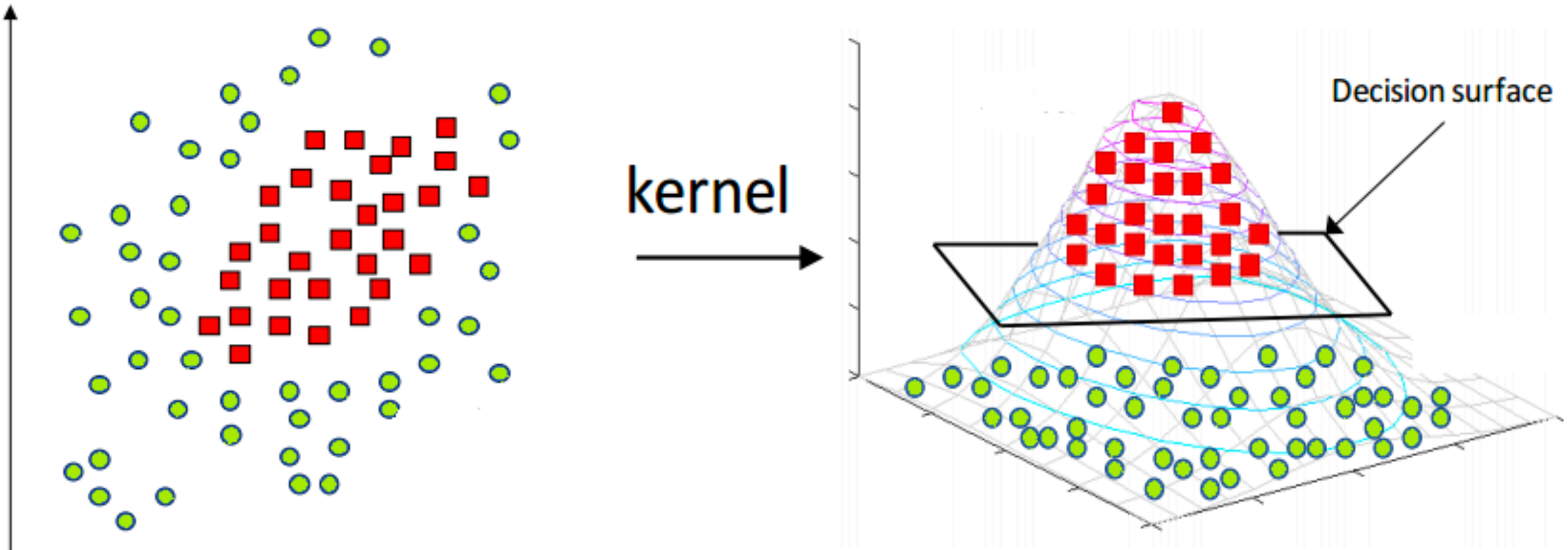
scales as O(n)

efficient and **computationally advantageous** method to transform data to higher dimensions!

Kernel Methods

Why working with kernels is advantageous?

Assume we work in a data space of **dimension p,** but classification requires a feature space of **dimension p' > p**



Data Space (p=2):

linear classification impossible

Feature space (p=3):

linear classification possible!

Kernel Methods

Kernel methods are used in a **wide variety of Machine Learning** applications:

- Pattern analysis
- Clustering
- Dimensional reduction
- Classification
- Anomaly detection

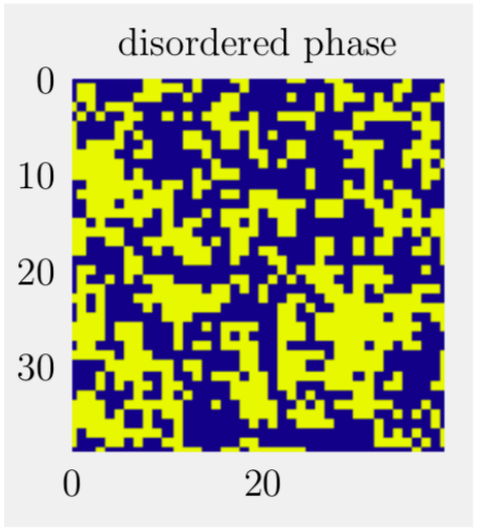
A kernel is **a similarity function:** it measures how similar two inputs are

working with **kernels** offers many advantages over working with **feature vectors,** which in general have infinite dimensionality

Kernel Methods

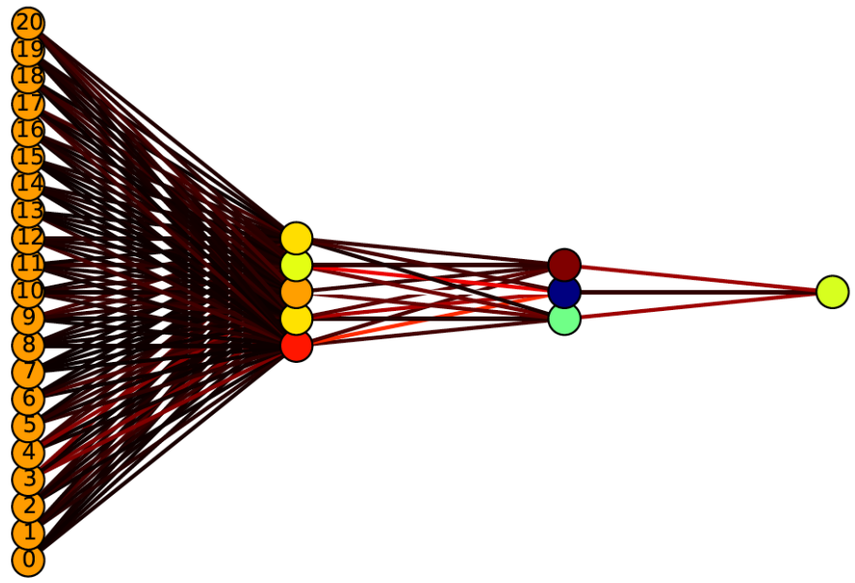
Classifier working on **feature space**

labelled training examples => Feature map => input to classifier



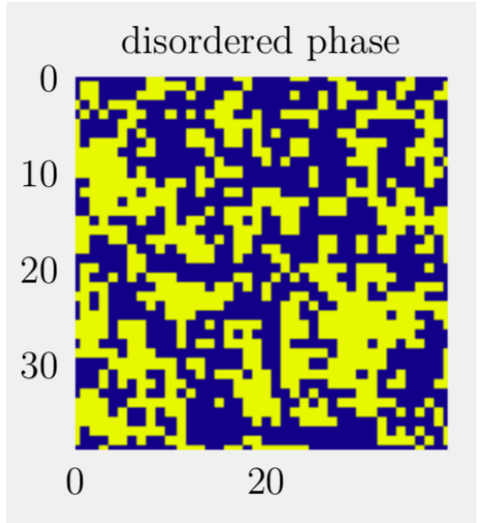
array with spin values

+ label indicating phase



Classifier working with **kernel method**

labelled training examples + kernel (similarity measure) => Classifier



$$k\left(\boldsymbol{x}, \boldsymbol{x}', \alpha, \beta, \ldots\right)$$

optimise hyper-params of the kernel

to achieve good classification

how to select a suitable kernel?

Support Vector Machines

Support Vector Machines

SVM are **kernel methods** popular for classification, regression, and novelty detection

A peculiar feature of SVM is that the model parameters are found by a solution of a convex optimisation problem: any local solution is also a **global optimum**

The advantages of support vector machines include:

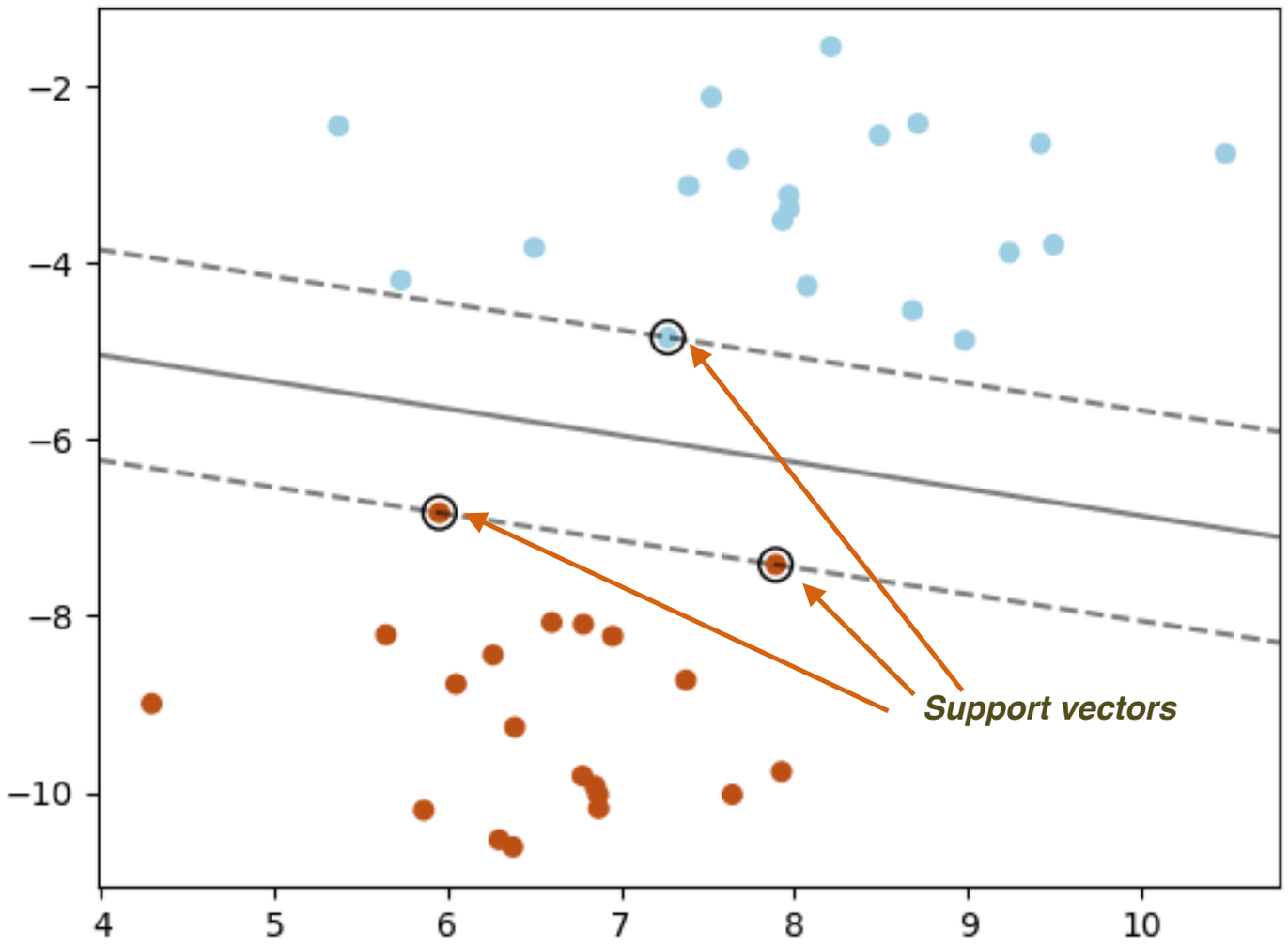
- Effective in **high dimensions.**
- Effective when number of dimensions is greater than number of samples.
- Includes on a **subset of training points** in the decision function (**support vectors**)
- Different **kernels functions** can be used for the figure of merit

kernel methods operate in a high-dimensional,
implicit feature space without computing the coordinates in data space

Support Vector Machines

SVM are ML algorithms popular for classification, regression, and novelty detection

A peculiar feature of SVM is that the model parameters are found by a solution of a convex optimisation problem: any local solution is also a **global optimum**



In SVM classification, the support vectors are samples that lie in the margin boundaries (at least for linearly separable problems)

Support Vector Machines

As other **classification algorithms**, the problem that SVM try to solve is the following. Starting from a training dataset

$$\{\boldsymbol{x}_i^T, t_i\}, \quad i = 1,\ldots,N, \quad t_i \in (-1,1)$$

binary classification

our goal is to find the **model parameters** such that the prediction from

$$\text{sign}(\boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}) + b)$$

linear in model parameters, but non-linear in feature space

is correct for most of the samples. Here $\boldsymbol{\phi}$ is the **feature-space transformation**, where the *feature space* is where our data lives

Example of a feature-space transformation

$\boldsymbol{x}$	$\boldsymbol{\phi}(\boldsymbol{x})$
Height	***Height/Weight***
Weight	***Weight+Height***
Age	***Age * Weight***

Support Vector Machines

the problem then that SVM are aiming to solve is

inverse of margin between the two categories

$$\min_{\theta,b,\xi}\left(\frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}+C\sum_{i=1}^{n}\xi_i\right)$$

subject to the **additional constraint** that

Class label of sample i → $t_i \times \left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x}_i)+b\right) \geq 1-\xi_i$ ← *tolerance (regulator)*

which essentially aims to **maximise the margin between the two categories** while including a penalty when when a sample is misclassified or within the margin boundary, where the team proportional to *C* plays the role of a **regulator**

> SVM are an example of a **constrained optimisation problem,** where one tries to minimise a quadratic function subject to **linear constraints.** They are most efficiently solved by means of the Lagrange multiplier method

Support Vector Machines

In which respect SVD is a **kernel method**? So far it seems to work only in **feature space …**

we can reformulate the problem such that the feature space disappears completely!

many ML models for regression and classification can be reformulated in a **dual representation** where the kernel functions arise naturally

in doing so, we will also show in which respect kernel methods are special in that they **explicitly include the training examples** in the final decision model

Dual representations

many ML models for regression and classification can be reformulated in a **dual representation** where the kernel functions arise naturally

consider a linear regression model with a regularised least-squares error function

$$E_{\mathrm{tr}}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x_n}) - t_n\right)^2 + \frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}$$

regulator

inner product in feature space

sum over training samples

output from training examples

same structure as SVD!

the model parameters are determined **analytically** by requiring the vanishing of the gradient

$$\boldsymbol{\theta} = \sum_{n=1}^{N} a_n\boldsymbol{\phi}(\boldsymbol{x_n}), \quad a_n = -\frac{1}{\lambda}\left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x_n}) - t_n\right)$$

recall that inner products take place in feature space

one can formulate a dual representation of the problem in terms of the **parameter vector**

$$\boldsymbol{a} = \left(a_1, a_2, \ldots, a_N\right)^T \qquad \boldsymbol{t} = \left(t_1, t_2, \ldots, t_N\right)^T$$

Dual representations

with some algebra one can show that the original figure of merit of the model

$$E_{\text{tr}}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right)^2 + \frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}$$

admits a **dual representation** in terms of **kernel function**

model params nor feature space maps do not appear in this formulation

$$E_{\text{tr}}(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{K}\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{K}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$

where ***K*** is known as the **Gram matrix**, an *N x N* symmetric matrix with entries

$$K_{nm} = k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = \boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{\phi}(\boldsymbol{x}_m)$$

matrix in space of input data

given by the **kernel function** evaluated over two of the **input training examples**

the crucial property of this dual representation is that now the predictions for new inputs will explicitly **depend on the training examples**

SVD can be expressed entirely in terms of kernels

Dual representations

the crucial property of this dual representation is that now the predictions for new inputs will explicitly **depend on the training examples**

$$y(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T \left(\boldsymbol{K} + \lambda \boldsymbol{I}_N\right)^{-1} \boldsymbol{t}$$

output of trained model

new input

depends on the N original inputs

$$\boldsymbol{k}(\boldsymbol{x}) = \left(k(\boldsymbol{x}_1, \boldsymbol{x}), k(\boldsymbol{x}_2, \boldsymbol{x}), \ldots\right)$$

in the dual formulation we invert a *N x N* matrix (data space) rather than a *M x M* one (feature space), so this does not appear to be advantageous …

the main benefit is being able to **work directly with kernel functions** and bypass a choice of feature map

this allows one to work in feature spaces of **very high (even infinite) dimensionality**

Support Vector Machines

https://www.youtube.com/watch?v=5zRmhOUjjGY

The kernel trick

recall that the kernel function is constructed from a **feature space mapping**

$$k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{\phi}(\boldsymbol{x}_m) = \sum_{i=1}^{M} \phi_i(\boldsymbol{x}_n)\phi_i(\boldsymbol{x}_m)$$

one can also construct kernel functions directly, provided they can be expressed as above

e.g., for a 2D input space

$$k\left(\boldsymbol{x}, \boldsymbol{z}\right) = \left(\boldsymbol{x}^T \boldsymbol{z}\right)^2 = \left(x_1 z_1 + x_2 z_2\right)^2 = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z})$$

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2\right)$$

note that feature space dimension (M=3) is bigger than input space dimensionality (d=2)

there exist more general methods to construct acceptable kernels e.g. using **basis functions**

a popular choice is Gaussian kernel, built from an **infinite-dimensional feature map**

$$k\left(\boldsymbol{x}, \boldsymbol{z}\right) = \exp\left(-||\boldsymbol{x} - \boldsymbol{z}||^2 / 2\sigma^2\right)$$

The kernel trick

a popular choice is Gaussian kernel, built from an **infinite-dimensional feature map**

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-||\boldsymbol{x} - \boldsymbol{z}||^2 / 2\sigma^2\right)$$

we can demonstrate that this is a **valid kernel** as follows: given two valid kernels

$$k_1(\boldsymbol{x}, \boldsymbol{z}) \ , \quad k_2(\boldsymbol{x}, \boldsymbol{z})$$

we can construct **new valid kernels** from them using the following rules

$$k(\boldsymbol{x}, \boldsymbol{z}) = f(\boldsymbol{x}) k_1(\boldsymbol{x}, \boldsymbol{z}) f(\boldsymbol{z}) \qquad k(\boldsymbol{x}, \boldsymbol{z}) = \exp(k(\boldsymbol{x}, \boldsymbol{z}))$$

so now we can check that the Gaussian kernel is indeed a *bona-fide* kernel by expanding

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\boldsymbol{x}^T \boldsymbol{x} / 2\sigma^2\right) \exp\left(-\boldsymbol{x}^T \boldsymbol{z} / 2\sigma^2\right) \exp\left(-\boldsymbol{z}^T \boldsymbol{z} / 2\sigma^2\right)$$

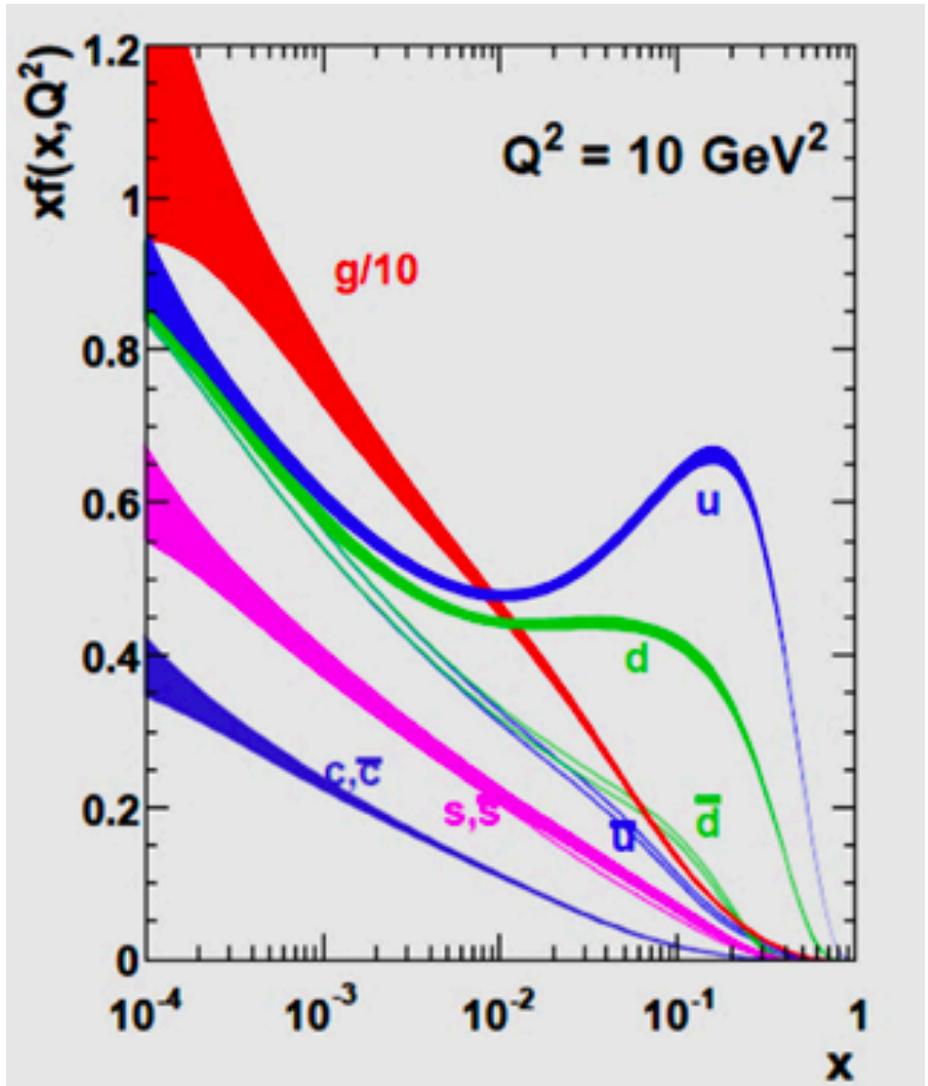
linear kernel

the feature map associated to this kernel has infinite dimensionality!

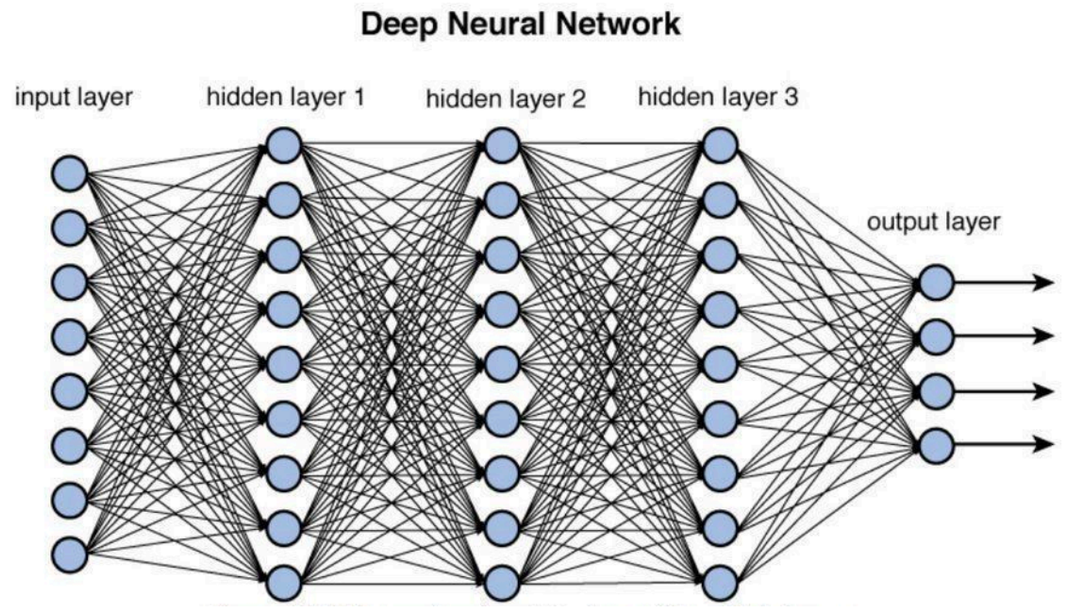
Gaussian Processes

ML for regression

In supervised machine learning, **non-linear regression models** such as **deep neural networks** are used to parametrise the information contained in the data



recall tutorial 2: the gluon PDFs from NNs



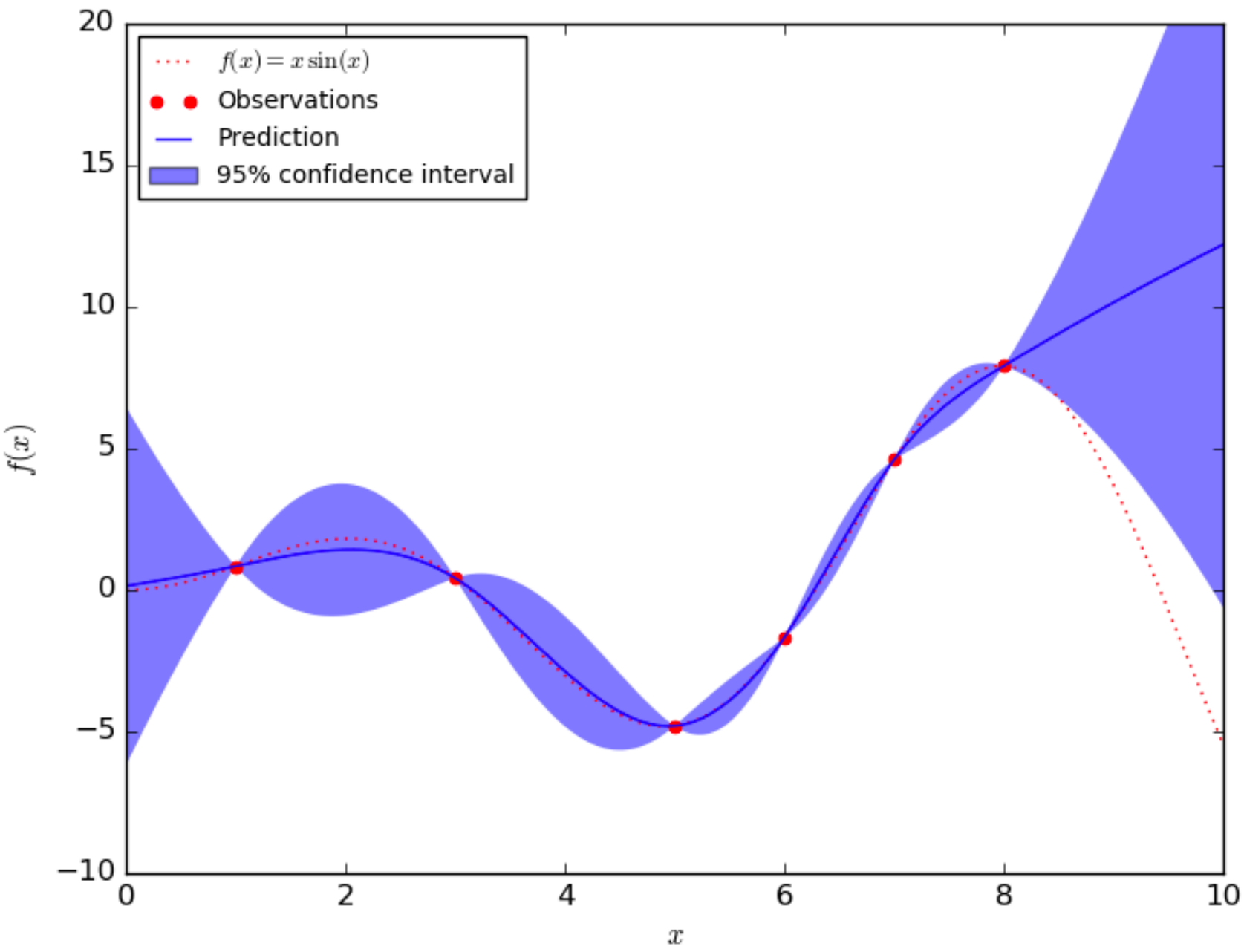
$$E(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i(\boldsymbol{\theta})\right)^2$$

$$\hat{y}_i(\boldsymbol{\theta}) = \mathrm{NN}(\boldsymbol{x}_i, \boldsymbol{\theta})$$

NN output *NN input* *NN parameters*

Gaussian processes

in Gaussian processes we bypass the need of a **parametric classifier/regressor** and work directly in the **space of functions for the (prior) probability distributions**



in regression models one would introduce a parametrisation and infer the model params from

instead we can to avoid parameterisations and express the problem directly at the level of the space of functions/models

Gaussian processes

So far mostly kernel method limited to **non-probabilistic models.** We now extend the kernel methods to **probabilistic discrimination processes**

in Gaussian processes we bypass the need of a **parametric classifier/regressor** and work directly in the **space of functions for the (prior) probability distributions**

we can illustrate the philosophy of the method by revisiting the **linear regression** example

$$y(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^{M} \theta_i \phi_i(\boldsymbol{x})$$

predictor — *model parameters* — *feature space map*

assume that the model parameters have as **prior distribution** an isotropic Gaussian

$$p(\boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{\theta} \,|\, \mathbf{0}, \alpha^{-1}\mathbf{1}\right)$$

for a given choice of the model parameters, we will have a different functional form of the predictor

the probability density for θ induces a probability density in the space of functions y !

Gaussian processes

consider that we are given a set of training examples and we evaluate the model predictions

$$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \quad \boldsymbol{y} \equiv (y(\boldsymbol{x}_1,) \ldots, y(\boldsymbol{x}_N))$$

note this is not the label of the training examples!

from the definition of the linear model we have

sum in latent space

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{\theta} = \sum_{i=1}^{M} \phi_i(\boldsymbol{x}_n)\boldsymbol{\theta}_i$$

and we can now construct the **probability density in the space of functions *y*** by noting that it is Gaussian itself (since θ is also Gaussian) with mean and covariance

$$\mathbb{E}[\boldsymbol{y}] = \boldsymbol{\Phi}\mathbb{E}[\boldsymbol{\theta}] = 0$$

$$\text{cov}[\boldsymbol{y}] = \mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T] = \boldsymbol{\Phi}\mathbb{E}[\boldsymbol{\theta}\boldsymbol{\theta}^T]\boldsymbol{\Phi}^T = \frac{1}{\alpha}\boldsymbol{\Phi}\boldsymbol{\Phi}^T = \boldsymbol{K}$$

the kernel function of the Gaussian process

$$K_{nm} = \frac{1}{\alpha}\boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{\phi}(\boldsymbol{x}_m)$$

Gaussian process are another application of kernel models

Gaussian processes

this model provides a specific example of a Gaussian process, whose **main features** are:

- Defined as **probability distribution in the space of functions**
- This distribution is defined by **second-order statistics**: we only need mean and covariance
- Its covariance is given by the **kernel function**
- A gaussian process depends on a number of **hyperparameters** that should be inferred from data

Gaussian kernel

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-||\boldsymbol{x} - \boldsymbol{z}||^2 / 2\alpha^2\right)$$

Exponential kernel

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\alpha||\boldsymbol{x} - \boldsymbol{z}||\right)$$

these hyperparameters can be inferred by **maximising the log-likelihood** comparing the model predictions with the output labels of the training data

e.g. for Gaussian process regression

log-likelihood

$$\ln p\left(\boldsymbol{t}|\boldsymbol{\alpha}\right) = -\frac{1}{2}\ln|\boldsymbol{C}_N| - \frac{1}{2}\boldsymbol{t}^T\boldsymbol{C}_N^{-1}\boldsymbol{t}$$

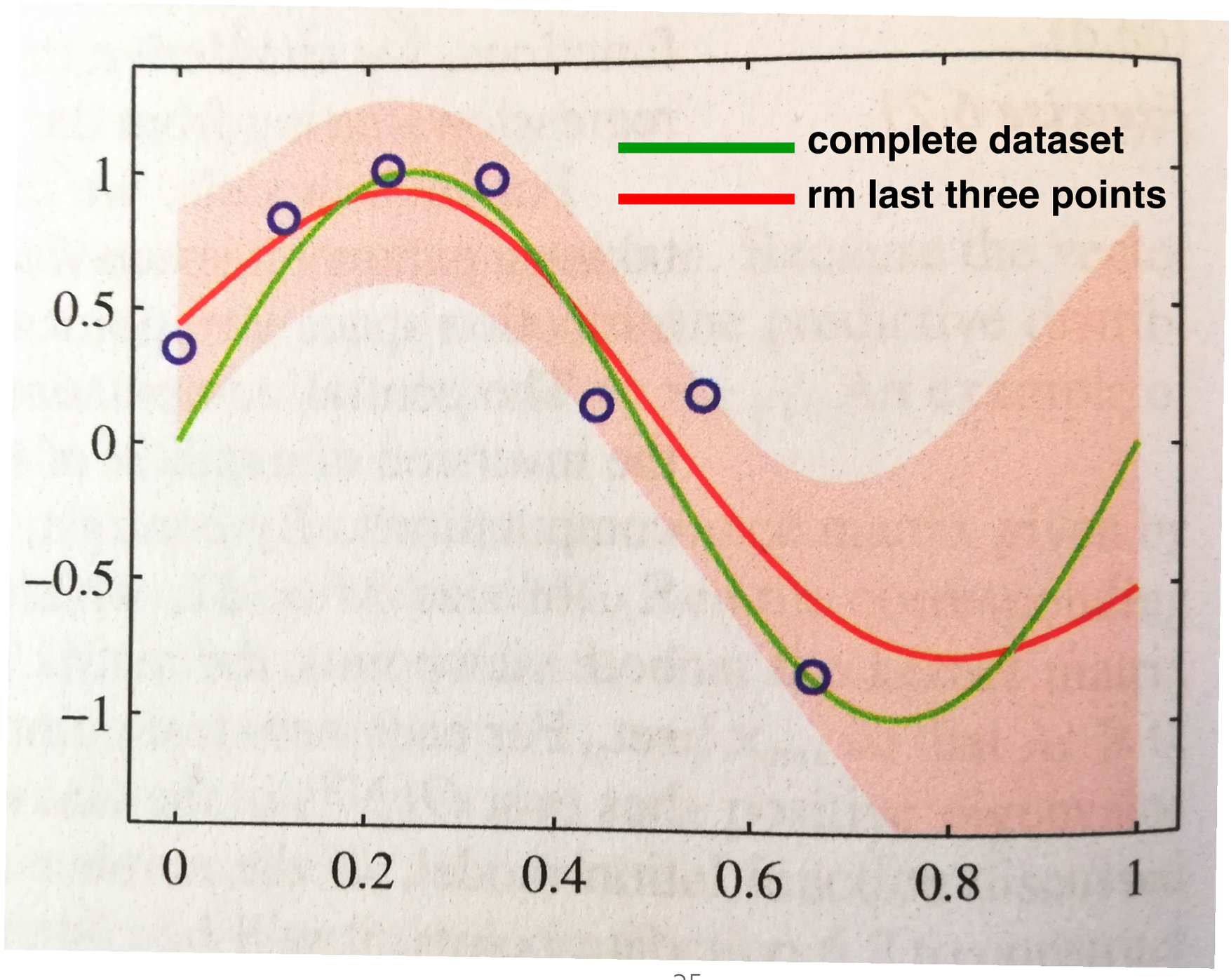
outputs from training dataset ***hyperparameters***

stochastic noise

$$C_{nm} = k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \beta^{-1}\delta_{nm}$$

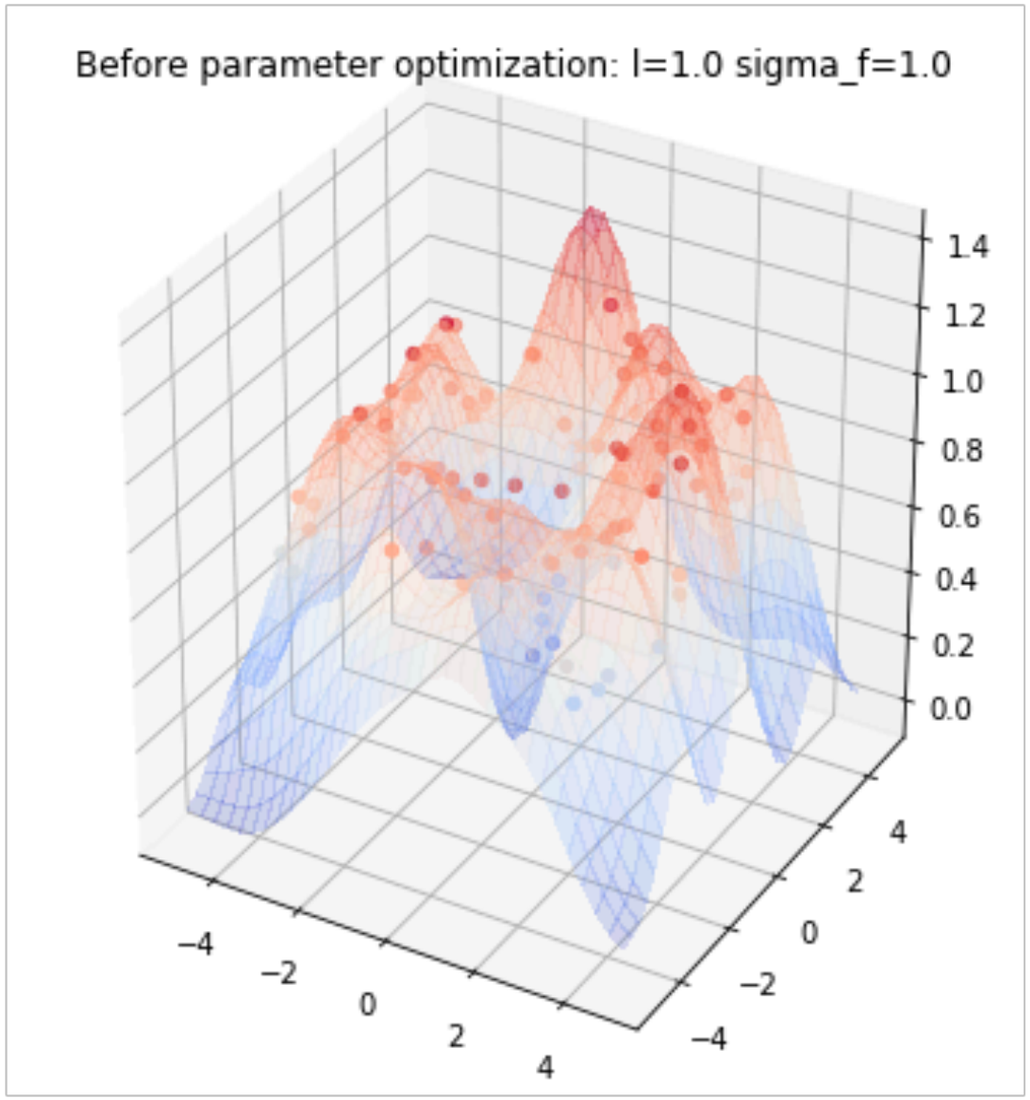
Gaussian processes

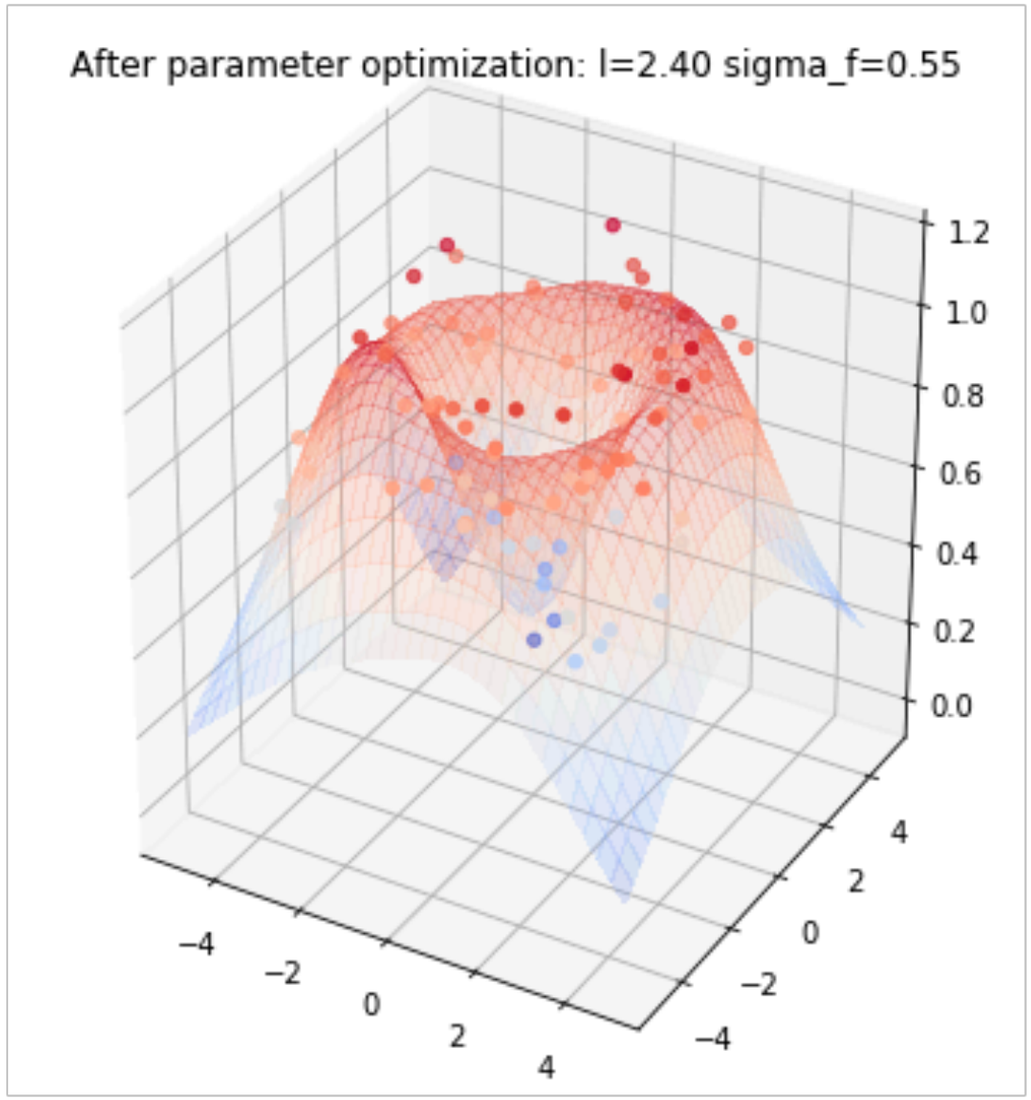
example of Gaussian process applied to a sinusoidal data with with noise added



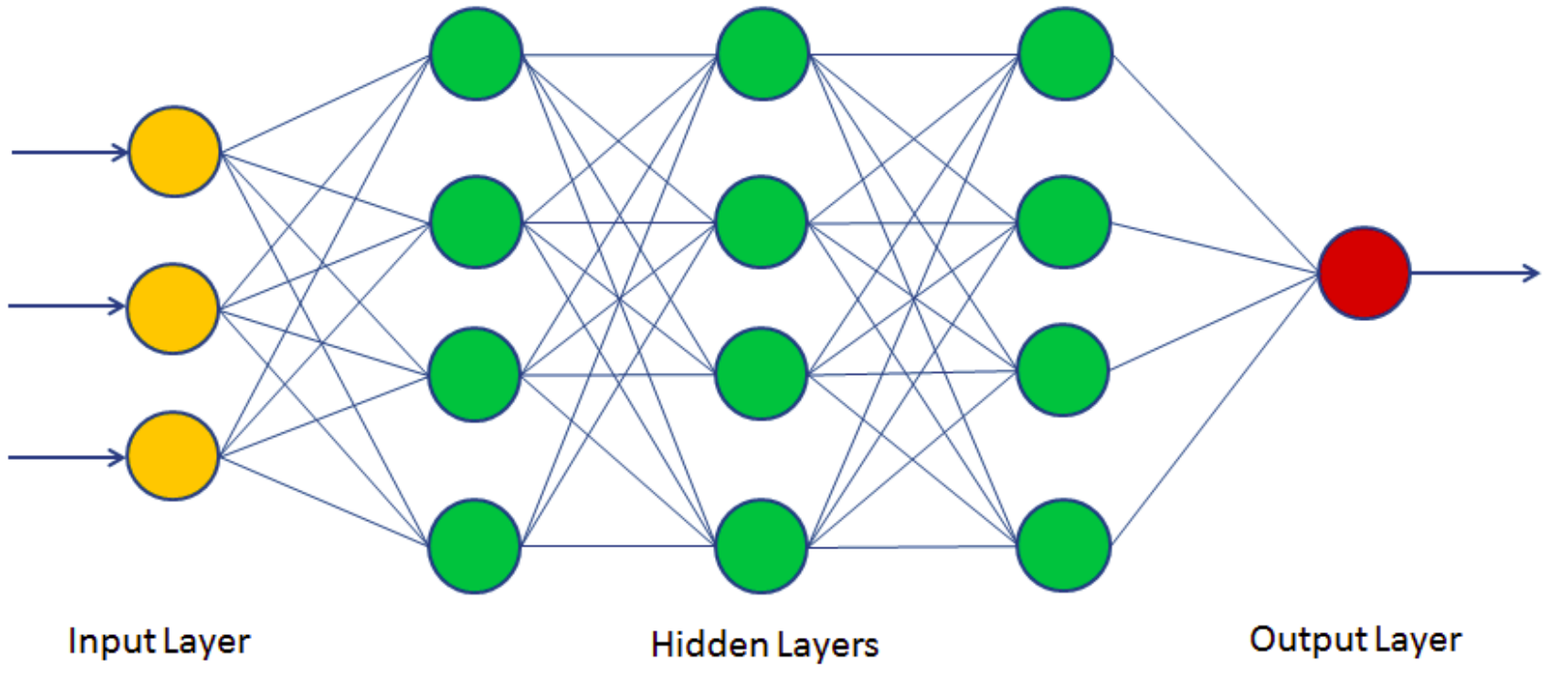
Gaussian processes

hyperparameter optimisation is crucial to construct reliable GP models





Connection with neural nets



we know that for a sufficient number ***M of hidden units*** a neural net can approximate any function

in a Bayesian neural network, the prior distribution over model parameters combined with the network function produced a **prior distribution over the space of network outputs**

$$p(\boldsymbol{\theta}) \; + \; f(\boldsymbol{x}, \boldsymbol{\theta}) \; \rightarrow \; p(y(\boldsymbol{x}))$$

prior prob parameters ***network function*** ***prob in space of network outputs***

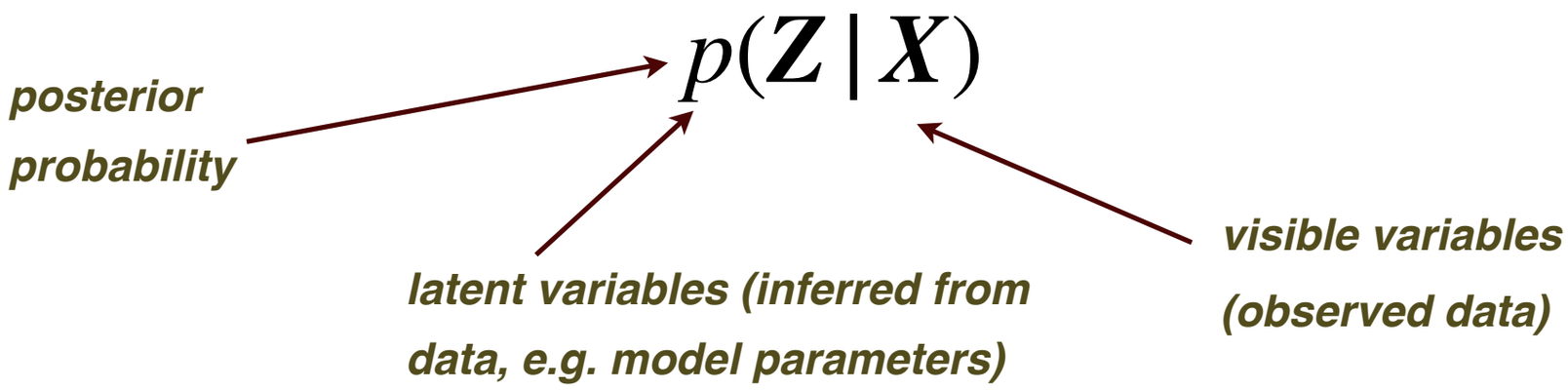
For a broad class of priors $p(\boldsymbol{\theta})$ in **limit M→∞** the NN outputs reproduce a Gaussian process

however in this limit the network outputs become statistical independent!

Variational Inference

Approximate Inference

for many ML models, the central task is the solution of the inference problem, namely determining the **posterior distribution of latent variables** given the observed data



in many cases an exact solution of the **inference problem** might not be possible

- **Latent space too high dimensionality:** too many model parameters
- **Too complex form of posterior probability**: expectation values intractable
- Numerical integrations might be prohibitive, and analytic integration impossible
- **Exponentially large number of configurations** to sum over

we need to rely on **approximate inference schemes** in such cases

Approximate Inference

approximate inference methods

Stochastic (sampling) techniques: Markov Chain Monte Carlo, Bayesian methods ….

guaranteed to give exact result in the limit of infinite computing resources

Deterministic techniques: variational inference, …

based on analytical approximations to the posterior distribution

here we will provide a brief introduction to the ideas **underlying variational inference**

Consider a **fully Bayesian model** where all parameters are given prior distributions

$$\boldsymbol{Z}, \quad \boldsymbol{X}$$

latent variables (incl. model parameters)

visible variables (observed data)

Goal is to find an **approximation** to the posterior distribution and the model evidence

$$p(\boldsymbol{Z} \mid \boldsymbol{X}) \qquad p(\boldsymbol{X})$$

Approximate Inference

as in the discussion of **generative models**, we can express the log marginal probability as

$$\ln p(\boldsymbol{X}) = \mathscr{L}(q) + \mathrm{KL}(q||p)$$

$$\mathscr{L}(q) = \int q(\mathbf{Z}) \ln \left(\frac{p(\mathbf{X}, \mathbf{Z})}{q(\mathbf{Z})} \right) d\mathbf{Z} ,$$

$$\mathrm{KL}(q||p) = -\int q(\mathbf{Z}) \ln \left(\frac{p(\mathbf{X}|\mathbf{Z})}{q(\mathbf{Z})} \right) d\mathbf{Z} ,$$

since in general the true solution will be too complicated, we will restrict the possible forms of the approximate solution. For example, one can make a choice of functional form

$$q(\mathbf{Z}|\boldsymbol{\theta})$$

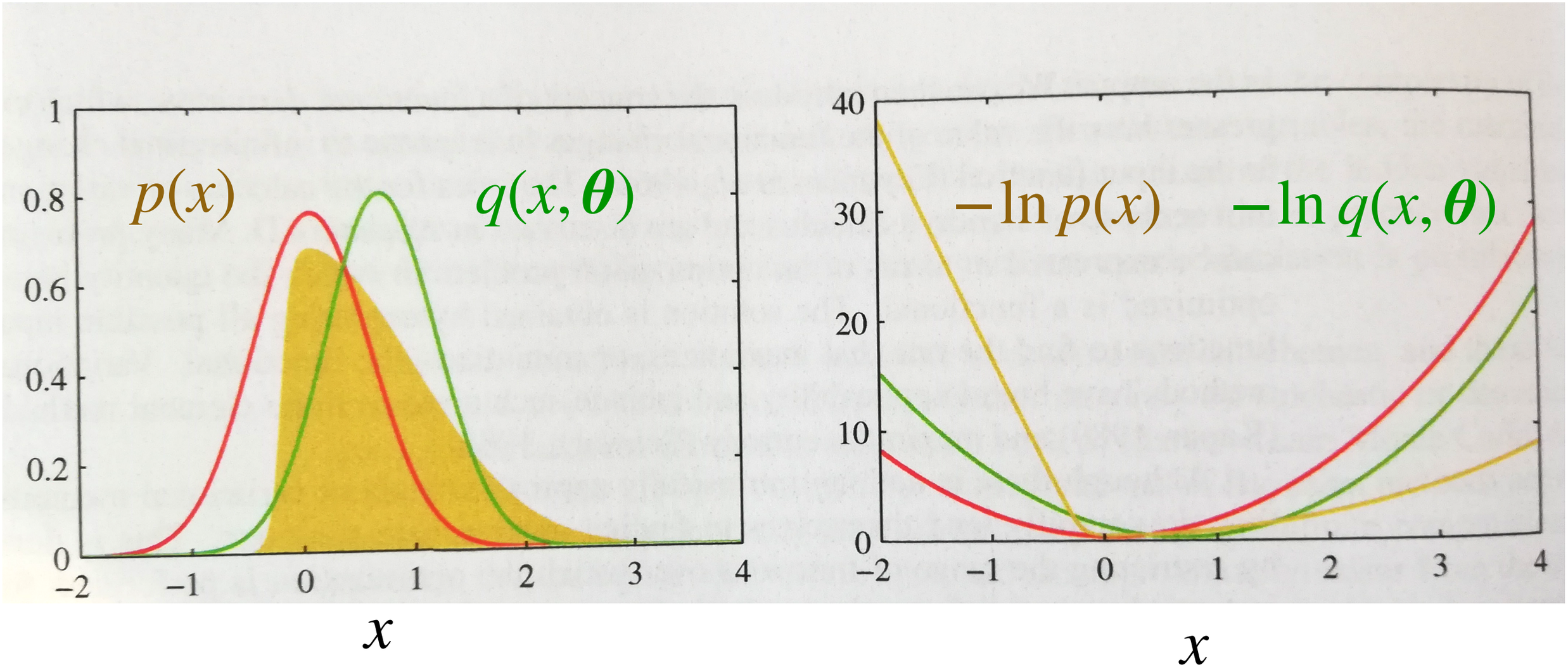
Optimal values of parameters can be found from non-linear optimisation methods by minimising the KL divergence

parameters e.g mean and variance of gaussian

Approximate Inference

$$q(\mathbf{Z}\,|\,\boldsymbol{\theta}) \longrightarrow \boldsymbol{\theta} = (\bar{x}, \sigma)$$

example assuming a gaussian for q(Z)



the more complex the model, the better the approximation to the true posterior probability