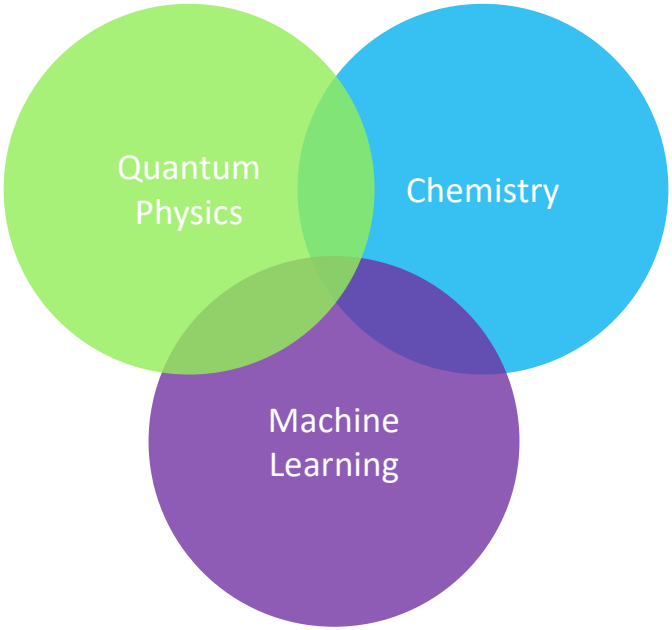
Interplay of Machine Learning and Quantum Theory

Roman Krems

University of British Columbia, Vancouver, Canada



https://www.chem.ubc.ca/roman-krems

August 24, 2021

We will discuss the following:

Part I:

- Hilbert Spaces
- Reproducing Kernel Hilbert Spaces (RKHS)
- Solving regularization problems with RKHS
  - kernel SVM
  - kernel ridge regression
  - Gaussian process regression

Part II:

- Bayesian optimization for inverse quantum problems
- Model selection metrics
  - Validation error, C_P, AIK, BIC
- How to design the best kernel?
- Accelerating Bayesian optimization by kernel improvement

Part III:

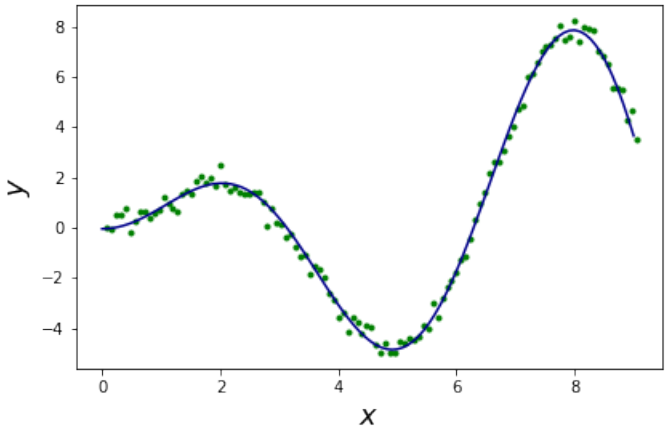
- Gaussian process regression for building accurate PES for large molecules
- Extrapolation of observables with Gaussian processes
- Quantum machine learning
- How to build the best quantum kernels

Part I

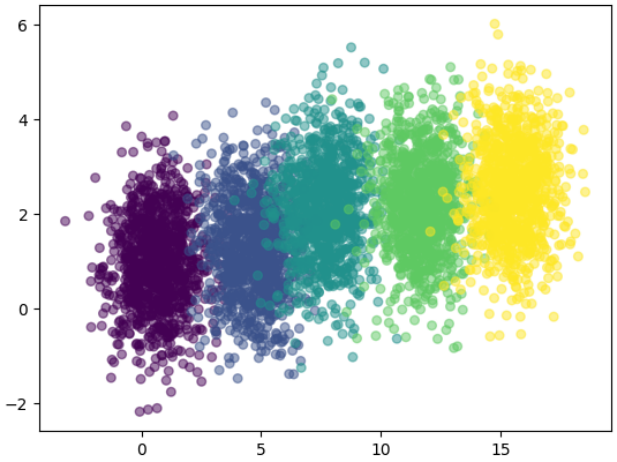
Submit questions about Part I by the end of the day to
rkrems@chem.ubc.ca
I will do my best to address these questions tomorrow

Preliminaries...

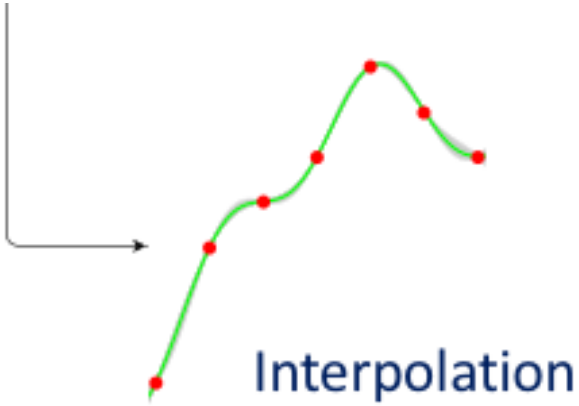
Regression:



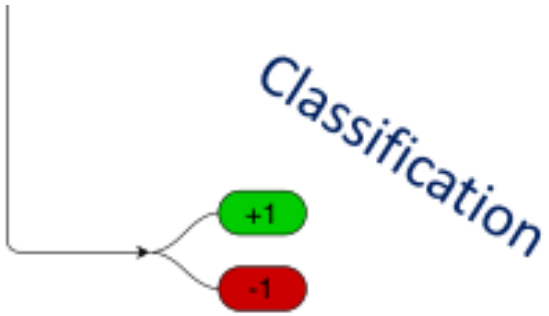
Classification:



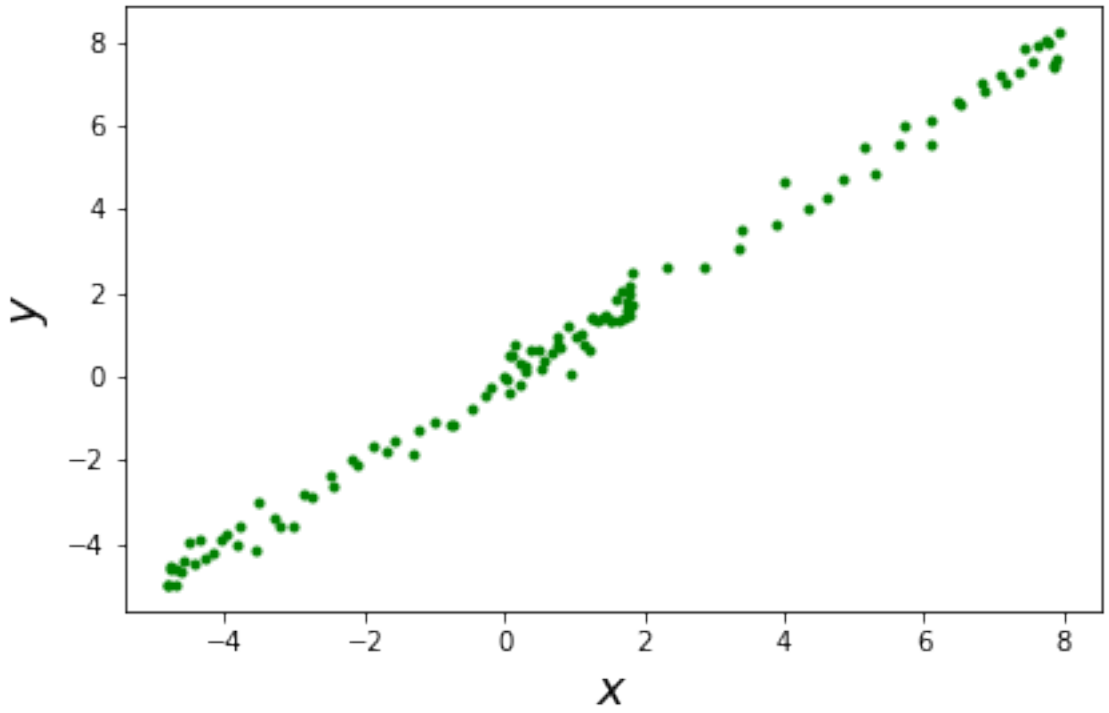
Model



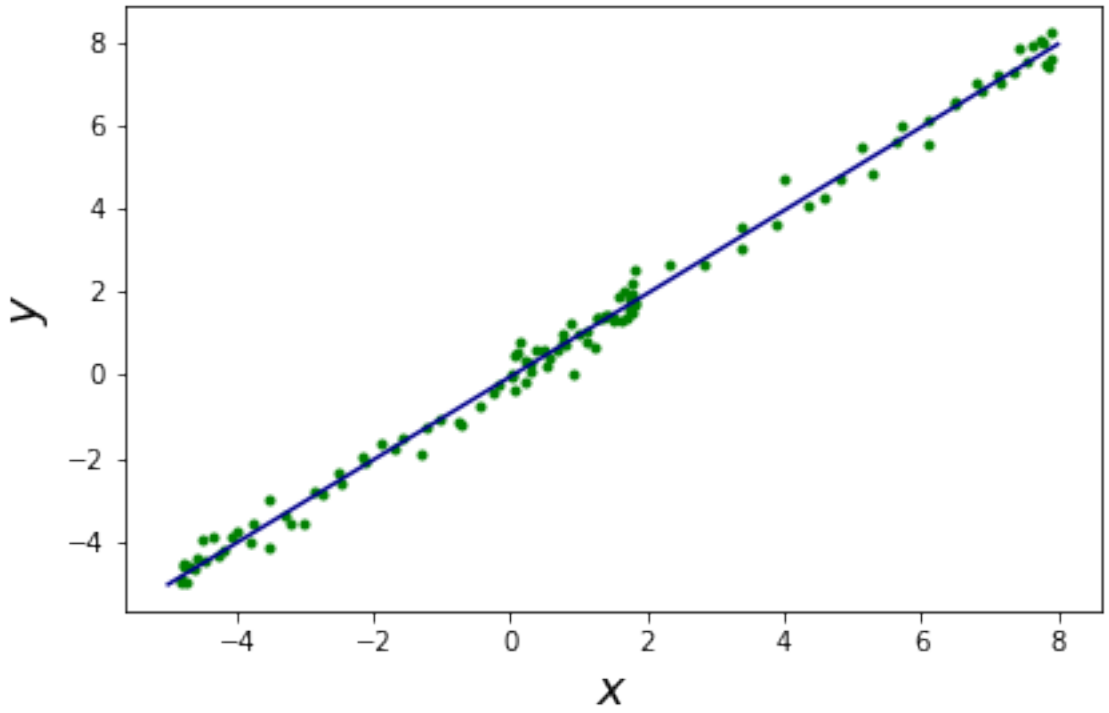
Model



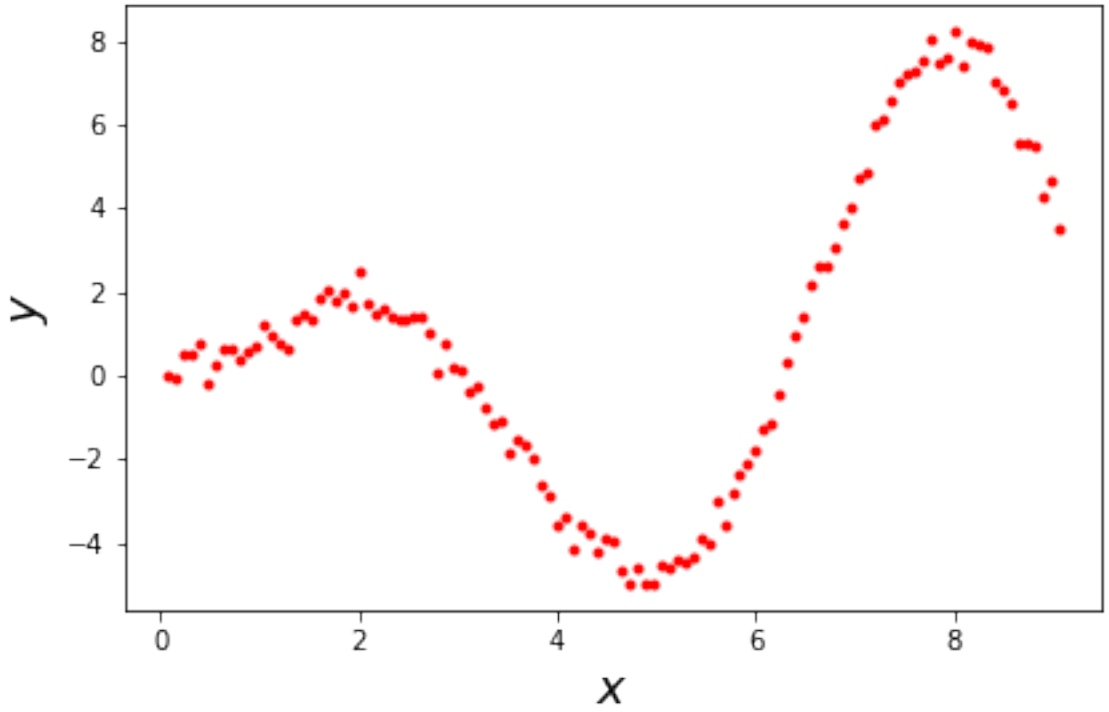
Machine Learning



Machine Learning

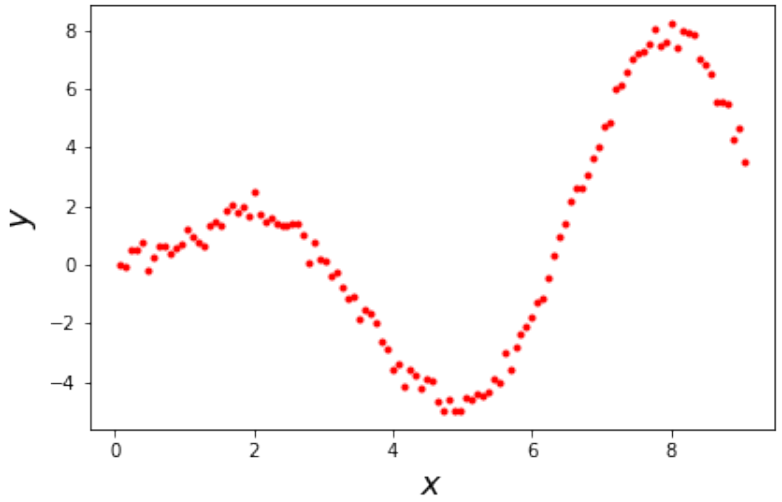


The reality is (most often) non-linear

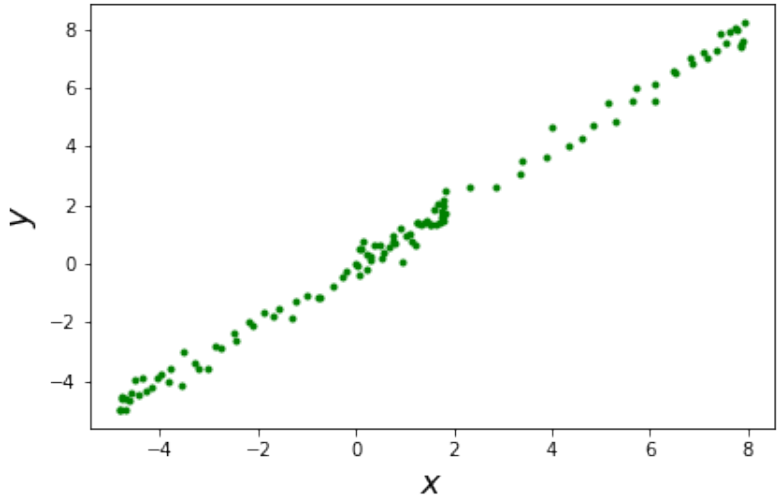


... but it can be made linear...

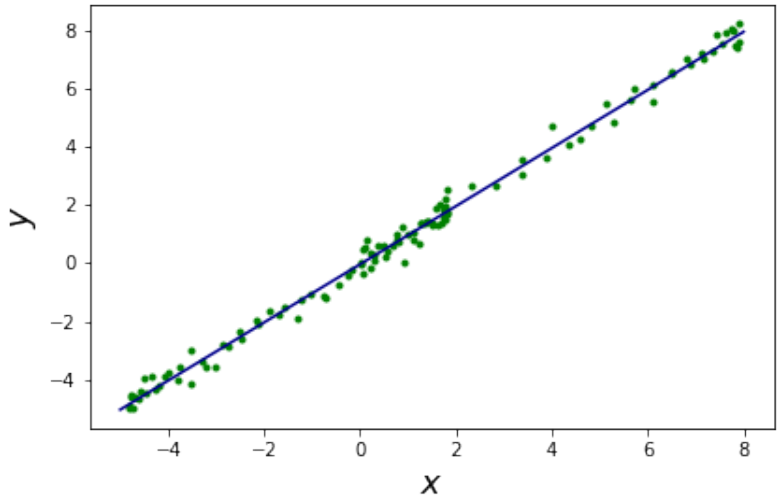
The change $x \to \Phi(x)$ with $\Phi(x) = x \sin(x)$, transforms



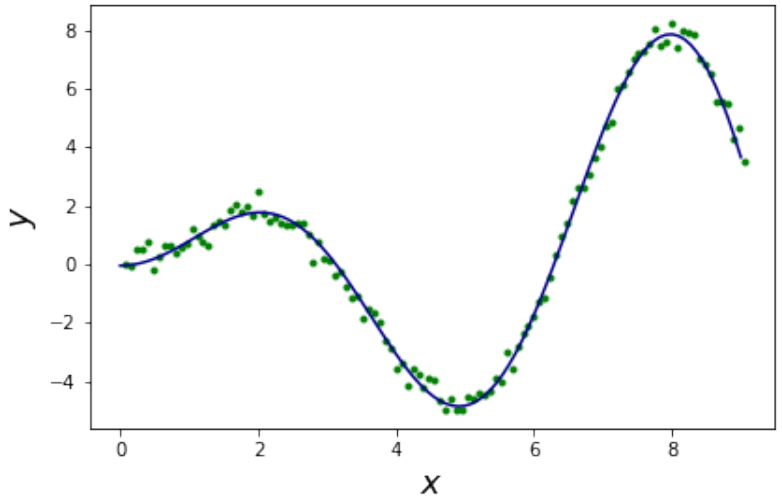
into



Performing regression



and transforming back, yields:



If only we knew $\Phi(x)$ for every problem ...

Some ML methods (e.g., Neural Networks) aim to determine $\Phi(x)$

Others (e.g., kernel methods) bypass this problem ...

... by using the kernel trick $\Leftarrow$ thanks to Hilbert

Kernel trick:

$\Rightarrow$ Avoids the need to determine $\Phi(x)$ explicitly
$\Rightarrow$ Writes all models in terms of inner products $\langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}$

What do we call a kernel?

What do we call a kernel?

In general, kernels are weights, whether in a sum or an integral.

For example,...

$$g(x) = \sum_{i=1}^{n} K(x, x_i) y(x_i)$$

$$g(x) = \int_a^b K(x, y) f(y) dy$$

Pay attention to these examples... they will be useful for us today

$$g(x) = \sum_{i=1}^{n} K(x, x_i) y(x_i)$$

If $y(x_i)$ is an observation at x_i, then

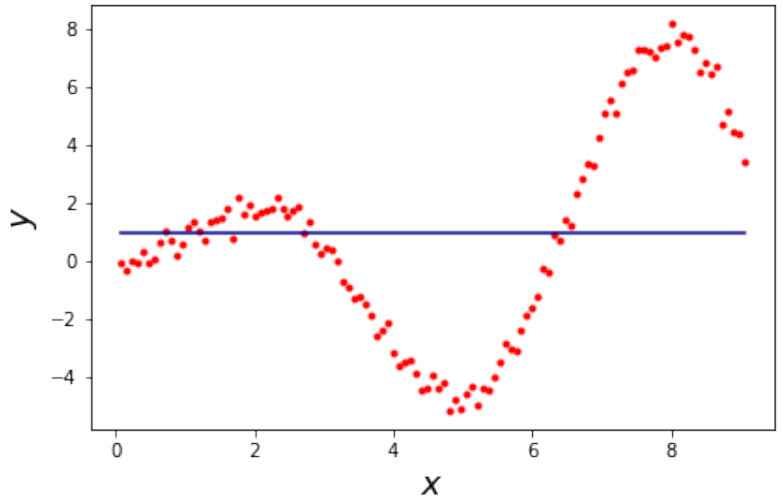
$$\hat{f}(x) = \sum_{i=1}^{n} K(x, x_i) \cdot \{\text{Observation at } x_i\}$$

is a kernel smoother.

Let's take a look at this in a little more detail...

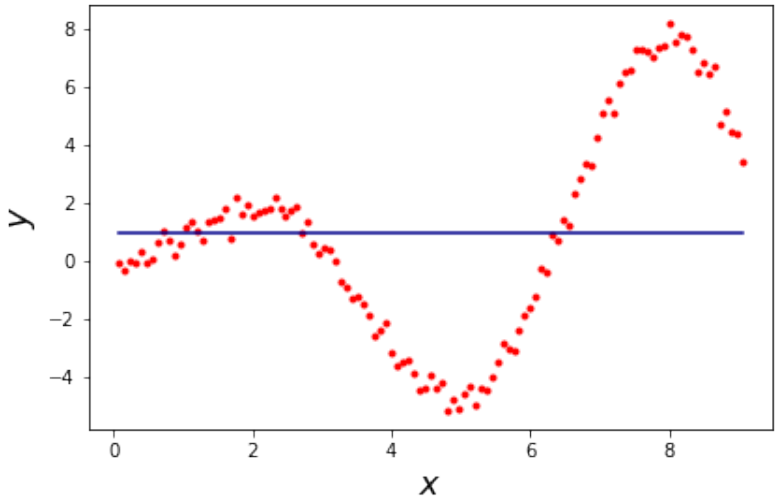
The simplest imaginable ML model:

$$\hat{f} = \frac{1}{n} \sum_{i}^{n} y_i$$



The simplest imaginable ML model:

$$\hat{f} = \frac{1}{n} \sum_{i}^{n} y_i$$

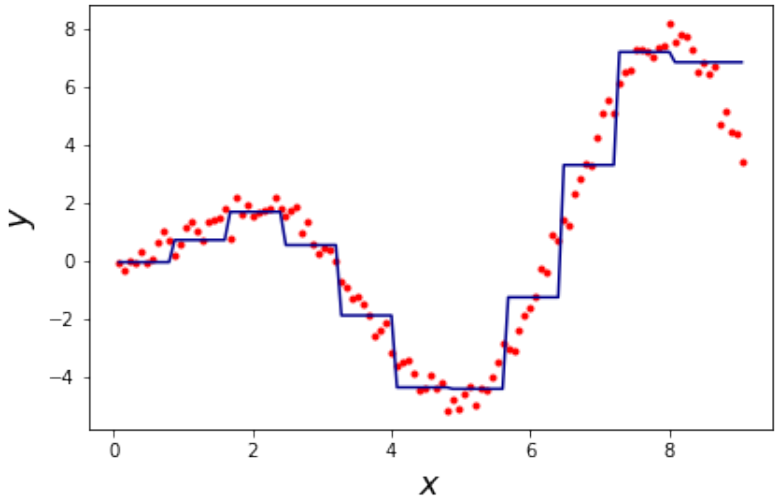


Let's take smaller chunks of data (e.g. 10 points at a time) and average:

$$\hat{f}(x) = \frac{1}{10} \sum_{i=1}^{10} y_i$$

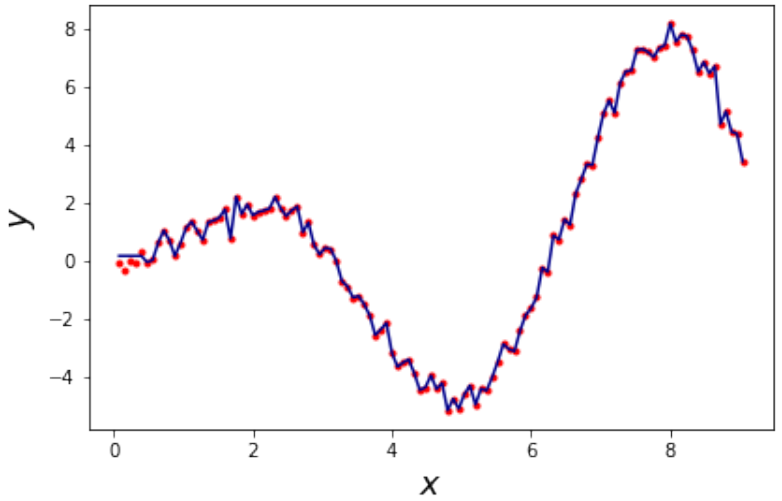
for i sampling points
in bin Δ_j with $x \in \Delta_j$
and $j = 1, \ldots, n/10 - 1$

Moving average:



We can perform such averaging for every point in x:

$$\hat{f}(x) = \sum_{i=1}^{n} h_i(x) y_i$$



$$h_i(x) = \begin{cases} \frac{1}{10} \text{ for 10 points near } x \\ 0 \text{ otherwise} \end{cases}$$

We will now replace $h_i(x)$ with smooth functions of x.

We will use functions that depend only on the distance from point x_i, i.e. $h_i(x) \Rightarrow \varphi(|x - x_i|)$.

Such functions are called radial basis functions (RBF).

We will use renormalized Gaussian functions for RBF.

If $h_i(x)$ are replaced with smooth functions, such as

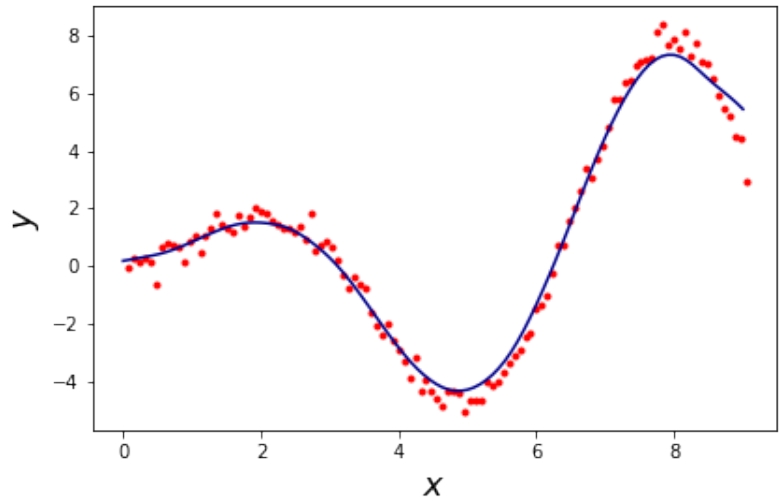
$$h_i(x) = k(x_i, x)/A(x) \quad \text{with} \quad A(x) = \sum_{i=1}^{n} k(x_i, x)$$

$$k(x_i, x) = \frac{1}{\sqrt{2\pi l}} e^{-(x-x_i)^2/2l} \quad \leftarrow \text{Gaussian functions}$$

then

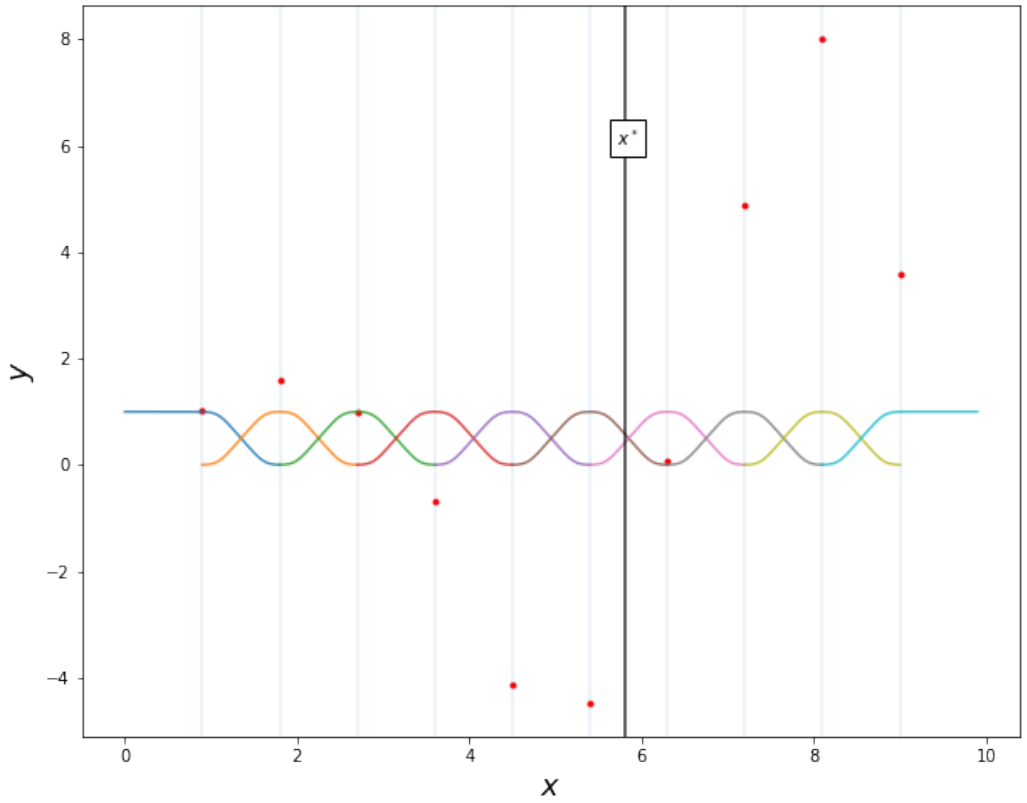
$$\hat{f}(x) = \sum_{i=1}^{n} h_i(x) y_i$$

becomes smooth:



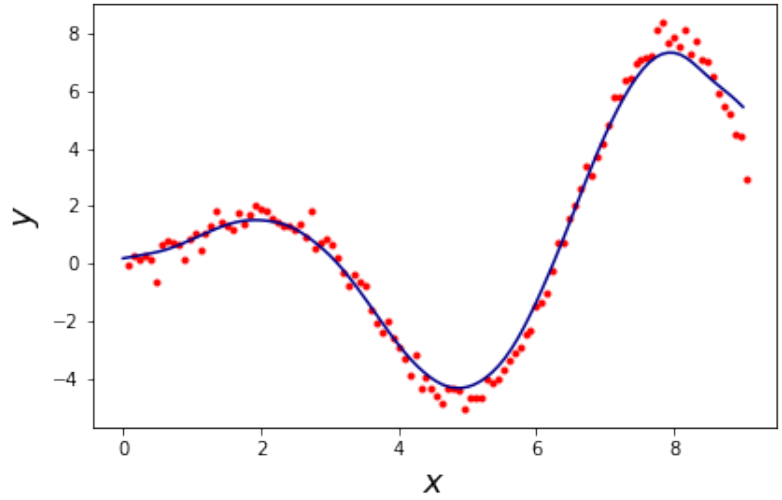
The last plot is simply the sum over y_i with weights given by the corresponding Radial Basis Functions.

The renormalized RBFs look like this:



To evaluate the model at x^*, we sum over y_i multiplied by the value of the corresponding RBF.

$$\hat{f}(x) = \sum_{i=1}^{n} h_i(x) y_i = \sum_{i=1}^{n} K(x, x_i) y_i$$



Remember this simple equation... we'll come back to it later..

Another example – integral transforms:

$$g(x) = \int_a^b K(x, y) f(y) dy$$

where $K(x, y)$ is the nucleus – or kernel – of the integral transform.

In path-integral formulation of quantum mechanics:

$$\psi(x, t) = \int_{-\infty}^{\infty} K(x, t; x', t') \psi(x', t') dx'$$

where the kernel is given by the Green's function of the Schrödinger equation.

...back to Hilbert...

Hilbert showed that the integrals over two real-valued, square-integrable functions

$$\text{inner product} \rightarrow \qquad \langle f, g \rangle = \int_a^b f(x) g(x) dx$$

have the same properties as a scalar product of two vectors in a Euclidean space.

For orthogonal functions, this means:

$$\langle f, g \rangle = 0$$

Why is this important?

The scalar product in a Euclidean space can be used to define the angle between two vectors:

$$\cos\theta = \frac{\boldsymbol{a}\cdot\boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}|}$$

So the scalar product determines the relationship between $\boldsymbol{a}$ and $\boldsymbol{b}$.

Hilbert's inner products can be used to define the relationship between $f(x)$ and $g(x)$

This can be used to define a Hilbert space.

What is a space?

A space is a set of elements with some rule(s) that provides the relationship between the elements

Hilbert's inner products provide the rule for a set of functions giving rise to a Hilbert space

Consider an integral transform:

$$f(x) = \int_a^b K(x,y) f(y) dy$$

with the kernel $K(x,y)$ that is symmetric in x and y.

Hilbert's discovery implies that the kernel can be written as

$$K(x,y) = \sum_{n=1}^{\infty} \lambda_n \phi_n(x) \phi_n(y)$$

where $\phi_n(x)$ are orthogonal functions.

If $\phi_n(x)$ are ortho-normal, $\lambda_n = 1$.

To see this, consider $f(y) = \phi_k(y)$:

$$\int_a^b K(x,y) f(y) dy = \sum_n \lambda_n \phi_n(x) \int_a^b \phi_n(y) \phi_k(y) dy = \lambda_k \phi_k(x)$$

For an arbitrary $f(y)$:

$$\int_a^b K(x,y) f(y) dy = \sum_n \lambda_n \phi_n(x) \int_a^b \phi_n(y) f(y) dy =$$
$$= \sum_n \lambda_n a_n \phi_n(x) = f(x)$$

The final equality holds provided $\lambda_n = 1 \; \forall \; n$.

Thus, we have

$$K(x,y) = \sum_{n=1}^{\infty} \phi_n(x)\phi_n(y)$$

This is a very special function.. why?

Let's index it by x and view it as a function of y:

$$K_x(y) = \sum_n a_{x,n}\phi_n(y)$$

Viewed this way K_x is a vector in the Hilbert space with ϕ_n as axes.

At the same time, we began with:

$$f(x) = \int_a^b K(x,y)f(y)dy = \langle f, K_x \rangle$$

Thus, any function in this space can be written as

$$f(x) = \langle f, K_x \rangle$$

and K_x is called the reproducing kernel of this Hilbert space

... because it reproduces any function in this space by the inner product above

$K(x, y)$ is the kernel function that defines a reproducing kernel Hilbert space

Reproducing kernel Hilbert space = RKHS

Thus, any function can be written as

$$f(x) = \langle f, K_x \rangle$$

What if $f(x) = K_y(x)$?

Thus, any function can be written as

$$f(x) = \langle f, K_x \rangle$$

What if $f(x) = K_y(x)$?

$$K_y(x) = \langle K_y, K_x \rangle$$

or

$$K(x, y) = \langle K_y, K_x \rangle$$

Thus, the kernel function is an inner product of K_x and K_y!

A brief summary of the above:

- A Hilbert space is defined by its elements and an inner product.

$$\text{inner product} \rightarrow \quad \langle f, g \rangle = \int_a^b f(x)g(x)dx \quad \rightarrow L^2 \text{ Hilbert space}$$

- The kernel function $K(x, y)$ defines a unique RKHS and vice versa.

$$\Rightarrow \text{Moore} - \text{Aronszajn theorem}$$

- Because $K(x, y)$ is an inner product, it must be a positive definite function.

Question:

- Can any positive definite function $K : \chi \times \chi \rightarrow \mathbb{R}$ be a kernel function?

Yes, if the inner product is appropriately redefined.

The inner product from above can be written as

$$\langle f, g \rangle = \int_a^b f(x)g(x)dx = \sum_{n=1}^{\infty} \langle f, \phi_n \rangle \langle g, \phi_n \rangle$$

Now, consider an arbitrary, positive-definite function $K(x, y)$ with the eigenvalue equation:

$$\int_a^b K(x, y)\phi_n(y)dy = \lambda_n \phi_n(x) \quad \text{with} \quad \lambda_n \geq 0$$

The function $K(x, y)$ can still be written as

$$K(x, y) = \sum_{n=1}^{\infty} \lambda_n \phi_n(x)\phi_n(y)$$

but λ_n are no longer equal to 1.

Is $K_x(y) := K(x, y)$ still a reproducing kernel?

The answer is yes, if we redefine the inner product of the Hilbert space as

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{n=1}^{\infty} \frac{\langle f, \phi_n \rangle \langle g, \phi_n \rangle}{\lambda_n}$$

In this case,

$$\langle f, K_x \rangle_{\mathcal{H}} = \sum_{n=1}^{\infty} \frac{\langle f, \phi_n \rangle \langle K_x, \phi_n \rangle}{\lambda_n} = \sum_{n} \langle f, \phi_n \rangle \phi_n(x) = f(x)$$

Thus, a choice of a real, positive definite function $K(x, y)$ defines the RKHS by defining the basis and the inner product of the Hilbert space through the eigenvalue equation:

$$\int_a^b K(x, y) \phi_n(y) dy = \lambda_n \phi_n(x) \quad \text{with} \quad \lambda_n \geq 0$$

How is this all related to machine learning?

Remember our feature map:

$$\boldsymbol{x} \rightarrow \Phi(\boldsymbol{x})$$

We will define the feature map Φ so that

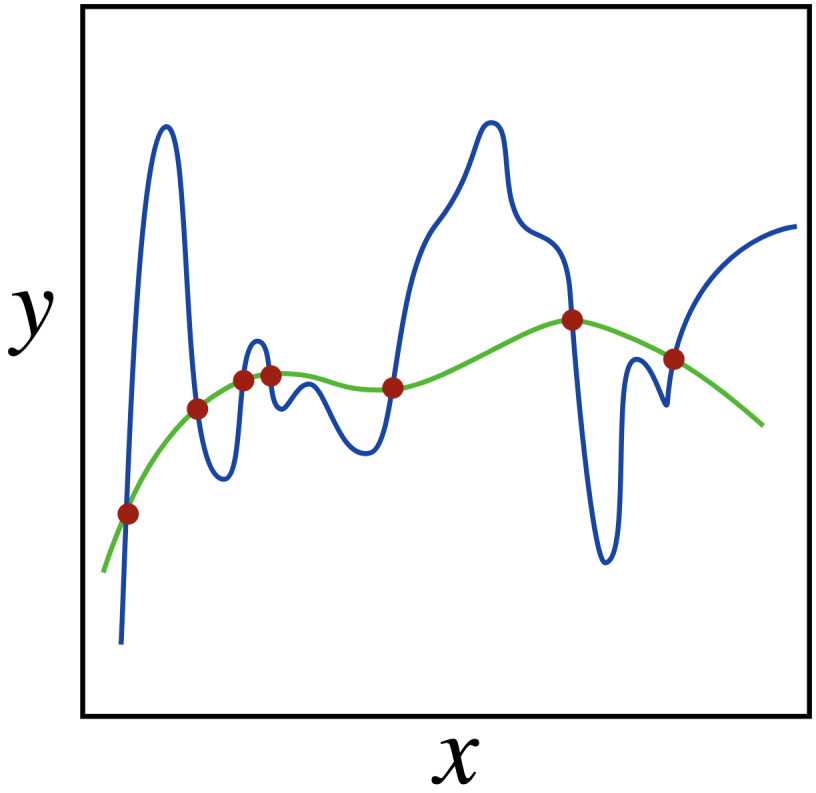
$$\langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle = K(\boldsymbol{x}, \boldsymbol{x}')$$

is a kernel function for a Hilbert space, in which our data are linear (or linearly separable).

What does this give us?

Regularization problems

Regularization



Which model?

How to chose the better one?

Figure source: By Nicoguaro – Own work, CC BY 4.0,

https://commons.wikimedia.org/w/index.php?curid=46259145

How do we train a ML model?

In general we want to minimize the distance between the data and the fit.

An obvious choice is to minimize this:

$$\sum_{i}^{n} (y_i - f(\boldsymbol{x}_i))^2 \quad \text{where } y_i \text{ are observations}$$

But is this a good choice?

Imagine replacing linear functions in the model $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\beta}$ with some other functions of arbitrary complexity:

$$\boldsymbol{x}_i \to \boldsymbol{\varphi}(\boldsymbol{x}_i) = [\varphi_1(\boldsymbol{x}_i), \varphi_2(\boldsymbol{x}_i), \cdots, \varphi_N(\boldsymbol{x}_i)]^{\mathrm{T}}$$

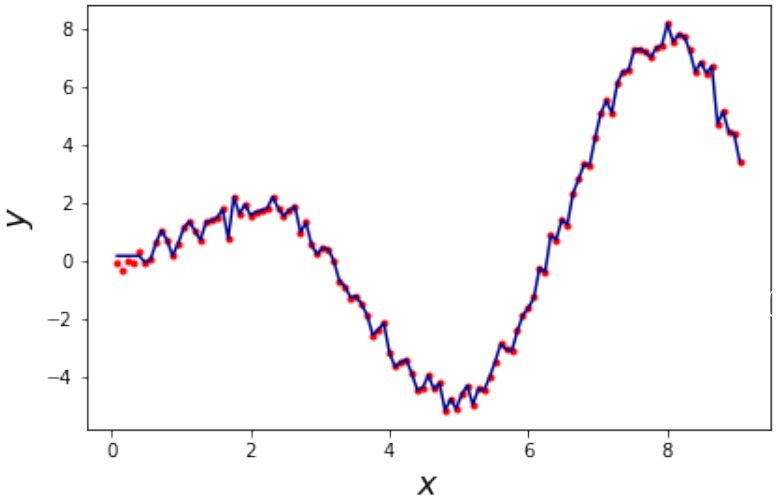
If no bounds are placed on the complexity of the functions φ_i, minimization of

$$\text{Loss function} = \sum_i^n \left(y_i - \boldsymbol{\varphi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\beta}\right)^2$$

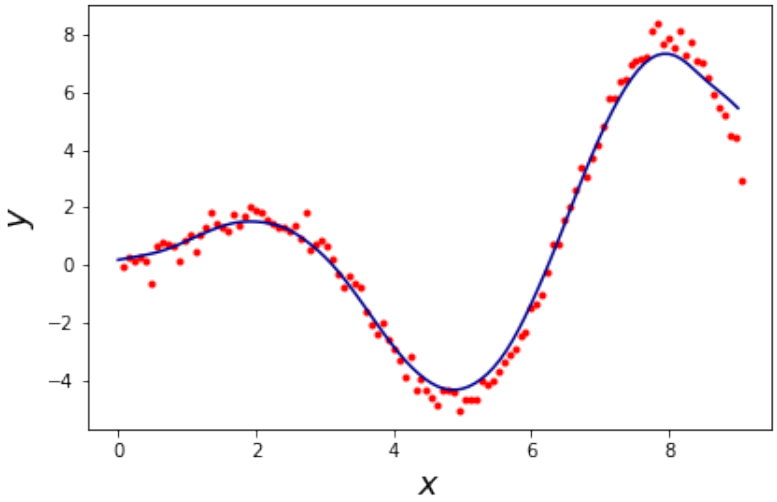
will lead to zero loss function.

The model will fit the noise, which is **overfitting**.

Regularization (aims to) prevents this.



Without regularization



With regularization

How to regularize ML models?

$$\text{Loss} = \sum_{i}^{n} (y_i - f(\boldsymbol{x}_i))^2 + \lambda \langle f, f \rangle_{\mathcal{H}} \quad \leftarrow \quad \text{Tikhonov regularization}$$

This particular regularization is used in ridge regression

How do we find the function $f(\boldsymbol{x})$ that minimizes this loss function?

We already know we can write this function (as any function in this Hilbert space) as

$$f(\boldsymbol{x}) = \sum_{m=1}^{\infty} a_m K(\boldsymbol{x}, \boldsymbol{x}_m)$$

But this sum extends to infinity, so it's not very useful...

How do we find the function $f(\boldsymbol{x})$ that minimizes this loss function?

We already know we can write this function (as any function in this Hilbert space) as

$$f(\boldsymbol{x}) = \sum_{m=1}^{\infty} a_m K(\boldsymbol{x}, \boldsymbol{x}_m)$$

But this sum extends to infinity, so it's not very useful...

However, if we are interested in a specific function $f(\boldsymbol{x})$ that minimizes the loss function on the previous page, we can write

$$f(\boldsymbol{x}) = \sum_{m=1}^{n} \alpha_m K(\boldsymbol{x}, \boldsymbol{x}_m)$$

where the sum runs over the finite number of training points.

(This result is one of what is known as the Representer Theorems)

Let me remind you that the ML problem minimizes

$$\sum_{i}^{n} (y_i - f(\boldsymbol{x}_i))^2 + \lambda \langle f, f \rangle_{\mathcal{H}}$$

and

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{n=1}^{\infty} \frac{\langle f, \phi_n \rangle \langle g, \phi_n \rangle}{\lambda_n}$$

where λ_i are the eigenvalues of the kernel.

What if some of the eigenvalues λ_n are zero?

(This will be important for quantum machine learning .. stay tuned)

Note that we now have two Hilbert spaces: the L^2 and RKHS.

The basis states ϕ_i are orthonormal in L^2.

The basis states ϕ_i corresponding to $\lambda_i \neq 0$ span the RKHS.

Consider a function

$$f(x) = \sum_{i=1}^{\infty} a_i \phi_i(x)$$

In L^2, it has the norm:

$$||f|| = \sum_{i=1}^{\infty} |a_i|^2$$

In RKHS, it has the norm:

$$||f||_{\mathcal{H}} = \sum_{i=1}^{\text{dim of RKHS}} |a_i|^2/\lambda_i$$

If the function $f(x)$ is in RKHS, the decomposition

$$f(x) = \sum_{i=1}^{\infty} a_i \phi_i(x)$$

will only have non-zero coefficients a_i that correspond to non-zero λ_i

If (a part of) the function is not in RKHS, that part of it cannot be learned using kernel regression.

Another way to look at it:

Machine learning problems solve regularization problems.

Kernel ridge regression regularizes the norm $||f||_{\mathcal{H}}$ in RKHS, i.e. the cost function is

$$\sum_{i}^{n} (y_i - f(\boldsymbol{x}_i))^2 + \lambda ||f||_{\mathcal{H}}^2$$

Imagine now that some of the kernel eigenvalues λ_i are zero.

The corresponding terms in the norm will be infinitely large so the corresponding a_i cannot be learned.

The larger the eigenvalues λ_i, the easier it is to learn the corresponding a_i.

The essential part of building a ML model is to find the right kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$.

How is this typically done?

One starts by assuming some functional form for the kernel function.

A few popular choices of the kernel function include:

$$\text{Polynomial:} \quad k(\boldsymbol{x}, \boldsymbol{x}') = \left[1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle\right]^d ,$$

$$\text{Radial basis:} \quad k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left[-\gamma(\boldsymbol{x} - \boldsymbol{x}')^2\right] ,$$

$$\text{Neural network:} \quad k(\boldsymbol{x}, \boldsymbol{x}') = \tanh\left[a_1 \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + a_2\right] ,$$

These functions are parameterized by a few parameters, such as d, or a_1 and a_2, or γ, or α and l in the examples above.

Having chosen a particular functional form of the kernel function, one varies these parameters to find the best model.

A few examples ...

Kernel Ridge regression

The functional to minimize

$$\sum_{i}^{n} (y_i - f(\boldsymbol{x}_i))^2 + \lambda ||f||^2_{\text{Hilbert space}}$$

is equivalent to that in ridge regression, except that the model now is in the high-(N)-dimensional Hilbert space.

Thus, we are dealing with **Kernel Ridge Regression**.

Because the shapes of the functions $f(\boldsymbol{x})$ can, in principle, be arbitrary, Kernel Ridge Regression is capable of building arbitrary models, not just linear models.

To build the model, we start by defining the kernel function $K(\boldsymbol{x}, \boldsymbol{x'})$.

Next, we need to find the equations for the coefficients α_j in

$$f(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j K(\boldsymbol{x}, \boldsymbol{x}_j)$$

The square-of-the-differences term can be written in matrix form as

$$\sum_{i}^{n} (y_i - f(\boldsymbol{x}_i))^2 = (\boldsymbol{y} - \mathbf{K}\boldsymbol{\alpha})^{\mathrm{T}} (\boldsymbol{y} - \mathbf{K}\boldsymbol{\alpha})$$

The regularization term can be written in matrix form as

$$\lambda ||f||^2 = \lambda \boldsymbol{\alpha}^{\mathrm{T}} \mathbf{K} \boldsymbol{\alpha}$$

and the solution for $\boldsymbol{\alpha}$ is

$$\hat{\boldsymbol{\alpha}} = [\mathbf{K} + \lambda \mathbf{I}]^{-1} \boldsymbol{y}$$

The matrix $\mathbf{K}$ is a square $n \times n$ matrix with elements $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the training points.

Given $\hat{\boldsymbol{\alpha}}$, we can write the estimator of the model at point $\boldsymbol{x}^*$

$$\hat{f}(\boldsymbol{x}^*) = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\boldsymbol{\alpha} = \\ = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\left[\mathbf{K} + \lambda\mathbf{I}\right]^{-1}\boldsymbol{y}$$

where

$$\boldsymbol{k}(\boldsymbol{x}^*) = \begin{pmatrix} K(\boldsymbol{x}^*, \boldsymbol{x}_1) \\ K(\boldsymbol{x}^*, \boldsymbol{x}_2) \\ K(\boldsymbol{x}^*, \boldsymbol{x}_3) \\ \vdots \\ K(\boldsymbol{x}^*, \boldsymbol{x}_n) \end{pmatrix}$$

We can see that the model is written in terms of:

$\Rightarrow$ The vector of observations $\boldsymbol{y}$;

$\Rightarrow$ The square matrix $\mathbf{K}$, whose elements are the kernel functions $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$;

$\Rightarrow$ The column vector $\boldsymbol{k}(\boldsymbol{x}^*)$, whose elements are the kernel functions $K(\boldsymbol{x}^*, \boldsymbol{x}_j)$.

Thus, the problem boils down to finding the kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$.

How is this done?

We start by assuming some function form for the kernel function.

A few popular choices of the kernel function include:

$$\text{Polynomial:} \quad K(\boldsymbol{x}, \boldsymbol{x}') = \left[1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle\right]^d,$$

$$\text{Radial basis:} \quad K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left[-\gamma(\boldsymbol{x} - \boldsymbol{x}')^2\right],$$

$$\text{Neural network:} \quad K(\boldsymbol{x}, \boldsymbol{x}') = \tanh\left[a_1 \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + a_2\right],$$

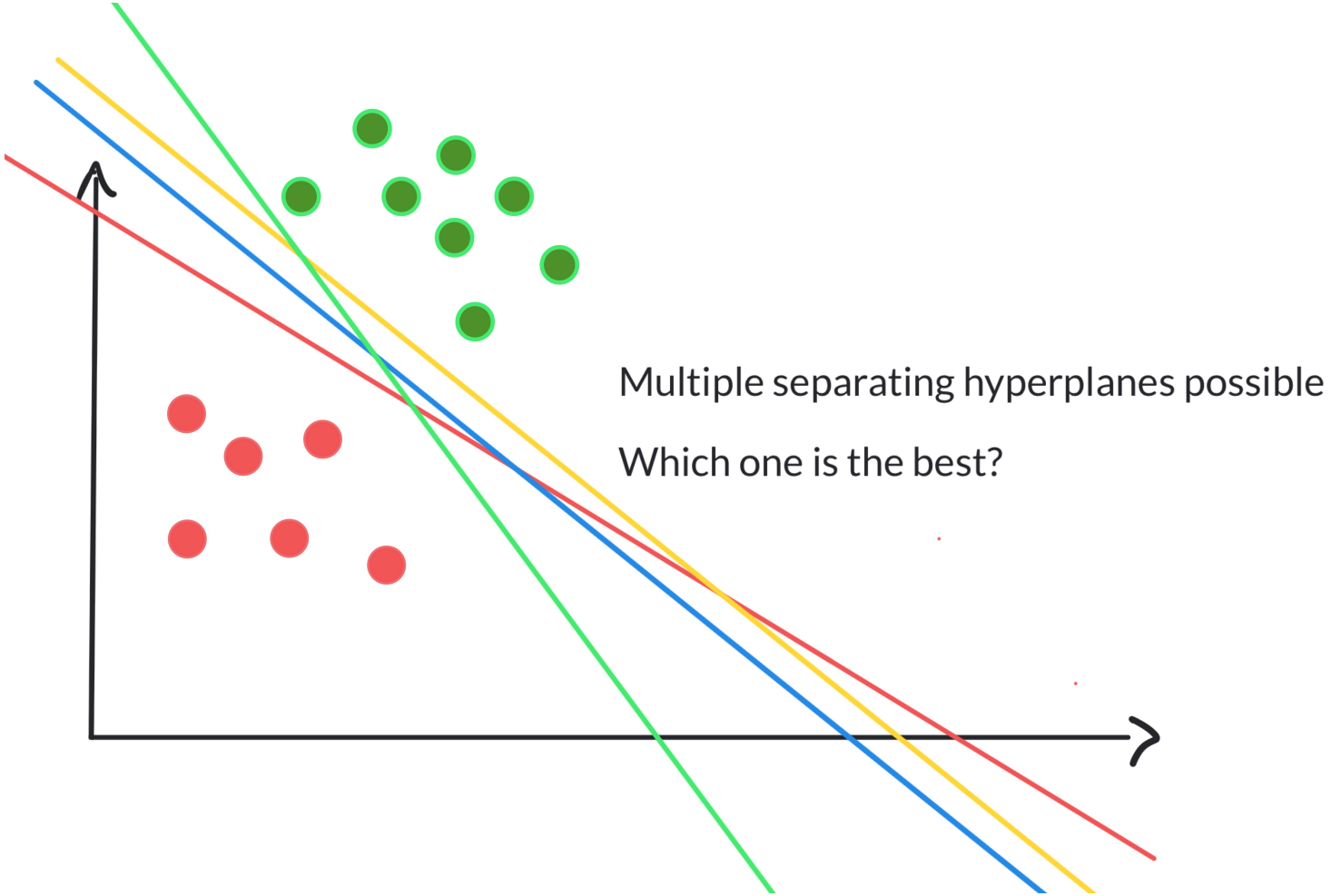
$$\text{Rational Quadratic:} \quad K(\boldsymbol{x}, \boldsymbol{x}') = \left(1 + \frac{|\boldsymbol{x}_i - \boldsymbol{x}_j|}{2\alpha l^2}\right)^{-\alpha},$$

These functions are parameterized by a few parameters, such as d, or a_1 and a_2, or γ, or α and l in the examples above.

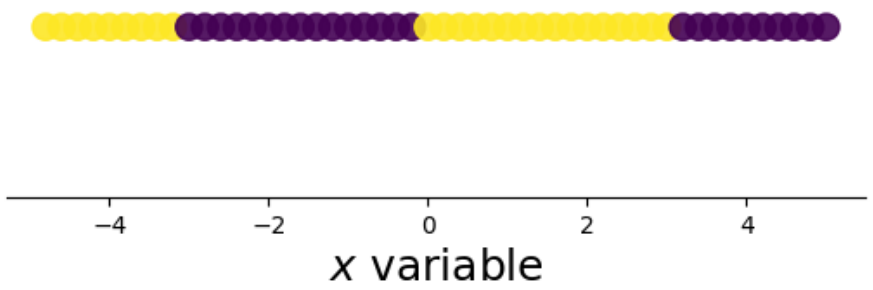
Having chosen a particular functional form of the kernel function, one varies these parameters to minimize the validation error.

For Kernel Ridge Regression, this is often done through cross-validation or by grid search.

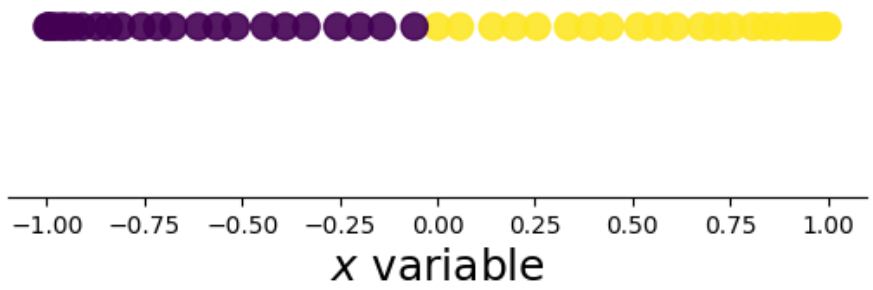
Using kernels for classification problems


Multiple separating hyperplanes possible
Which one is the best?

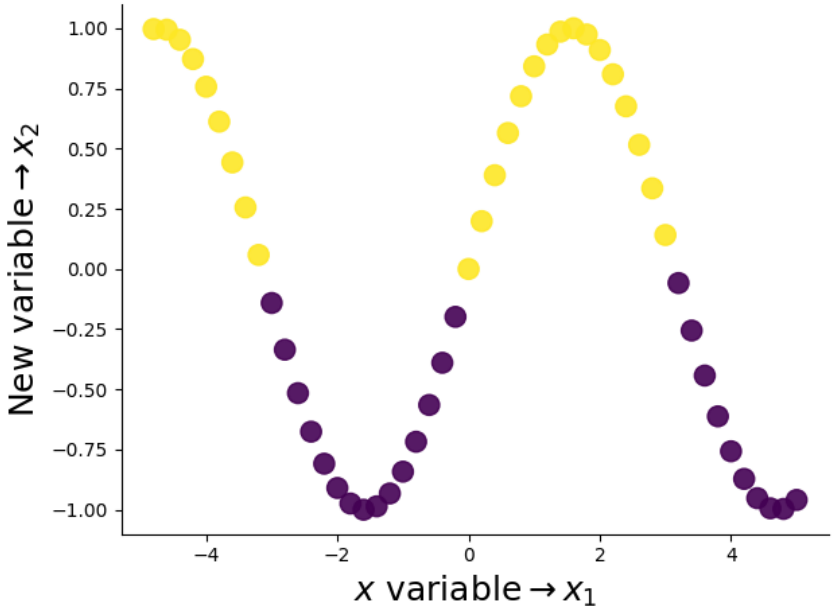
What if the data are not linearly separable, as in this one-dimensional example:



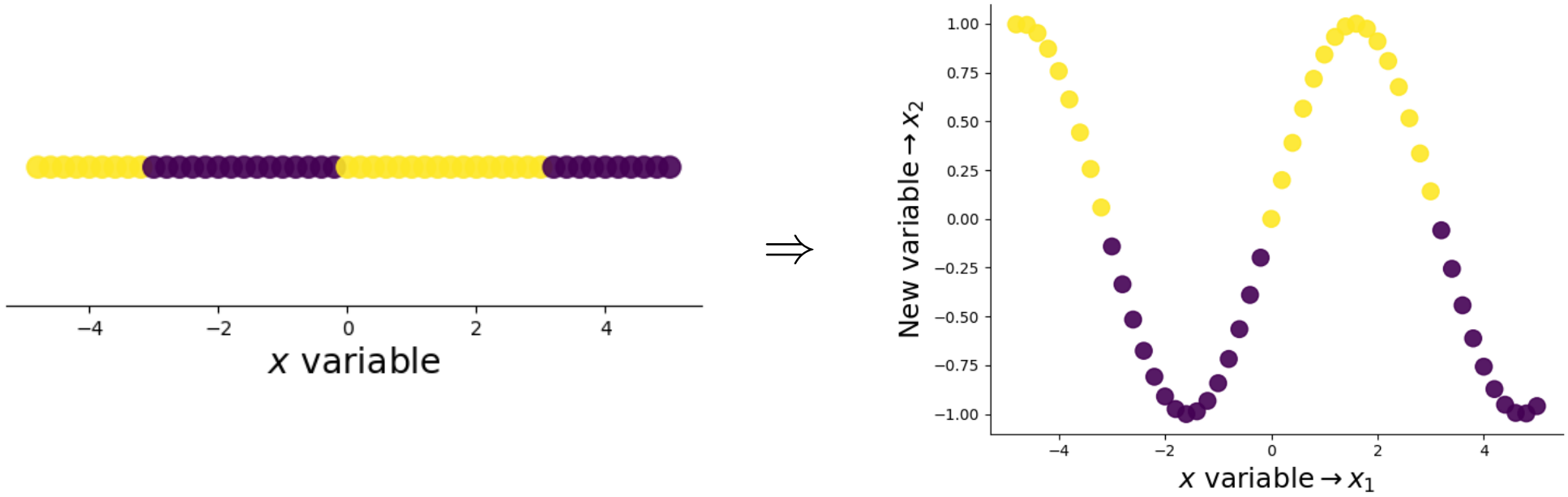
This can be taken care of by a non-linear transformation. In particular, transforming the inputs as $x \rightarrow \sin(x)$ yields:



Introducing another feature as $x_2 \rightarrow \sin(x_1)$ yields:



This is an example of lifting data to a higher-dimensional space to make it separable:



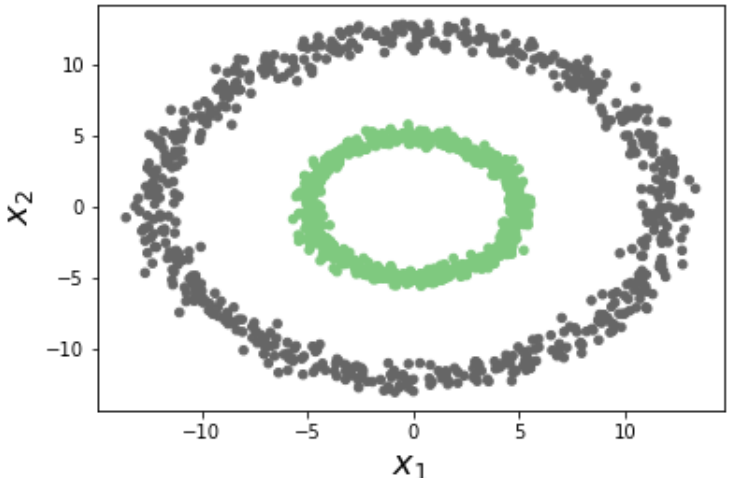
In 1D, the classes are not linearly separable

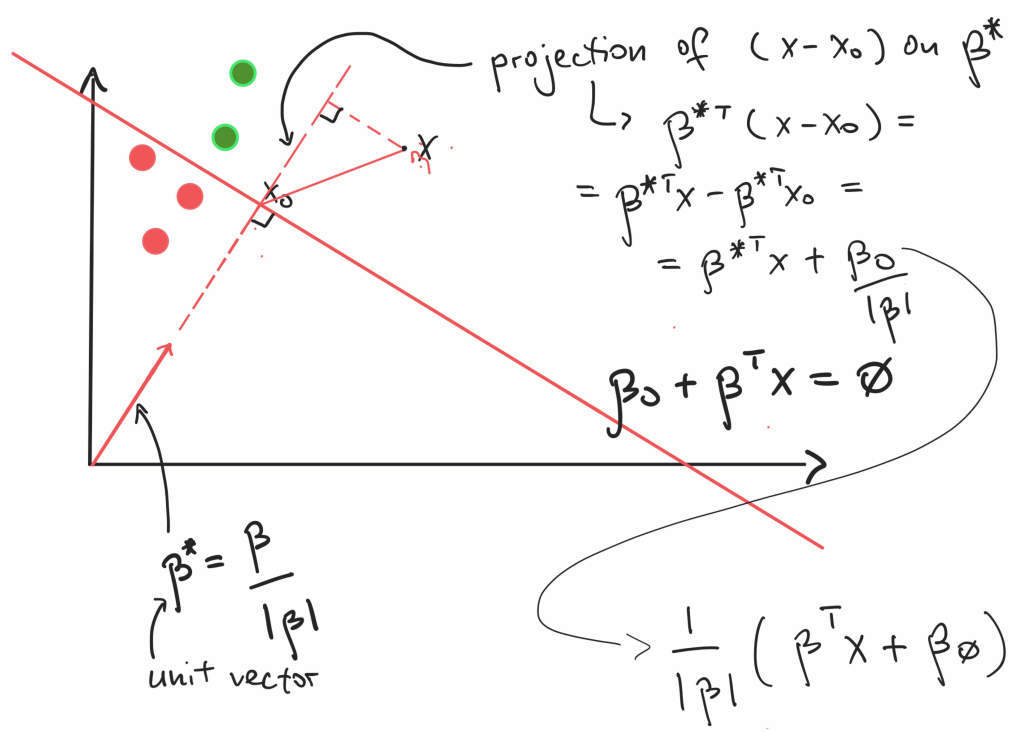
In 2D, the classes are separable by the $x_2 = 0$ line

Support Vector Machine or SVM

SVM is most commonly used to classify data that are not linearly separable.

We will apply it to the following example of clearly non-linearly-separable data:




projection of $(x - x_0)$ on β^*
$\beta^{*T}(x - x_0) =$
$= \beta^{*T}x - \beta^{*T}x_0 =$
$= \beta^{*T}x + \frac{\beta_0}{|\beta|}$
$\beta_0 + \beta^T x = 0$
$\beta^* = \frac{\beta}{|\beta|}$
unit vector
$\frac{1}{|\beta|}(\beta^T x + \beta_0)$
x
x_0

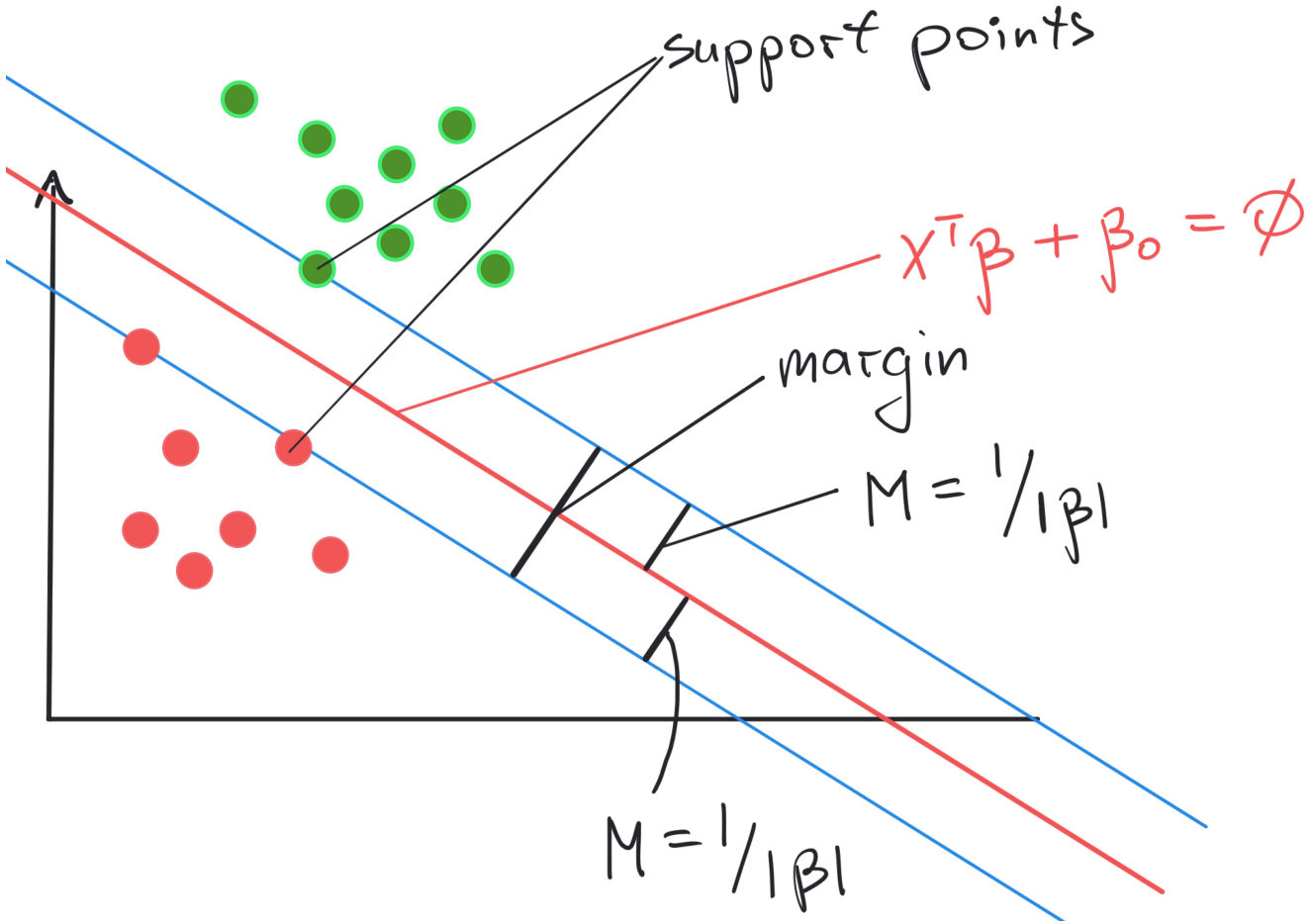
Let me summarize what the figure on the previous page depicts:

$\Rightarrow$ The equation for the hyperplane is $\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x} + \beta_0 = 0$

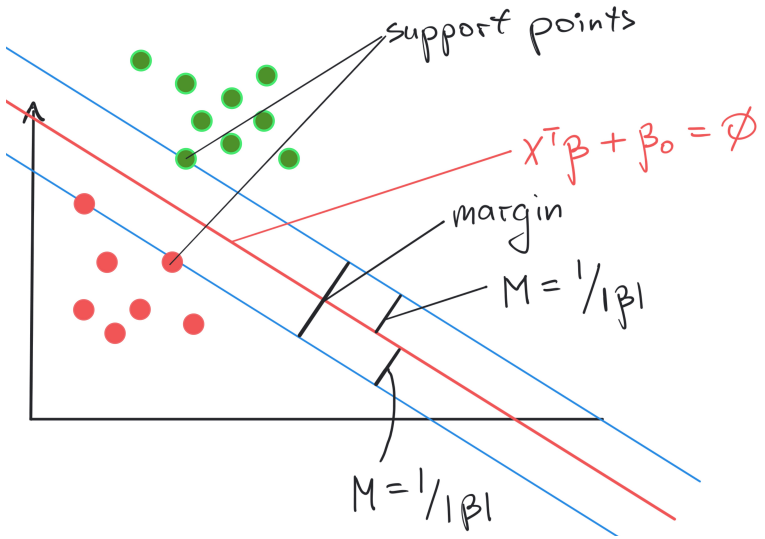
$\Rightarrow$ The shortest distance between a point $\boldsymbol{x}$ and the hyperplane is $\left(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x} + \beta_0\right) / |\boldsymbol{\beta}|$

How does one find the coefficients $\boldsymbol{\beta}$ and β_0 of the hyperplane?

One way to determine the optimal hyperplane is by maximizing the distance M from the data points.



Note that we want every point to be at least the distance $= M$ away from the hyperplane.



Recall that the distance of point $\boldsymbol{x}_i$ from the hyperplane is

$$\left(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x}_i + \beta_0\right)/|\boldsymbol{\beta}|$$

So we want for each point in the dataset:

$$y_i\left(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x}_i + \beta_0\right)/|\boldsymbol{\beta}| \geq M$$

or

$$y_i\left(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x}_i + \beta_0\right) \geq M|\boldsymbol{\beta}|$$

Note that if we scale each of the β coefficients by the same factor, the (in)equality

$$y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) \geq M |\boldsymbol{\beta}|$$

is preserved.

Therefore, we can arbitrarily rescale $\boldsymbol{\beta}$ and β_0 to have

$$|\boldsymbol{\beta}| = 1/M,$$

which will lead us to the condition:

$$y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) \geq 1$$

for every point in the data set.

So, to find the optimal hyperplane, we want to minimize $|\boldsymbol{\beta}|$, while ensuring $y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) \geq 1$ for every data point.

Use: https://en.wikipedia.org/wiki/Lagrange_multiplier

We can do this by minimizing the following (Lagrange) function:

$$L = \frac{1}{2}|\boldsymbol{\beta}|^2 - \sum_i^n \alpha_i \left[y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) - 1 \right]$$

where α_i are chosen such that

$$\alpha_i \left[y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) - 1 \right] = 0 \quad \text{for each } i$$

Let's take the derivative of L and set it to zero:

$$\boldsymbol{\beta} = \sum_i^n \alpha_i y_i \boldsymbol{x}_i$$

$$0 = \sum_i^n \alpha_i y_i$$

We can see that the coefficients $\boldsymbol{\beta}$ are given by the Lagrange multipliers α_i, which can be found numerically.

One way to find the Lagrange multipliers is by restricting α_i to be positive.

If we then substitute the equations on the previous page into L, we will get

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j \qquad \text{subject to } \alpha_i \geq 0$$

It can be shown that L_D provides the lower bound for L.

The minimum of L can therefore be found by maximizing L_D.

If interested, check this https://en.wikipedia.org/wiki/Wolfe_duality

Note that the condition

$$\alpha_i \left[y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) - 1 \right] = 0 \quad \text{for each } i$$

implies the following:

$\Rightarrow$ If $\alpha_i > 0$, then $\left[y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) - 1 \right] = 0$, which means the point $\boldsymbol{x}_i$ lies on the boundary of the margin slab.

$\Rightarrow$ If $\left[y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) - 1 \right] > 0$, the points is outside the margin and $\alpha_i = 0$.

Therefore, the final model coefficients

$$\boldsymbol{\beta} = \sum_{i}^{n} \alpha_i y_i \boldsymbol{x}_i$$

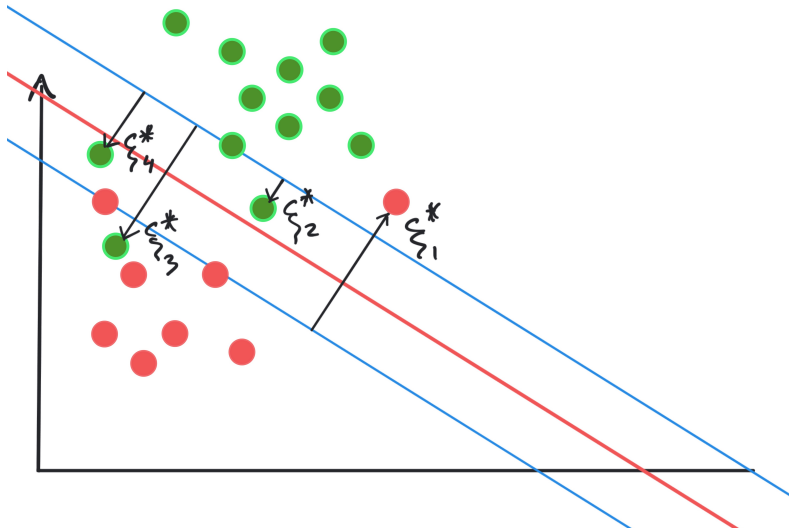
are given in terms of the points $\boldsymbol{x}_i$ that lie on the boundary of the slab.

These points are the support points.

Support Vector Machine (SVM)

We will now consider the case of classes that are not linearly separable.

To do this, we must first allow some points to be on the wrong side of the margin, as in his plot:



Previously, our condition was

$$y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) \geq 1$$

Now we have to require that

$$y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) \geq 1 - \xi_i$$

where $\xi_i = 0$ if the point is on the right side of the margin.

As before, we want to maximize the margin (or minimize $|\beta|$), but let us now add another constraint:

$$\sum_i \xi_i < C = \text{const} \qquad \leftarrow \text{this will add a cost term to the } L \text{ function}$$

The Lagrange function will be

$$L = \frac{1}{2}|\boldsymbol{\beta}|^2 - \sum_i^n \alpha_i \left[y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) - (1 - \xi_i) \right] + C \sum_i \xi_i - \sum_i \mu_i \xi_i$$

This optimization problem has the following conditions:

$$\begin{aligned} &\mu_i \xi_i = 0, \\ &\alpha_i \left[y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) - (1 - \xi_i) \right] = 0, \\ &\left[y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) - (1 - \xi_i) \right] \geq 0 \end{aligned}$$

The goal is to find the minimum of L by varying $\boldsymbol{\beta}$, β_0 and ξ_i.

By setting the corresponding derivatives of L to zero, we get:

$$\boldsymbol{\beta} = \sum_{i}^{n} \alpha_i y_i \boldsymbol{x}_i$$

$$0 = \sum_{i}^{n} \alpha_i y_i$$

$$\alpha_i = C - \mu_i \quad \text{for all } i.$$

By replacing these back into the equation for L, we get:

$$L_D = \sum_{i} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j \qquad \text{subject to } \alpha_i \geq 0$$

Again, L_D provides the lower bound for L.

The minimum of L can therefore be found by maximizing L_D.

So, we need to maximize

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j$$

or

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$$

To deal with non-linearities, we will now introduce feature map:

$$\boldsymbol{x}_i \Rightarrow \Phi(\boldsymbol{x}_i)$$

which gives us

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle$$

But, remember,

$$\langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle = K(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad \text{is the kernel function}$$

So we can write L_D as

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

which can be maximized numerically to find the coefficients α_i and the parameters of the kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

What is the equation for the hyperplane that separates the classes in the high-dimensional space?

In the high-dimensional space: $$\boldsymbol{\beta} = \sum_{i}^{n} \alpha_i y_i \boldsymbol{\varphi}(\boldsymbol{x}_i)$$

$$f(\boldsymbol{x}) = \boldsymbol{\beta}^T \boldsymbol{\varphi}(\boldsymbol{x}) + \beta_0 = \sum_{i} \alpha_i y_i \langle \varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{x}_i) \rangle + \beta_0 = \sum_{i} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + \beta_0$$

To find β_0, we can use $y_i f(\boldsymbol{x}_i) = 1$ for any point, for which $\alpha_i > 0$.

Again, we see that the solution is written in terms of the sum of the kernel functions.

Once and the kernel functions are defined and α_i are found, we know $f(\boldsymbol{x})$.

Can we compare SVM with Ridge Regression?

Let's take another look at the loss function we need to minimize in order to train SVM.

We started by saying that we need to minimize $|\boldsymbol{\beta}|^2$ subject to the following conditions:

$$y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) \geq 1 - \xi_i \quad \text{with } \xi_i \geq 0$$

and

$$\sum_i \xi_i < C = \text{const}$$

How about we simply write

$$\xi_i \geq 1 - y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right)$$

and vary $\boldsymbol{\beta}$ and β_0 to find the minimum of the following function:

$$L(\boldsymbol{\beta}, \beta_0) = \frac{1}{2}|\boldsymbol{\beta}|^2 + C \sum_i^n \left[1 - y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0 \right) \right]_+$$

The problem is equivalent to minimizing the following function:

$$L(\boldsymbol{\beta}, \beta_0) = \frac{1}{2C}|\boldsymbol{\beta}|^2 + \sum_i^n \left[1 - y_i \left(\boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}_i + \beta_0\right)\right]_+$$

This is equivalent to minimizing the following function:

$$L(\boldsymbol{\beta}, \beta_0) = \frac{1}{2}\lambda|\boldsymbol{\beta}|^2 + \sum_i^n \max\left[0, 1 - y_i f(\boldsymbol{x}_i)\right]$$

The red part is the the same as the regularization term in Ridge regression.

The green part is called Hinge loss

Linear Ridge regression:

$$\text{Minimize} \quad L(\boldsymbol{\beta}, \beta_0) = \frac{1}{2}\lambda|\boldsymbol{\beta}|^2 + \sum_i^n [y_i - f(\boldsymbol{x}_i)]^2$$

SVM:

$$\text{Minimize} \quad L(\boldsymbol{\beta}, \beta_0) = \frac{1}{2}\lambda|\boldsymbol{\beta}|^2 + \sum_i^n \max[0, 1 - y_i f(\boldsymbol{x}_i)]$$

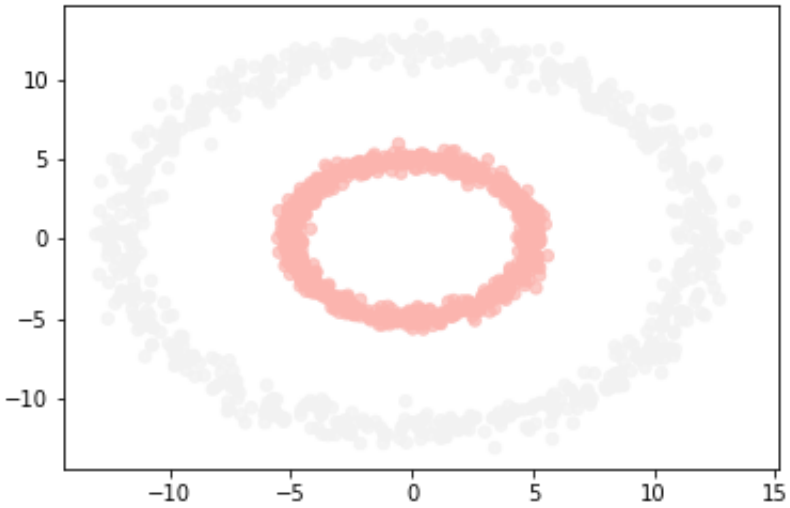
For kernel SVM, a few popular choices of the kernel function include:

$$\text{Polynomial:} \quad k(\boldsymbol{x}, \boldsymbol{x}') = \left[1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle\right]^d,$$
$$\text{Radial basis:} \quad k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left[-\gamma(\boldsymbol{x} - \boldsymbol{x}')\right],$$
$$\text{Neural network:} \quad k(\boldsymbol{x}, \boldsymbol{x}') = \tanh\left[a_1 \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + a_2\right]$$

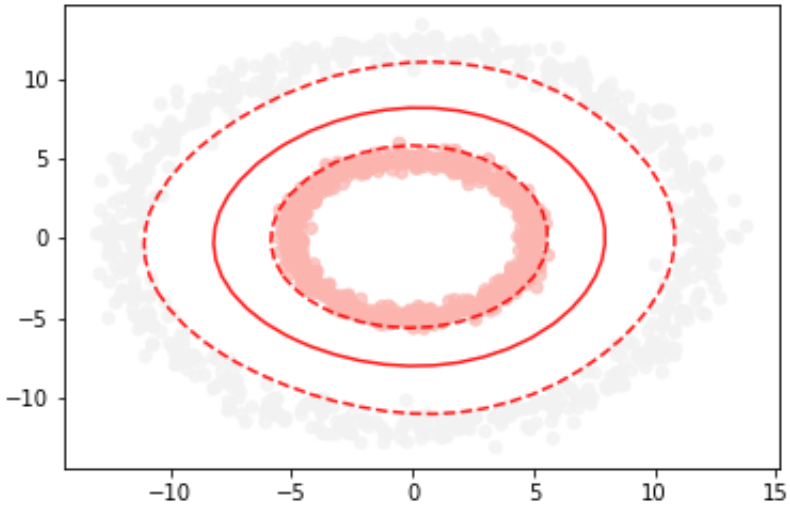
In the following example, we will use the following kernels:

- the polynomial kernel with $d = 1$ (which is equivalent to linear classification), $d = 3$, $d = 6$ and $d = 100$
- the radial basis kernel, or RBF.

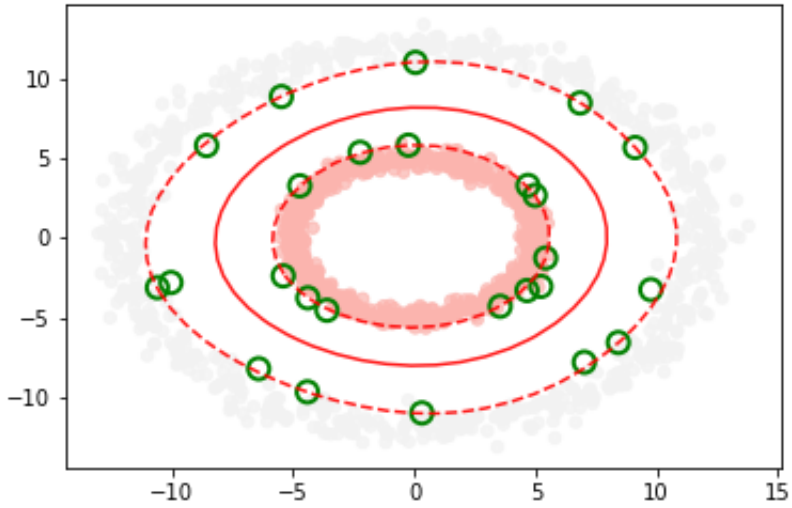
Here's the data we're classifying, dimmed:



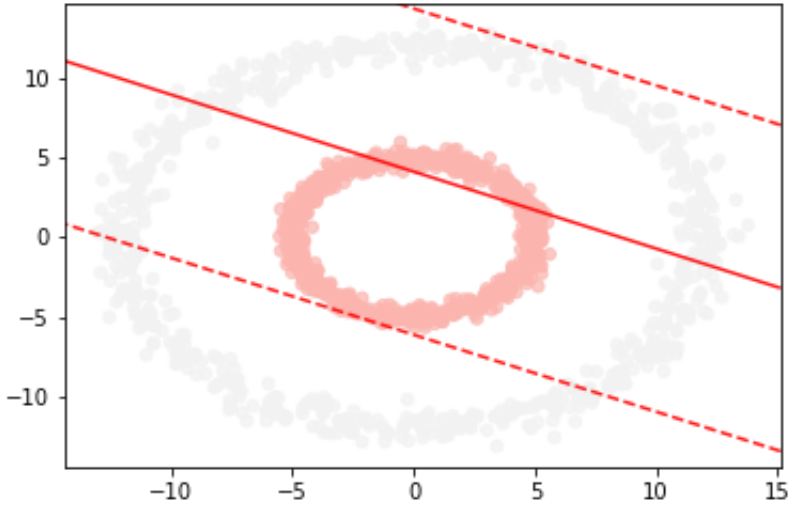
SVM with the RBF kernel produces this for the boundaries of the margin and the separating hyperplane:



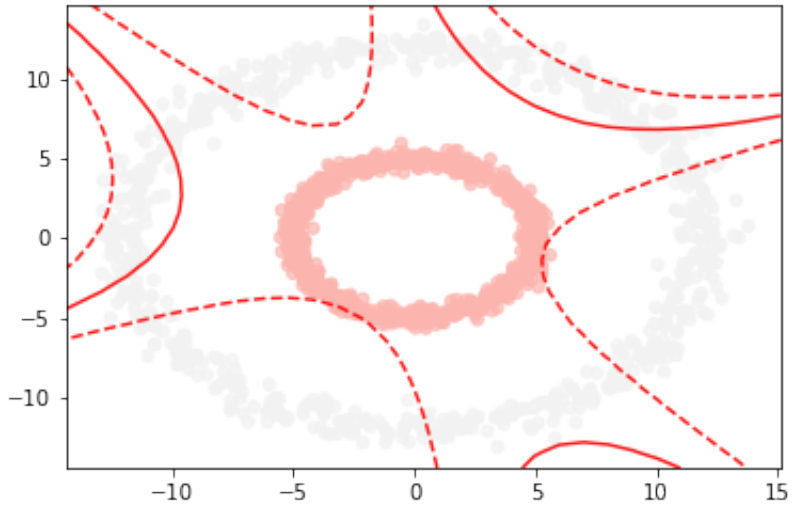
Let's add the support vectors (as open circles):



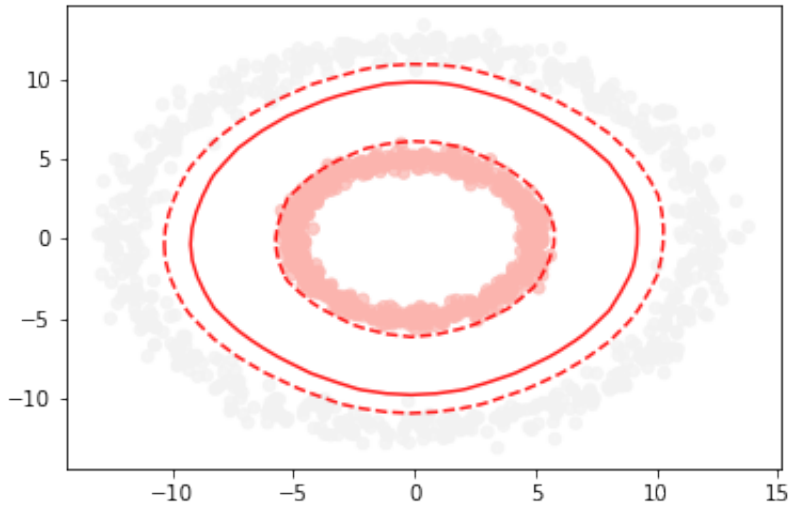
Here's what we get with the linear kernel ($d = 1$):



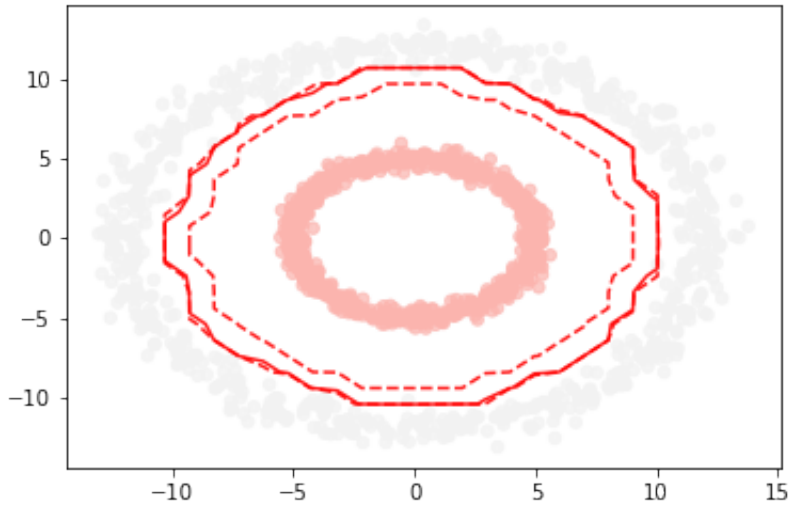
Here's what we get with the polynomial kernel ($d = 3$):



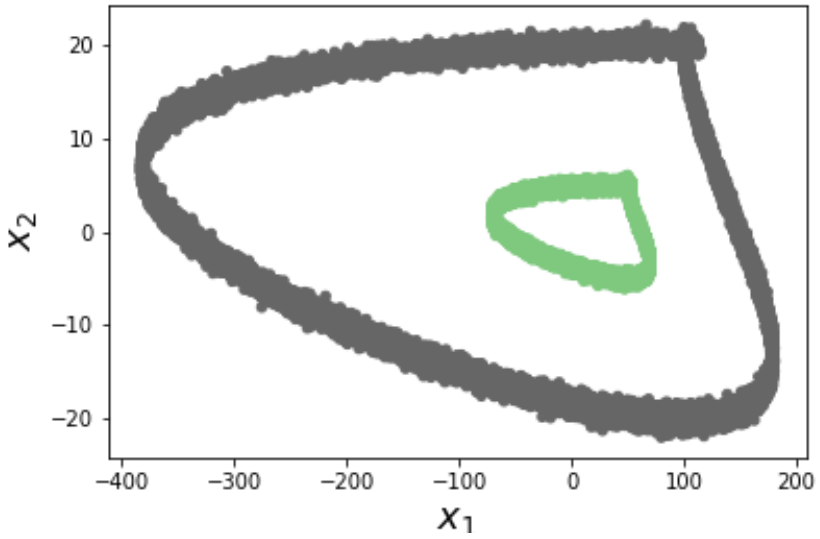
Here's what we get with the polynomial kernel ($d = 6$):



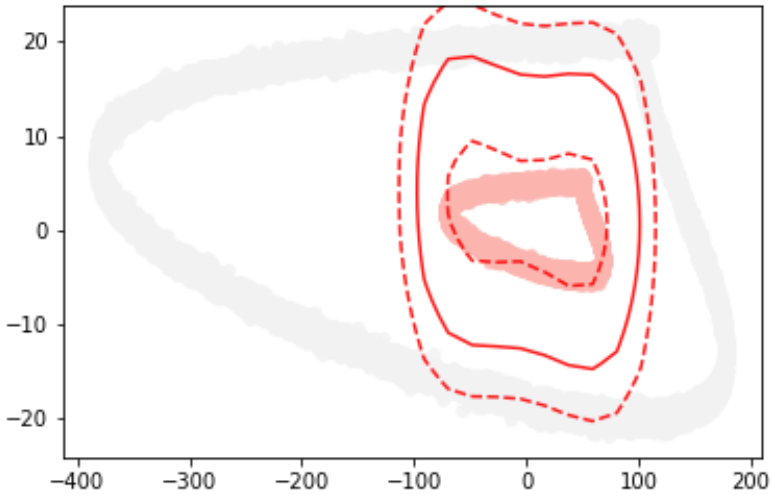
Here's what we get with the polynomial kernel ($d = 100$):

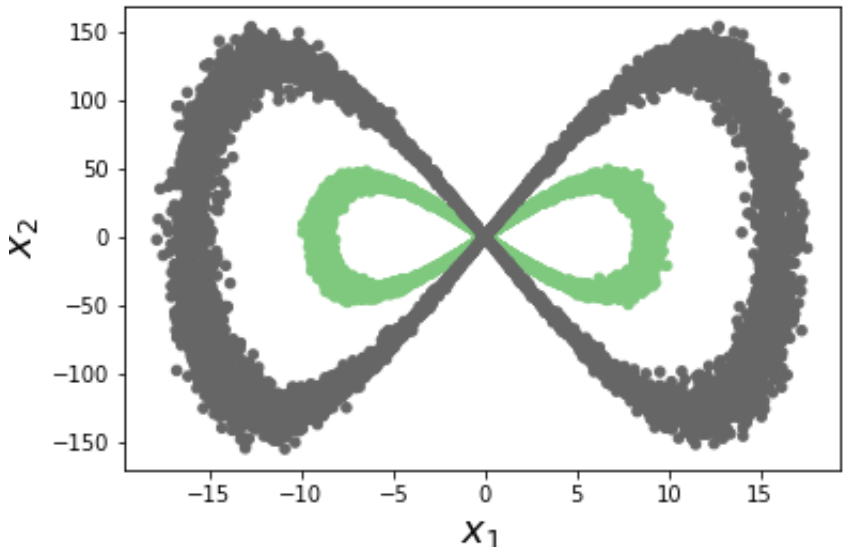


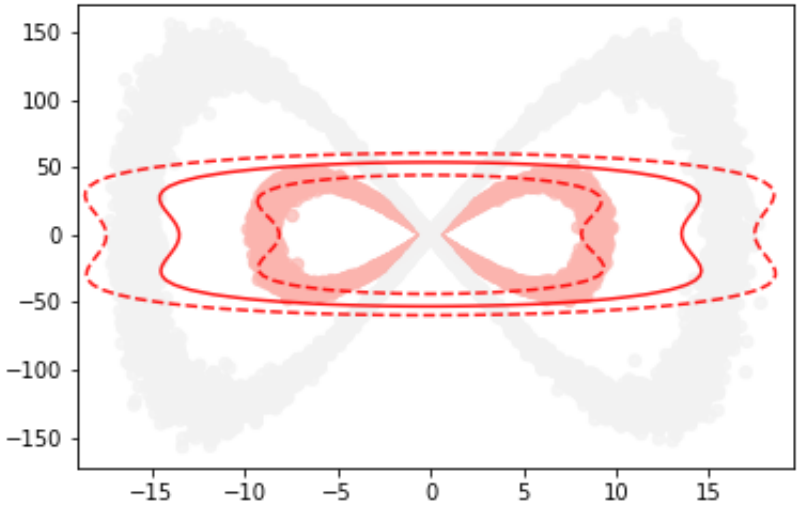
Now, since we are playing with this, let's distort the data to make them look strange, – will our classifier still perform?



Here's what we get with the RBF kernel:

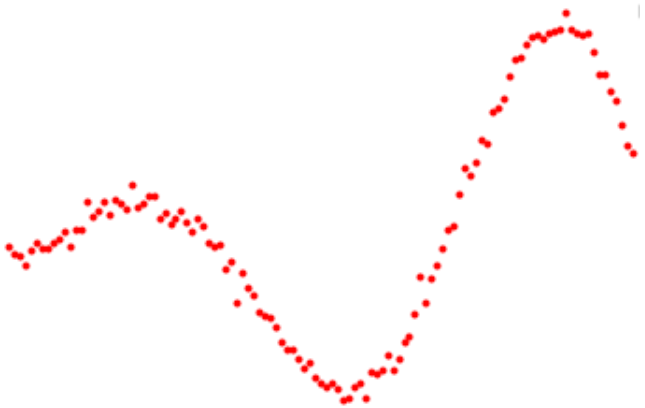



150
100
50
0
−50
−100
−150
−15
−10
−5
0
5
10
15
x_2
x_1

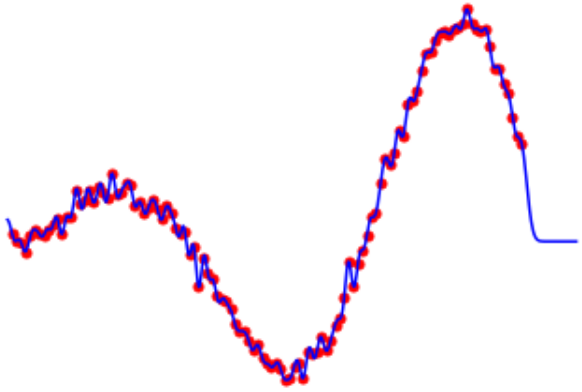

150
100
50
0
−50
−100
−150
−15
−10
−5
0
5
10
15

Gaussian Process Regression – what we will do:

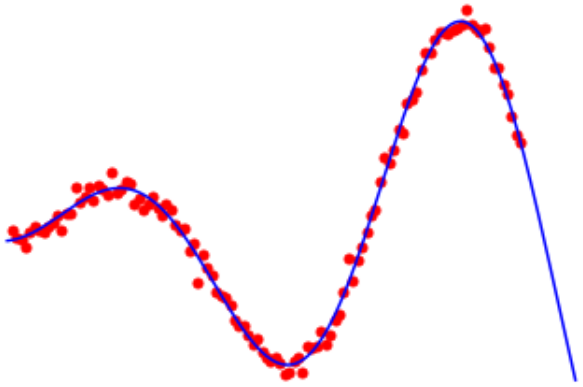
Our goal is to regress data



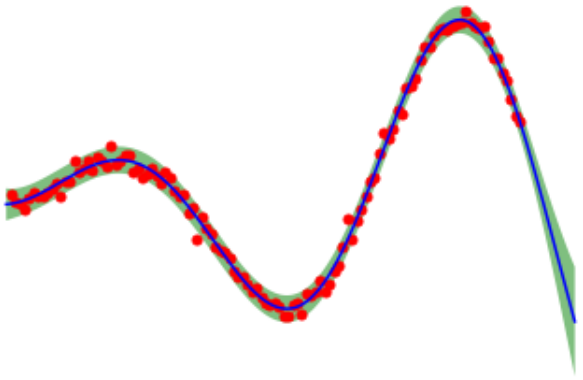
Assuming zero noise variance:



With noise variance:



Including model uncertainty:



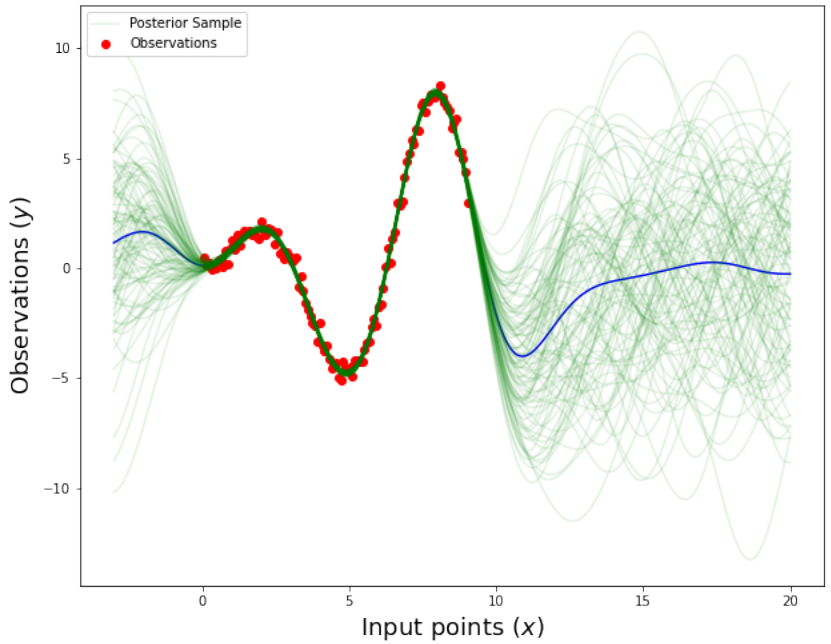
Including noise in GP is equivalent to regularization in KRR.

Take a look at the codes for Gaussian Process Regression in

`gaussian-process-regression.ipynb`

The code uses sklearn and TensorFlow

TensorFlow will offer more control and flexibility, e.g., sampling functions directly from the posterior, as shown here:



Gaussian Process Regression

We have already learned a powerful and general regression tool: Kernel Ridge Regression (KRR).

Why do we need another regression model?

Gaussian Process Regression does what KRR, but also allows us to calculate:

$\Rightarrow$ Bayesian uncertainty of our predictions

As we shall see, this can be used, at least, for two powerful applications:

$\Rightarrow$ Bayesian optimization

$\Rightarrow$ Building better kernel functions

Gaussian Process

Consider the following linear regression function in an N-dimensional space:

$$f(\boldsymbol{x}) = (\beta_0, \beta_1, \cdots, \beta_N) \begin{pmatrix} 1 \\ \varphi_1(\boldsymbol{x}) \\ \vdots \\ \varphi_N(\boldsymbol{x}) \end{pmatrix}$$

Each of the functions $\varphi_i(\boldsymbol{x})$ is some non-linear function of $\boldsymbol{x}$, such as, for example:

$$\varphi_i(\boldsymbol{x}) = \tanh \left[a_i \sum_{j=1}^{p} x_j + b_i \right]$$

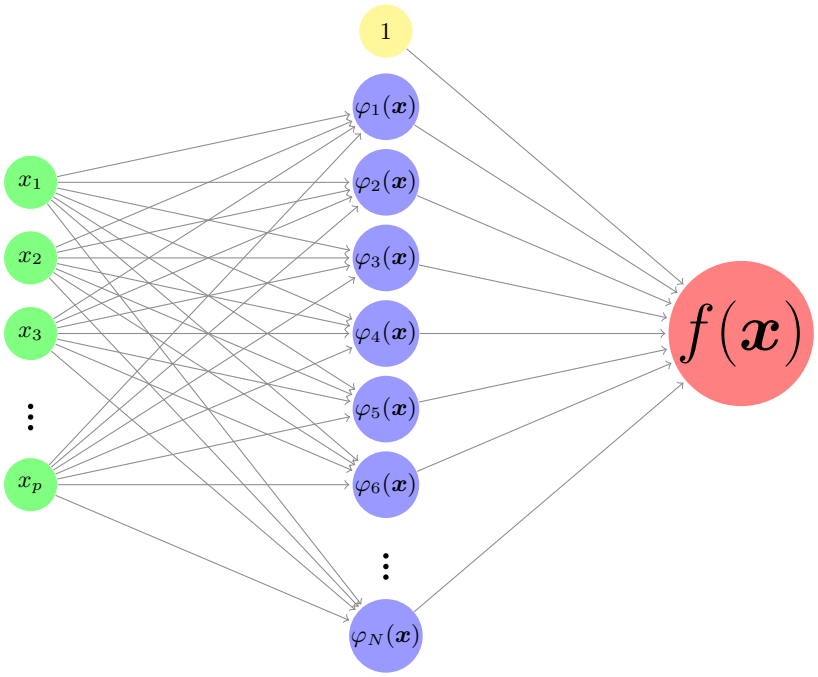
And $\boldsymbol{x}$ is – as before – p-dimensional:

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix}$$

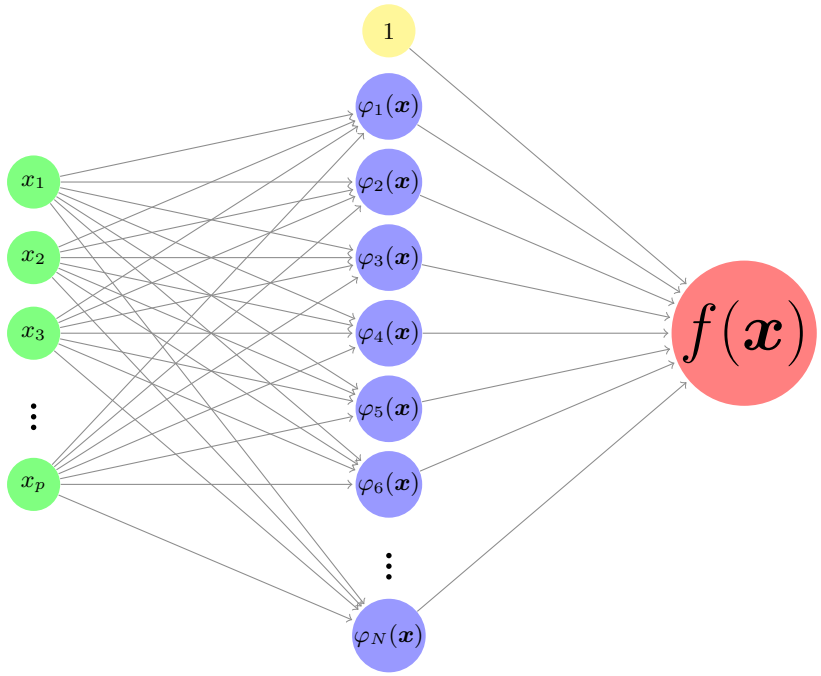
The equation

$$f(\boldsymbol{x}) = (\beta_0, \beta_1, \cdots, \beta_N) \begin{pmatrix} 1 \\ \varphi_1(\boldsymbol{x}) \\ \vdots \\ \varphi_N(\boldsymbol{x}) \end{pmatrix}$$

depicts the following Neural Network:



If all of the parameters a_i, b_i, and $\beta_1, \beta_1, \cdots, \beta_N$ are fixed, then the NN



produces for each $\boldsymbol{x}$ a single value equal to $f(\boldsymbol{x})$.

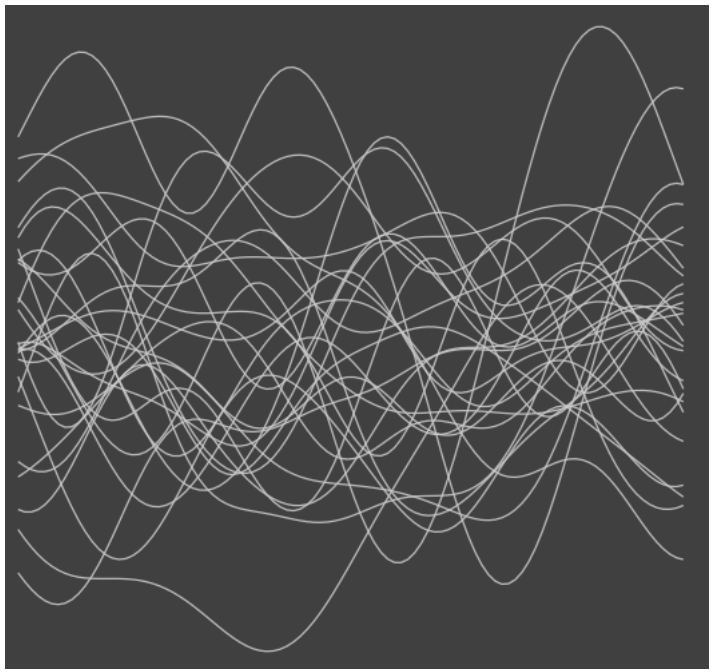
Let us now replace each of $\beta_0, \beta_1, \cdots, \beta_p$ with a random variable, each distributed according to some distribution.

Since the variables are random, each time we draw a value for $\beta_0, \beta_1, \cdots, \beta_N$ and evaluate the NN, it will produce a different, random result.

Thus, the output of the NN for a fixed $\boldsymbol{x}$ is a random variable.

The Neural Network becomes a Bayesian Neural Network.

If we plot the output $f(\boldsymbol{x})$ of a Bayesian Neural Network as a function of $\boldsymbol{x}$, it will be something like this:

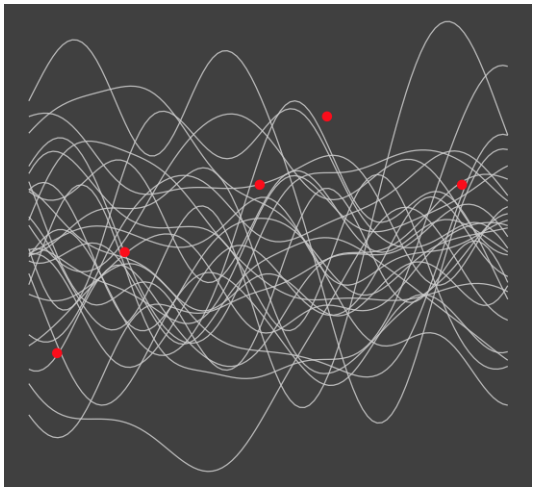


where each curve corresponds to a different combination of $\beta_0, \beta_1, \cdots, \beta_N$.

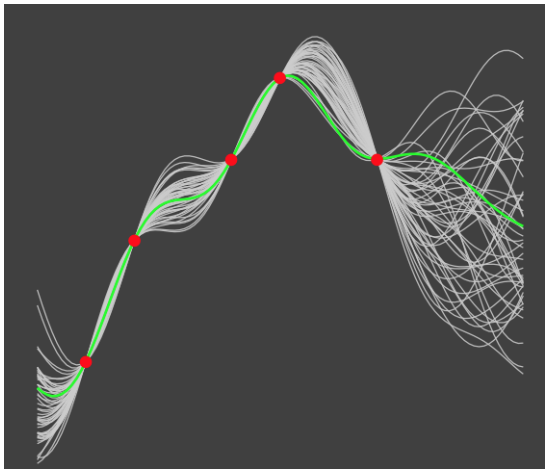
How do we use a Bayesian NN to model data?

One way is to select only the functions that go through our data points and discard all other functions.

We want to go from this:



To this:



In principle, this can be done by sampling functions from the original distributions. But not in practice. There are just too many functions.

Perhaps, the Bayes' theorem can help us?

Let's denote a function corresponding to a specific combination of all parameters of the NN collectively by θ

$$\theta \Rightarrow \mathrm{NN}\left(a_i, b_i, \beta_0, \beta_1, \cdots, \beta_N\right), \qquad \text{where } i = 1, N$$

The output of the NN is a distribution of such function, which we can call $p(\theta)$.

Our goal is to find the functions θ that give a good description of the data: $\{\boldsymbol{X}, \boldsymbol{y}\}$.

Let's set the notation straight:

$p(\theta|\mathbf{X})$ is the unconditional distribution of the NN outputs at the locations of the input space given by $\mathbf{X}$

$p(\theta|\mathbf{X}, \boldsymbol{y})$ is the conditional distribution of the NN outputs at the locations of the input space given by $\mathbf{X}$ conditioned by the observations $\boldsymbol{y}$

We want to condition our distribution:

$$p(\theta|\mathbf{X}) \Rightarrow p(\theta|\{\mathbf{X}, \mathbf{y}\})$$

The Bayes' theorem gives us this conditional distribution:

$$p(\theta|\{\mathbf{X}, \mathbf{y}\}) = \frac{p(\boldsymbol{y}|\theta, \mathbf{X})p(\theta|\mathbf{X})}{p(\boldsymbol{y}|\mathbf{X})}$$

This helps.. but not much.

Even if we could calculate $p(\theta|\{\mathbf{X},\mathbf{y}\})$, how would we use it to make predictions?

Our ultimate goal is to make a prediction of y^* at some point $\boldsymbol{x}^*$.

We can write an equation for the distribution:

$$p(y^*|\boldsymbol{x}^*,\{\mathbf{X},\mathbf{y}\}) = \int_\theta p(y^*|\boldsymbol{x}^*,\theta)p(\theta|\{\mathbf{X},\mathbf{y}\})d\theta$$

and use the LHS to predict the most probable value of y^*.

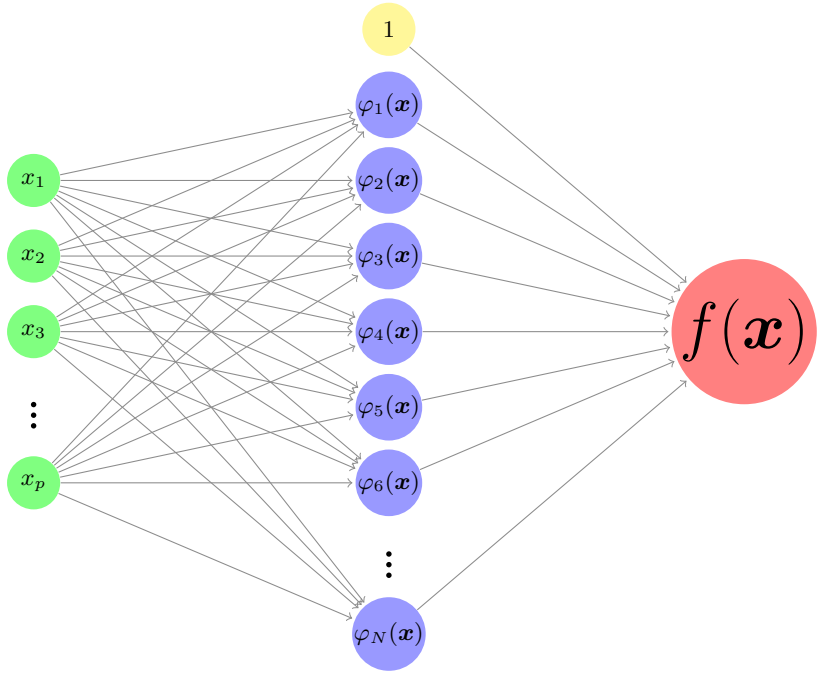
BUT... look at the integral on the RHS. How many parameters determine θ?

There are $N+1$ β_i and then there are $2N$ of a_i and b_i, which is a total of $3N+1$. With $N=20$ (a modest NN), we need to evaluate 61-dimensional integrals in the equation above .. Yikes!

And this is where Gaussian Processes help!

Let's *increase* the size of the NN by making $N \to \infty$.

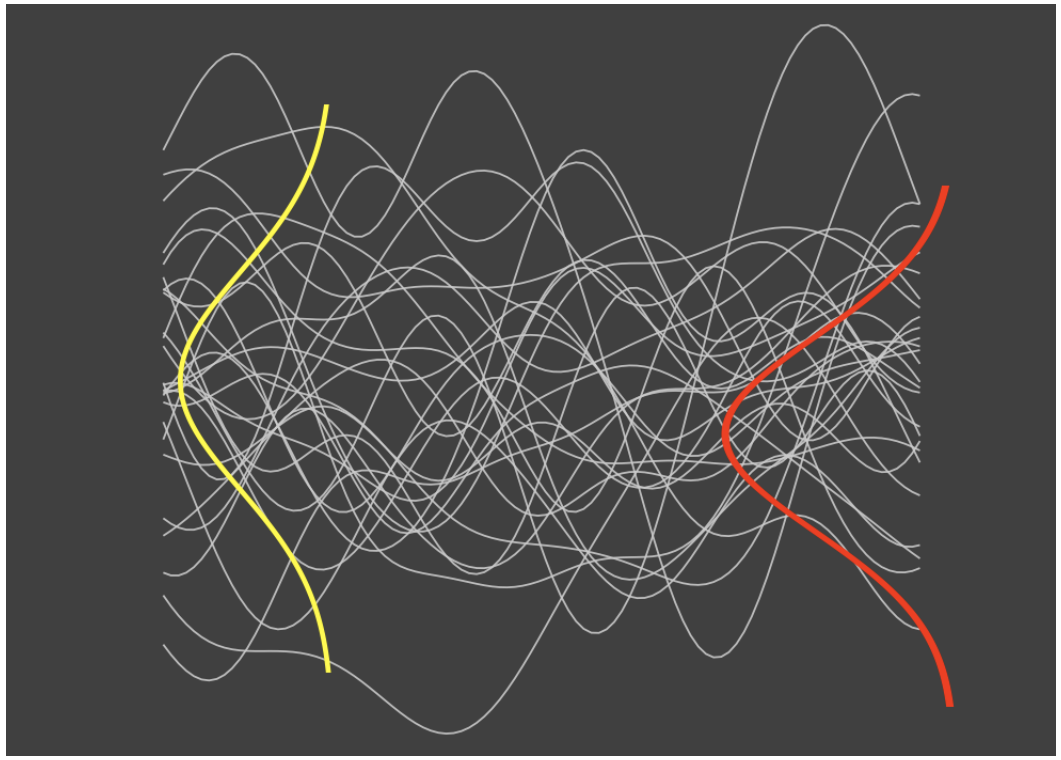
If each of the inputs into the output (red) neuron in the NN



are independent random variables and $N \to \infty$, what is the distribution of the output values for fixed $\boldsymbol{x}$?

The central limit theorem says this distribution will be Gaussian.

This distribution must remain Gaussian for all $\boldsymbol{x}$.

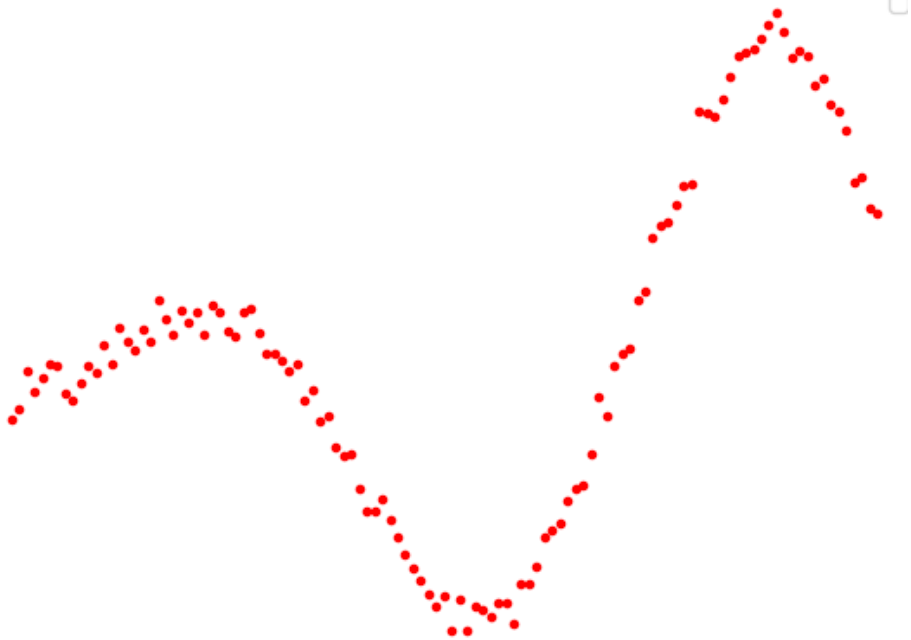


Thus, $f(\boldsymbol{x})$ is a **Gaussian process**.

This helps a lot because the integrals on the previous page can be dealt with analytically!

Let's see how.

Our goal is to infer a function that describes the data:



Data $= f(\boldsymbol{x}) + \varepsilon$, where the noise is usually Gaussian-distributed (with zero mean and some variance σ^2):

$$\varepsilon \sim \mathcal{N}(0, \sigma^2)$$

Let's first consider a linear model:

$$f(\boldsymbol{x}) = \boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x}.$$

Recall that, given a single observation y_i at $\boldsymbol{x}_i$, the model likelihood is defined as equal to $p(\boldsymbol{y}_i|\boldsymbol{\beta}, \boldsymbol{x}_i)$.

We will assume that each data point is independent.

Then, for n observations $\boldsymbol{y}$ at positions in the data matrix $\mathbf{X}$, the likelihood is

$$p(\boldsymbol{y}|\boldsymbol{\beta}, \mathbf{X}) = \prod_{i=1}^{n} p(\boldsymbol{y}_i|\boldsymbol{\beta}, \boldsymbol{x}_i)$$

Because the function $f(\boldsymbol{x})$ differs from the data by Gaussian noise, we can write

$$p(\boldsymbol{y}_i|\boldsymbol{\beta}, \boldsymbol{x}_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(y_i - \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x}_i)^2}{2\sigma^2}\right]$$

Thus, the model likelihood, given the data $\boldsymbol{y}$, is a product of Gaussians.

We can write the likelihood explicitly:

$$p(\boldsymbol{y}|\boldsymbol{\beta}, \mathbf{X}) = \prod_{i=1}^{n} p(\boldsymbol{y}_i|\boldsymbol{\beta}, \boldsymbol{x}_i) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left[-\frac{|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}|^2}{2\sigma^2}\right]$$

Our goal is to use the Bayes' theorem to calculate the posterior:

$$p(\boldsymbol{\beta}|\boldsymbol{y}, \mathbf{X}) = \frac{p(\boldsymbol{y}|\boldsymbol{\beta}, \mathbf{X})p(\boldsymbol{\beta})}{p(\boldsymbol{y}|\mathbf{X})}$$

Note that $p(\boldsymbol{y}|\mathbf{X})$ is just a normalization constant that does not depend on $\boldsymbol{\beta}$.

As always, we have freedom in the choice of the prior (as long as our choice is reasonable), so we choose it as:

$$p(\boldsymbol{\beta}) \sim \mathcal{N}(0, \boldsymbol{\Sigma}_p)$$

which is a joint normal distribution with zero mean and co-variance matrix $\boldsymbol{\Sigma}_p$.

We have:

$$p(\boldsymbol{\beta}|\boldsymbol{y},\mathbf{X}) = \frac{p(\boldsymbol{y}|\boldsymbol{\beta},\mathbf{X})p(\boldsymbol{\beta})}{p(\boldsymbol{y}|\mathbf{X})} = \frac{1}{A}\exp\left[-\frac{|\boldsymbol{y}-\mathbf{X}\boldsymbol{\beta}|^2}{2\sigma^2}\right]\exp\left[-\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\beta}\right]$$

where we put all constant terms into the constant A.

The last equation can be written as

$$p(\boldsymbol{\beta}|\boldsymbol{y},\mathbf{X}) = \frac{1}{A}\exp\left[-\frac{1}{2\sigma^2}(\boldsymbol{y}-\mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y}-\mathbf{X}\boldsymbol{\beta})\right]\exp\left[-\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\beta}\right]$$

which can be further re-written as

$$p(\boldsymbol{\beta}|\boldsymbol{y},\mathbf{X}) = \frac{1}{A}\exp\left[-\frac{1}{2}(\boldsymbol{\beta}-\boldsymbol{\mu})^T\mathbf{C}(\boldsymbol{\beta}-\boldsymbol{\mu})\right]$$

where

$$\boldsymbol{\mu} = \sigma^{-2}\left[\sigma^{-2}\mathbf{X}^{\mathrm{T}}\mathbf{X}+\boldsymbol{\Sigma}_p^{-1}\right]^{-1}\mathbf{X}^T\boldsymbol{y} \qquad \text{mean of the posterior}$$

and

$$\mathbf{C}^{-1} = \left[\sigma^{-2}\mathbf{X}^{\mathrm{T}}\mathbf{X}+\boldsymbol{\Sigma}_p^{-1}\right]^{-1} \qquad \text{covariance matrix}$$

So we have the conditional distribution of the model parameters $p(\boldsymbol{\beta}|\boldsymbol{y}, \mathbf{X})$ given the data $\boldsymbol{y}, \mathbf{X}$.

Again, it is a normal distribution with

mean $\boldsymbol{\mu} = \sigma^{-2}\mathbf{C}^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{y}$ and covariance matrix $\mathbf{C}^{-1}$

Now, we need to multiply this by $p(y^*|\boldsymbol{x}^*, \boldsymbol{\beta})$ and integrate over $\boldsymbol{\beta}$ to obtain

$$p(y^*|\boldsymbol{x}^*) = \int_{\boldsymbol{\beta}} p(y^*|\boldsymbol{x}^*, \boldsymbol{\beta}) p(\boldsymbol{\beta}|\boldsymbol{y}, \mathbf{X})$$

$$p(y^*|\boldsymbol{x}^*) = \int_{\boldsymbol{\beta}} \exp\left[-\frac{(y^* - \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x}^*)^2}{2\sigma^2}\right] \exp\left[-\frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\mu})^T \mathbf{C}(\boldsymbol{\beta} - \boldsymbol{\mu})\right] d\boldsymbol{\beta}$$

It's easy to see that this distribution is going to be Gaussian:

$$p(y^*|\boldsymbol{x}^*) \propto \exp\left[-\frac{(y^* - \hat{\mu})^2}{2\hat{\sigma}^2}\right]$$

It's easy to see that this distribution is going to be Gaussian:

$$p(y^*|\boldsymbol{x}^*) \propto \exp\left[-\frac{(y^* - \hat{\mu})^2}{2\hat{\sigma}^2}\right]$$

The means and the variance of this final distribution can be found by expanding the terms in the integral to be:

$$\hat{\mu} = \boldsymbol{x}^{*\mathrm{T}}\boldsymbol{\mu} = \boldsymbol{x}^{*\mathrm{T}}\sigma^{-2}\mathbf{C}^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{y}$$

$$\hat{\sigma}^2 = \boldsymbol{x}^{*\mathrm{T}}\mathbf{C}^{-1}\boldsymbol{x}^* \quad \text{with} \quad \mathbf{C} = \left[\sigma^{-2}\mathbf{X}^{\mathrm{T}}\mathbf{X} + \boldsymbol{\Sigma}_p^{-1}\right]$$

These are the conditional mean and variance.

The mean can be used to make predictions.

Let's compare this with what we have for linear regression and ridge regression.

Linear regression:

$$\boldsymbol{x}^{*\mathrm{T}}\hat{\boldsymbol{\beta}} = \boldsymbol{x}^{*\mathrm{T}}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{y}$$

Linear Ridge regression:

$$\boldsymbol{x}^{*\mathrm{T}}\hat{\boldsymbol{\beta}} = \boldsymbol{x}^{*\mathrm{T}}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{n}\mathbf{I}\right)^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{y}$$

Linear Gaussian process regression:

$$\hat{\mu} = \boldsymbol{x}^{*\mathrm{T}}\sigma^{-2}\left[\sigma^{-2}\mathbf{X}^{\mathrm{T}}\mathbf{X} + \boldsymbol{\Sigma}_p^{-1}\right]^{-1}\mathbf{X}^{\mathrm{T}}\boldsymbol{y}$$

In order to describe non-linear data, we introduce feature map:

$$\boldsymbol{x} \rightarrow \boldsymbol{\varphi} \Rightarrow \text{vector in the high-dimensional space}$$
$$\mathbf{X} \rightarrow \boldsymbol{\Phi}$$
$$\mathbf{X}^{\mathrm{T}} \rightarrow \boldsymbol{\Phi}^{\mathrm{T}}$$

Then, we have

$$\hat{\mu} = \boldsymbol{\varphi}^{*\mathrm{T}} \boldsymbol{\mu} = \boldsymbol{\varphi}^{*\mathrm{T}} \sigma^{-2} \mathbf{C}^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{y}$$
$$\hat{\sigma}^2 = \boldsymbol{\varphi}^{*\mathrm{T}} \mathbf{C}^{-1} \boldsymbol{\varphi}^* \quad \text{with} \quad \mathbf{C} = \left[\sigma^{-2} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} + \boldsymbol{\Sigma}_p^{-1} \right]$$

Let us now write

$$\mathbf{C} = \sigma^{-2} \left[\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Sigma}_p + \sigma^2 \mathbf{I} \right] \boldsymbol{\Sigma}_p^{-1}$$

$$\mathbf{C} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\mathrm{T}} = \sigma^{-2} \boldsymbol{\Phi}^{\mathrm{T}} \left[\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\mathrm{T}} + \sigma^2 \mathbf{I} \right]$$

$$\mathbf{C}^{-1} \mathbf{C} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\mathrm{T}} = \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\mathrm{T}} = \sigma^{-2} \mathbf{C}^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \left[\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\mathrm{T}} + \sigma^2 \mathbf{I} \right]$$

$$\mathbf{C}^{-1} \boldsymbol{\Phi}^{\mathrm{T}} = \sigma^2 \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\mathrm{T}} \left[\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\mathrm{T}} + \sigma^2 \mathbf{I} \right]^{-1}$$

This gives:

$$\hat{\mu} = \boldsymbol{\varphi}^{*\,\mathrm{T}} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\,\mathrm{T}} \left[\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\,\mathrm{T}} + \sigma^2 \mathbf{I} \right]^{-1} \boldsymbol{y}$$

For variance, we will use the so-called Woodbury formula

$$(\mathbf{E} + \mathbf{FGH})^{-1} = \mathbf{E}^{-1} - \mathbf{E}^{-1}\mathbf{F} \left(\mathbf{G}^{-1} + \mathbf{H}\mathbf{E}^{-1}\mathbf{F} \right)^{-1} \mathbf{H}\mathbf{E}^{-1}$$

applied to

$$\mathbf{C}^{-1} = \left[\sigma^{-2} \boldsymbol{\Phi}^{\,\mathrm{T}} \boldsymbol{\Phi} + \boldsymbol{\Sigma}_p^{-1} \right]^{-1}$$

Replacing:

$$\mathbf{E} \to \boldsymbol{\Sigma}_\mathbf{p}{}^{-1} \quad \mathbf{G}^{-1} \to \mathbf{I}\sigma^2 \quad \mathbf{F} \to \boldsymbol{\Phi}^{\,\mathrm{T}} \quad \mathbf{H} \to \boldsymbol{\Phi}$$

we get

$$\mathbf{C}^{-1} = \boldsymbol{\Sigma}_\mathbf{p} - \boldsymbol{\Sigma}_\mathbf{p} \boldsymbol{\Phi}^{\,\mathrm{T}} \left[\mathbf{I}\sigma^2 + \boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\,\mathrm{T}} \right]^{-1} \boldsymbol{\Phi} \boldsymbol{\Sigma}_p$$

Remember the variance is

$$\hat{\sigma}^2 = \boldsymbol{\varphi}^{*\,\mathrm{T}} \mathbf{C}^{-1} \boldsymbol{\varphi}^*$$

We thus have:

$$\hat{\mu} = \boldsymbol{\varphi}^{*\,\mathrm{T}} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\,\mathrm{T}} \left[\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\,\mathrm{T}} + \sigma^2 \mathbf{I} \right]^{-1} \boldsymbol{y}$$

$$\hat{\sigma}^2 = \boldsymbol{\varphi}^{*\,\mathrm{T}} \boldsymbol{\Sigma}_p \boldsymbol{\varphi}^* - \boldsymbol{\varphi}^{*\,\mathrm{T}} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\,\mathrm{T}} \left[\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\,\mathrm{T}} + \sigma^2 \mathbf{I} \right]^{-1} \boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\varphi}^*$$

Notice that everything is written in terms of $\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\,\mathrm{T}}$ or $\boldsymbol{\varphi}^{*\,\mathrm{T}} \boldsymbol{\Sigma}_p \boldsymbol{\varphi}^*$ or $\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\varphi}^*$.

Remember also that $\boldsymbol{\Sigma}_p$ is the covariance matrix of the prior, which we have some freedom to choose.

To simplify matters, let's choose it to be $\boldsymbol{\Sigma}_p = \mathbf{I}$.

Then, we get:

$$\hat{\mu} = \boldsymbol{\varphi}^{*\,\mathrm{T}} \boldsymbol{\Phi}^{\,\mathrm{T}} \left[\boldsymbol{\Phi} \boldsymbol{\Phi}^{\,\mathrm{T}} + \sigma^2 \mathbf{I} \right]^{-1} \boldsymbol{y}$$

$$\hat{\sigma}^2 = \boldsymbol{\varphi}^{*\,\mathrm{T}} \boldsymbol{\varphi}^* - \boldsymbol{\varphi}^{*\,\mathrm{T}} \boldsymbol{\Phi}^{\,\mathrm{T}} \left[\boldsymbol{\Phi} \boldsymbol{\Phi}^{\,\mathrm{T}} + \sigma^2 \mathbf{I} \right]^{-1} \boldsymbol{\Phi} \boldsymbol{\varphi}^*$$

Look closely at these equations:

$$\boldsymbol{\varphi}^{*\,\mathrm{T}}\boldsymbol{\varphi}^* = k(\boldsymbol{x}^*, \boldsymbol{x}^*) \quad \text{is the kernel function at } (\boldsymbol{x}^*, \boldsymbol{x}^*)$$
$$\boldsymbol{\Phi}\boldsymbol{\varphi} = \boldsymbol{k}(\boldsymbol{x}^*) \quad \text{is a vector of kernel functions } K(\boldsymbol{x}_i, \boldsymbol{x}^*)$$
$$\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} = \mathbf{K} \quad \text{is a matrix of kernel functions } K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

With this in mind we can write:

$$\hat{\mu} = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\left[\mathbf{K} + \sigma^2\mathbf{I}\right]^{-1}\boldsymbol{y}$$

$$\hat{\sigma}^2 = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\left[\mathbf{K} + \sigma^2\mathbf{I}\right]^{-1}\boldsymbol{k}(\boldsymbol{x}^*)$$

We see that everything is determined by the kernel functions $K(\boldsymbol{x}, \boldsymbol{x}')$.

You might ask if the choice of $\boldsymbol{\Sigma}_p = \mathbf{I}$ is justified?

I will say: It doesn't matter. If the prior co-variance matrix is not identity, I can redefine the kernels, as follows:

$$\boldsymbol{\varphi}^{*\,\mathrm{T}} \boldsymbol{\Sigma}_p \boldsymbol{\varphi}^* = K(\boldsymbol{x}^*, \boldsymbol{x}^*) \quad \text{is the kernel function at } (\boldsymbol{x}^*, \boldsymbol{x}^*)$$
$$\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\varphi} = \boldsymbol{k}(\boldsymbol{x}^*) \quad \text{is a vector of kernel functions } K(\boldsymbol{x}^*, \boldsymbol{x}_i)$$
$$\boldsymbol{\Phi} \boldsymbol{\Sigma}_p \boldsymbol{\Phi}^{\mathrm{T}} = \mathbf{K} \quad \text{is a matrix of kernel functions } K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

We can do this, because

$\Rightarrow \boldsymbol{\Sigma}_p$ is just a constant matrix.

$\Rightarrow \boldsymbol{\Sigma}_p$ is positive definite (since it is a co-variance matrix).

Final results for Gaussian Process regression:

Mean of the predictive distribution to be used as the prediction of the model:

$$\hat{\mu} = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*) \left[\mathbf{K} + \sigma^2 \mathbf{I}\right]^{-1} \boldsymbol{y}$$

Conditional variance to be used as Bayesian uncertainty:

$$\hat{\sigma}^2 = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*) \left[\mathbf{K} + \sigma^2 \mathbf{I}\right]^{-1} \boldsymbol{k}(\boldsymbol{x}^*)$$

Compare this with what we obtained for Kernel Ridge Regression:

$$\hat{f}(\boldsymbol{x}^*) = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\boldsymbol{\alpha} = \\ = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*) \left[\mathbf{K} + \lambda \mathbf{I}\right]^{-1} \boldsymbol{y}$$

How are Gaussian Processes trained?

- As before, one starts by choosing a function for $K(\boldsymbol{x}, \boldsymbol{x'})$.
- The parameters of this function are then varied to maximize $p(\boldsymbol{y}|\mathbf{X})$.
- Recall $p(\boldsymbol{y}|\mathbf{X})$.. This is the term in the denominator of the Bayes' theorem:

$$p(\boldsymbol{y}|\mathbf{X}) = \int_{\theta} p(\boldsymbol{y}|\theta, \mathbf{X}) p(\theta|\mathbf{X}) d\theta$$

This multiplies the model likelihood $p(\boldsymbol{y}|\theta, \mathbf{X})$ by $p(\theta|\mathbf{X})$ and integrates over all random function values.

When we integrate over a random variable as θ above, we are 'marginalizing' this variable.

The likelihood integrated as above is thus called 'marginal likelihood'.

Marginal likelihood

Marginal likelihood for a Gaussian process can be expressed in terms of kernels. – How?

To answer this questions, let's consider again the prior $p(\theta|\mathbf{X})$.

I remind you that

θ represents functions drawn from a Gaussian process

$p(\theta|\mathbf{X})$ is the unconditional distribution of the NN outputs at the locations of the input space given by $\mathbf{X}$

Because the functions θ are drawn from a Gaussian process, $p(\theta|\mathbf{X})$ is a multi-variate Gaussian.

This means that $p(\theta|\boldsymbol{x}_i)$ and $p(\theta|\boldsymbol{x}_j)$ are both Gaussian with some covariance $\mathrm{Cov}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ between them.

We will choose the covariance of the Gaussian Process prior to be the kernel function:

$$\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{x'}) = K(\boldsymbol{x}, \boldsymbol{x'})$$

This is actually equivalent to the choice of the prior we had previously made:

$$p(\boldsymbol{\beta}) \sim \mathcal{N}(0, \boldsymbol{\Sigma}_p)$$

To see this, consider

$$f(\boldsymbol{x}) = \boldsymbol{\beta}^T \boldsymbol{\varphi}(\boldsymbol{x})$$

Then:

$E[f(\boldsymbol{x})] = \boldsymbol{\varphi}(\boldsymbol{x})^{\mathrm{T}} E[\boldsymbol{\beta}] = 0 \quad$ and

$\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{x'}) = E[f(\boldsymbol{x}) f(\boldsymbol{x'})] = \boldsymbol{\varphi}^{\mathrm{T}}(\boldsymbol{x}) E[\boldsymbol{\beta}\boldsymbol{\beta}^{\mathrm{T}}] \boldsymbol{\varphi}(\boldsymbol{x'}) = \boldsymbol{\varphi}^{\mathrm{T}}(\boldsymbol{x}) \boldsymbol{\Sigma}_p \boldsymbol{\varphi}(\boldsymbol{x'})$

Our goal is to evaluate the marginal likelihood:

$$p(\boldsymbol{y}|\mathbf{X}) = \int_{\theta} p(\boldsymbol{y}|\theta, \mathbf{X}) p(\theta|\mathbf{X}) d\theta$$

It is more convenient to work with the logarithm of marginal likelihood.

Given that the prior's covariance is the kernel matrix and the integrand is a product of two Gaussians, it is actually possible to write this in terms of the kernel matrix as follows:

$$\log p(\boldsymbol{y}|\mathbf{X}) = -\frac{1}{2}\boldsymbol{y}^{\mathrm{T}} \left(\mathbf{K} + \sigma^2 \mathbf{I}\right) \boldsymbol{y} - \frac{1}{2} \log |\mathbf{K} + \sigma^2 \mathbf{I}| - \frac{n}{2} \log 2\pi$$

The purpose of training a Gaussian process is to find the parameters of the kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$ that maximizes the logarithm of marginal likelihood.

End of Part I

Submit questions about Part I by the end of the day to
rkrems@chem.ubc.ca

Interplay of Machine Learning and Quantum Theory

Roman Krems

University of British Columbia, Vancouver, Canada

August 26, 2021

Part II:

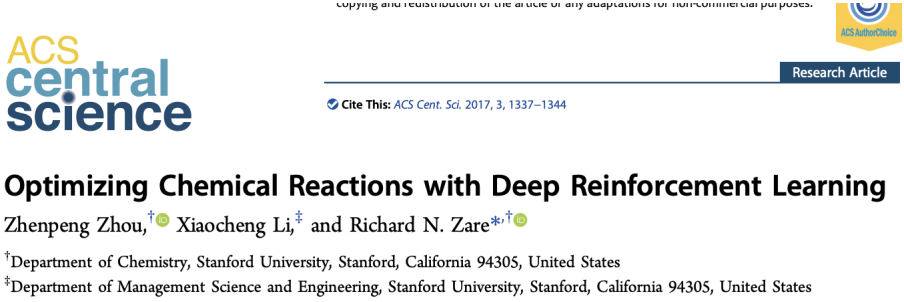
- The Bayes' theorem, likelihood
- Bayesian optimization for inverse quantum problems
- Model selection metrics

  Bayesian Information Criterion
- How to design the best kernel?
- Accelerating Bayesian optimization by kernel improvement

$$\mathcal{L}(\boldsymbol{\beta}) = ||\boldsymbol{y} - \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x}||_2^2 + \lambda|\boldsymbol{\beta}| \qquad \leftarrow \text{Lasso}$$

$$\mathcal{L}(\boldsymbol{\beta}) = ||\boldsymbol{y} - \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x}||_2^2 + \lambda|\boldsymbol{\beta}|^2 \qquad \leftarrow \text{Ridge}$$

For example, in



The experimental variables are

$$p = \text{gas pressure},\ u = \text{voltage},\ v = \text{flow rate}.$$

One could work with this 3-dimensional feature space.

However, what if these features are not the best? What if some combinations of these features are better for modelling reaction yields?

To determine the best features, the authors engineered new features as: $u, v, pu, pv, uv, p/u, p/v, u/v, p^2, u^2, v^2$

Using these new features as variables, they trained 11-dimensional linear regression and looked at the three main variables determining the variation of their output. The result was $p/v, u/v, p2$.

We will need two important concepts for today's discussion:

The Bayes' theorem
Likelihood

Bayes' theorem

- Bayes' theorem is the basis of Bayesian inference.
- It shows how a degree of belief must be modified to account for evidence.
- Bayes' theorem operates with probabilities or probability densities.
- Therefore, the degree of belief must be expressed as a probability.

First, some basic definitions:

$P(X)$ is a probability if X is a discrete variable (e.g., heads or tails)

$P(X|Y)$ is a conditional probability – i.e. probability of X given Y.

If X is continuous, we must work with the probability density.

$\int_a^b p_X(x)dx$ is the probability that $X \in [a, b]$, if X is continuous.

In this case, $p_X(x)$ is the (probability) density.

We will also encounter conditional probability density:

$$p_{X|Y=y}(x) = p_X(x|Y = y)$$

For two events A and B, Bayes' theorem is

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

An event is a realization of a random discrete variable:

$$A = \{X = x\} \qquad B = \{Y = y\}$$

The New York Times

The Mathematics of Changing Your Mind

By John Allen Paulos

Aug. 5, 2011

Consider a concrete example. Assume that you're presented with three coins, two of them fair and the other a counterfeit that always lands heads. If you randomly pick one of the three coins, the probability that it's the counterfeit is 1 in 3. This is the prior probability of the hypothesis that the coin is counterfeit. Now after picking the coin, you flip it three times and observe that it lands heads each time. Seeing this new evidence that your chosen coin has landed heads three times in a row, you want to know the revised posterior probability that it is the counterfeit. The answer

Example: An undergraduate student wants to calculate the probability that she/he will get into a chemistry graduate program if she/he gets an A in Chem121.

Rephrasing this in the language of Bayesian inference, this problem should read something like this:

You meet a first-year undergraduate chemistry student. The student asks you: what is the probability that I will be admitted into your grad school?

You say: 5/100

The student says: but I just passed Chem121 and got an A.

You say: In this case, the probability is 16.7/100

How did you come up with this answer? Let's see...

$$\text{Define} \Rightarrow \quad X = \begin{cases} 1, \text{if Chem121 grade} \geq \text{A} \\ 0, \text{otherwise} \end{cases}$$

$$Y = \begin{cases} 1, \text{ if in grad school} \\ 0, \text{otherwise} \end{cases}$$

$$P(Y = 1) = \frac{\text{number of graduate students}}{\text{total number of undergraduate students}}$$

$$P(X = 1|Y = 1) = \frac{\text{number of graduate students with A in Chem121}}{\text{total number of graduate students}}$$

$$P(X = 1) = \frac{\text{number of A grades in Chem121}}{\text{total number of undergraduate students}}$$

For example, let's say that 1 in 20 undergraduate students go to grad school and that 2/3 graduate students had an A in Chem121 when they took it. The number of A's in Chem121 is typically 1 in 5.

The naive probability to become a graduate student is $1/20 = 5/100$.

The Bayesian probability, given an A in first year Chemistry, is

$$P(Y = 1|X = 1) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{2/3 \times 1/20}{1/5} = 16.7/100$$

For two events A and B, Bayes' theorem is

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Or, we can use $A = \{X = x\}$ and $B = \{Y = y\}$, to write Bayes' theorem for two discrete random variables X and Y as

$$P(Y = y|X = x) = \frac{P(X = x|Y = y)P(Y = y)}{P(X = x)}$$

If X and/or Y are continuous, replace the corresponding probabilities with probability densities.

For example, if X is continuous,

$P(X = x|Y = y) \to p_{X|Y=y}(x)$ and

$P(X = x) \to p_X(x)$ are the probability density functions.

Likelihood

Likelihood = likelihood function

Let's start by saying what likelihood is not.

Likelihood is not a probability or probability density.

Consider first the case of a discrete random variable X.

Consider a stochastic process that produces one of the values of X

Every process is usually a function of one or more parameters β.

Example:

Stochastic process $\Rightarrow$ Flip of a coin
Random variable $\Rightarrow X \Rightarrow \{H, T\}$
Parameter β: Fairness of the coin $\rightarrow$ the outcome depends on fairness
Define β as a continuous variable $\in [0, 1]$ equal to the probability of H

Given β thus defined, we can calculate the probability of observing HH in two coin flips:

$$P(HH|\beta = 0) = 0$$
$$P(HH|\beta = 0.5) = 1/4$$
$$P(HH|\beta = 1) = 1$$

Likelihood is a **function of** β, given an observation.

It is defined as follows:

$$\mathcal{L}(\beta|HH) = P(HH|\beta)$$

Why do we need likelihood?

Let's say you have a coin, but you don't know how fair it is: i.e. you don't know β_{coin}.

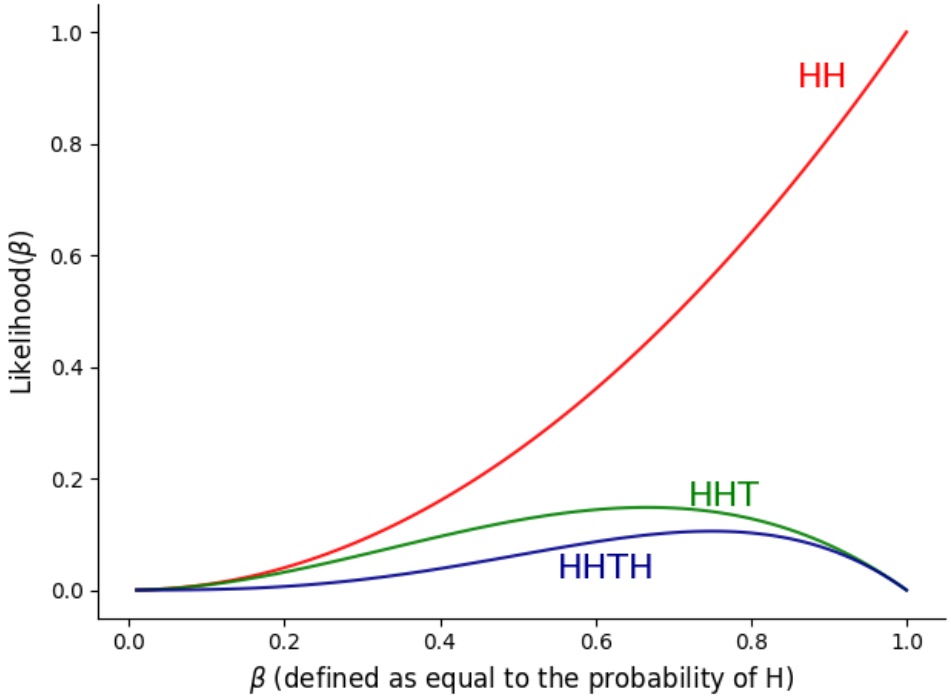
Likelihood allows you to estimate β_{coin}**!** Let's see how..

First, note that **likelihood depends on the observations!**

If we flip the coin twice and get HH, $\mathcal{L}(\beta|HH)$ is the red curve below.

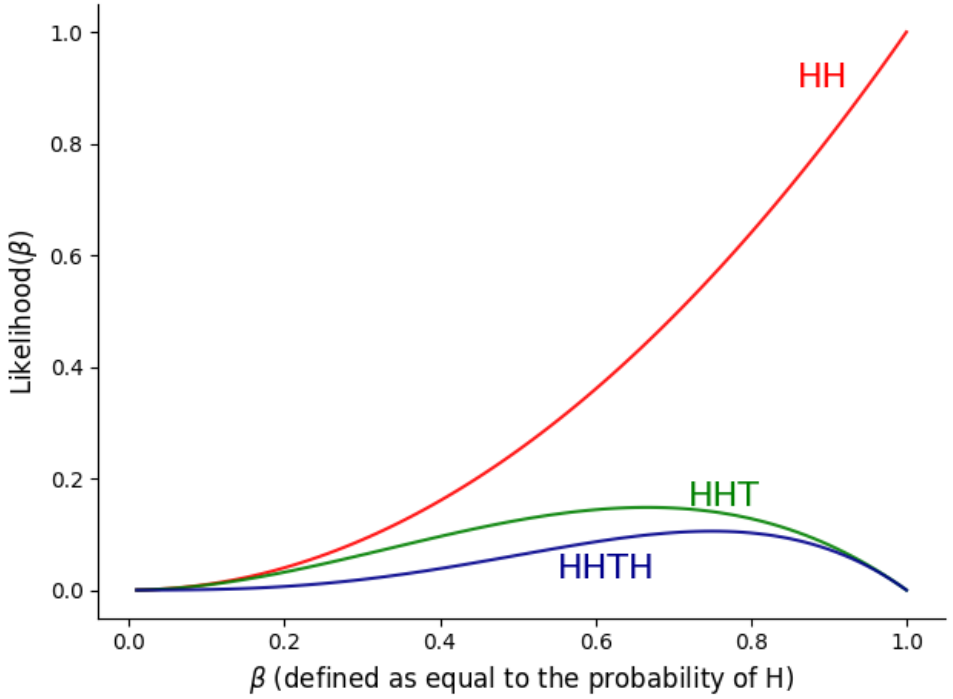
For three coin tosses producing HHT, likelihood is the green curve

For four producing HHTH, we get the blue curve.



Since we have a particular observation, it is reasonable to assume that β_{coin} corresponds to the **maximum of likelihood**.

For this value of β the stochastic process is the most likely to lead to the result we observed.



Notice how the result HH suggest that $\beta_{\text{coin}} = 1$, HHT suggests $\beta_{\text{coin}} \approx 0.65$ and HHTH suggests $\beta_{\text{coin}} \approx 0.79$

If our random variable is continuous, likelihood is defined as equal to the probability density:

$$\mathcal{L}(\beta|X = x) = p_X(x|\beta)$$

Note that on the left hand side x is not a variable, but a fixed outcome of random variable X.

Please do not confuse likelihood with probability.

It is wrong to say that $\mathcal{L}(\beta|\text{observation})$ is the probability that β has some value given an observation.

A brief overview of Gaussian Processes

Gaussian Process Regression

We have already learned a powerful and general regression tool: Kernel Ridge Regression (KRR).

Why do we need another regression model?

Gaussian Process Regression does what KRR, but also allows us to calculate:

$\Rightarrow$ Bayesian uncertainty of our predictions

As we shall see, this can be used, at least, for two powerful applications:

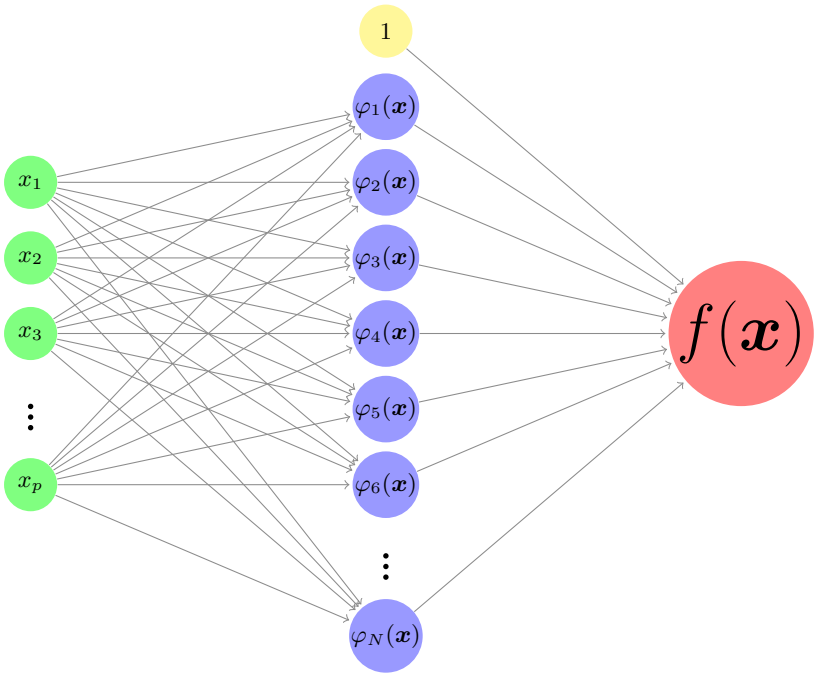
$\Rightarrow$ Bayesian optimization

$\Rightarrow$ Building better kernel functions

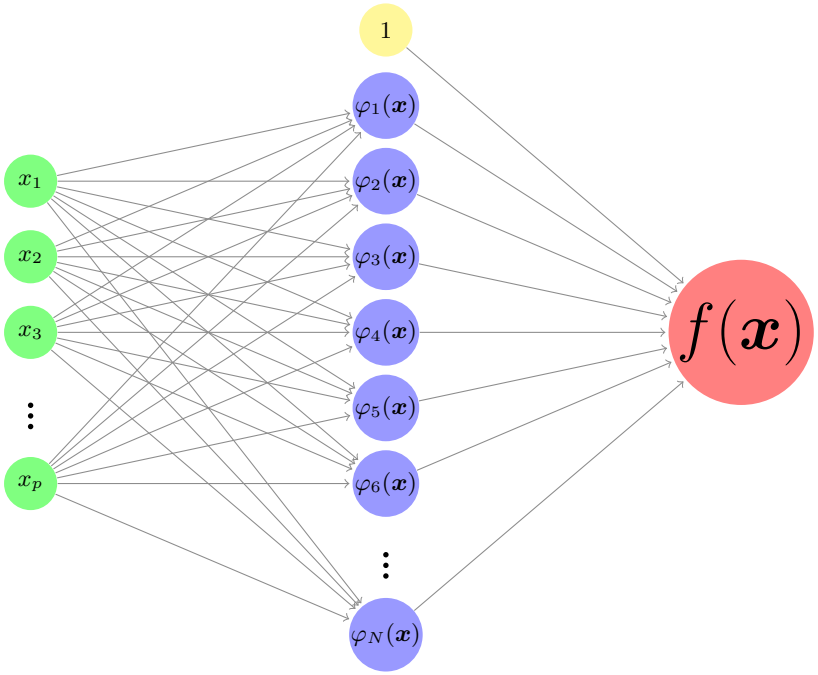
Consider the following model:

$$f(\boldsymbol{x}) = (\beta_0, \beta_1, \cdots, \beta_N) \begin{pmatrix} 1 \\ \varphi_1(\boldsymbol{x}) \\ \vdots \\ \varphi_N(\boldsymbol{x}) \end{pmatrix}$$

which can be depicted as a Neural Network:



If all of the parameters of the NN are fixed, then the NN



produces for each $\boldsymbol{x}$ a single value of $f(\boldsymbol{x})$.

Let us now replace each of $\beta_0, \beta_1, \cdots, \beta_p$ with a random variable, each distributed according to some distribution.

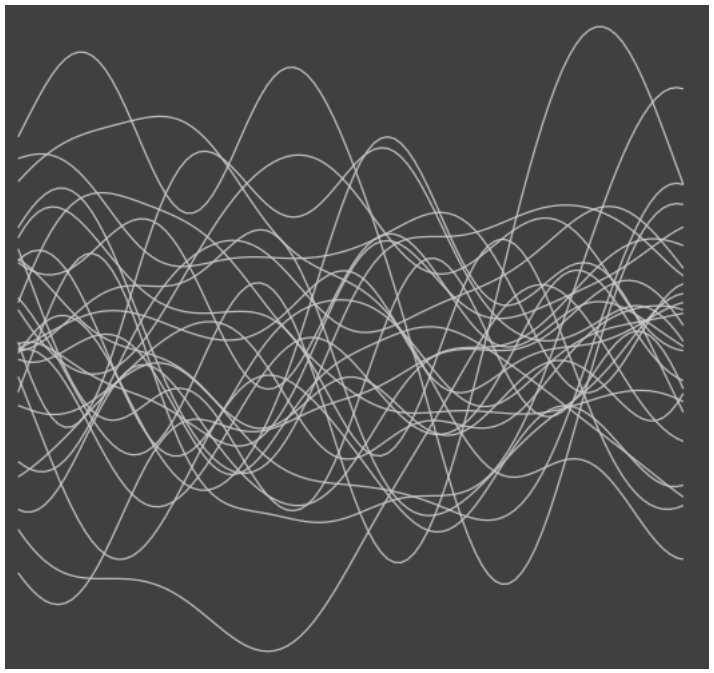
Since the variables are random, each time we draw a value for

$$\beta_0, \beta_1, \cdots, \beta_N$$

and evaluate the NN, it will produce a different, random result.

Thus, the output of the NN for a fixed $\boldsymbol{x}$ is a random variable.

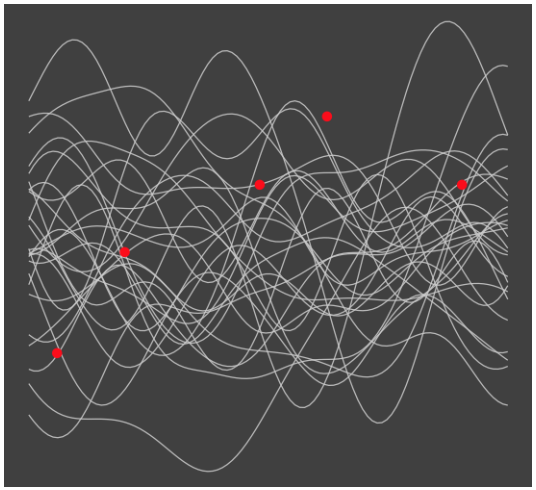
If we plot the output $f(\boldsymbol{x})$ of such a Neural Network as a function of $\boldsymbol{x}$, it will be something like this:



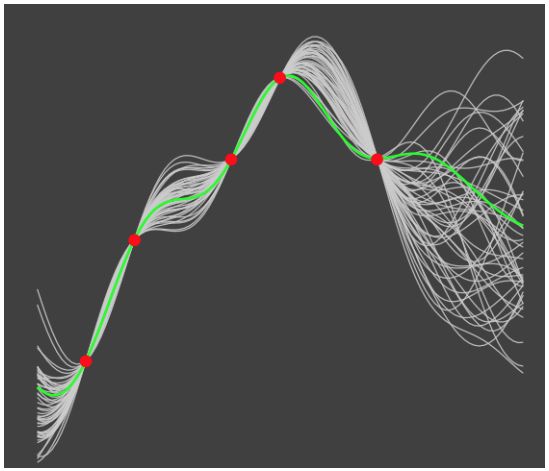
where each curve corresponds to a different combination of $\beta_0, \beta_1, \cdots, \beta_N$.

To model data,

We want to go from this:



To this:



We want to condition our distribution:

$$p(\theta|\mathbf{X}) \Rightarrow p(\theta|\{\mathbf{X}, \mathbf{y}\})$$

The Bayes' theorem gives us this conditional distribution:

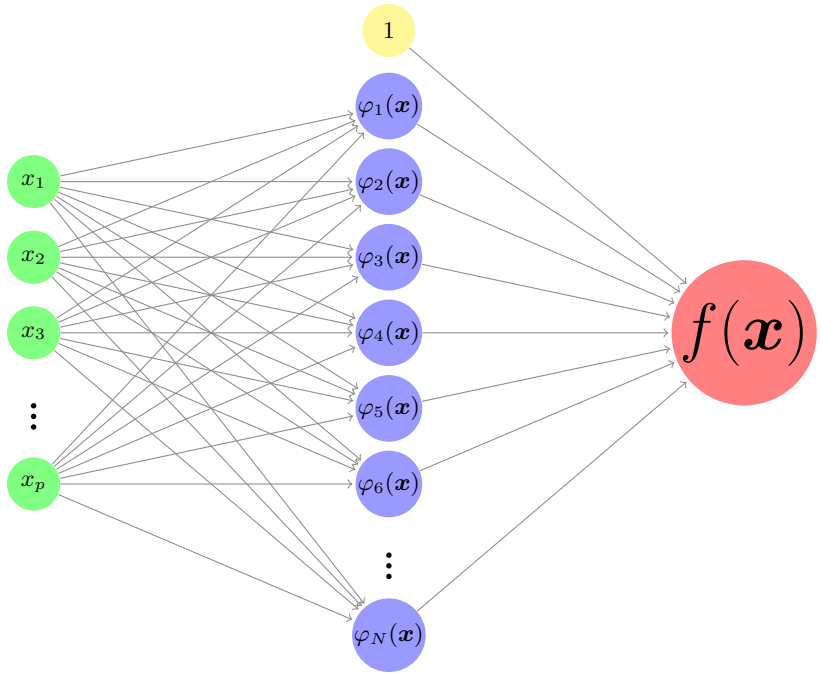
$$p(\theta|\{\mathbf{X}, \mathbf{y}\}) = \frac{p(\boldsymbol{y}|\theta, \mathbf{X})p(\theta|\mathbf{X})}{p(\boldsymbol{y}|\mathbf{X})}$$

The prediction at $\boldsymbol{x}^*$ can be made using:

$$p(y^*|\boldsymbol{x}^*, \{\mathbf{X}, \mathbf{y}\}) = \int_\theta p(y^*|\boldsymbol{x}^*, \theta)p(\theta|\{\mathbf{X}, \mathbf{y}\})d\theta$$

Let's *increase* the size of the NN by making $N \to \infty$.

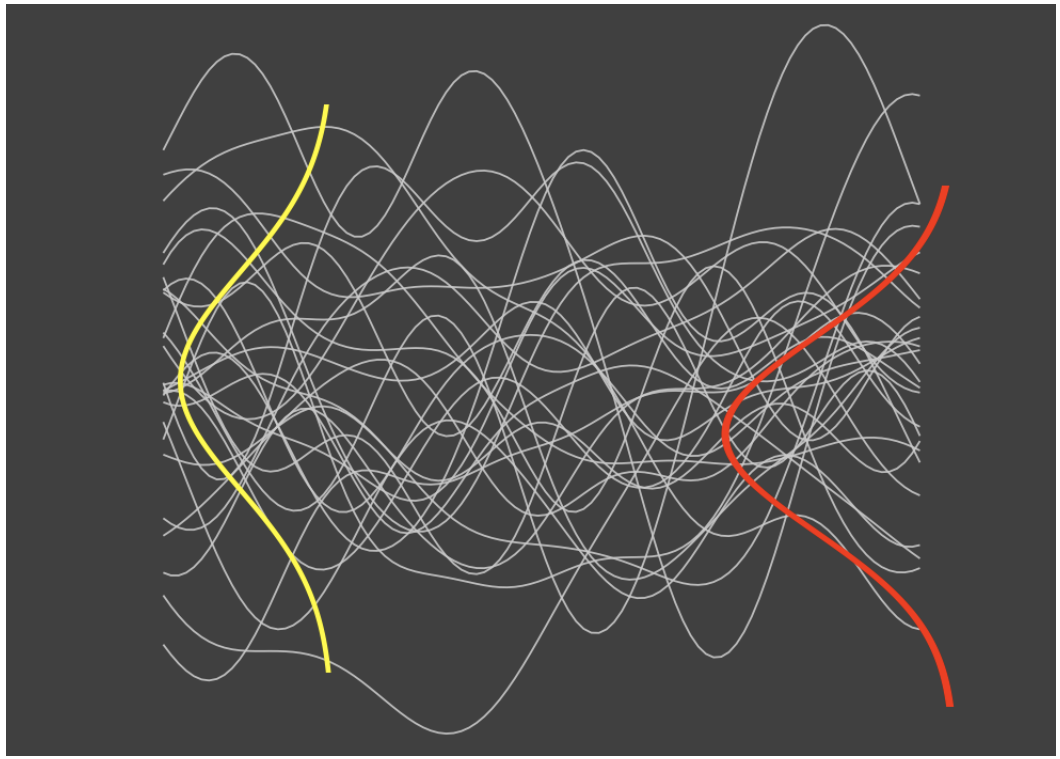
If each of the inputs into the output (red) neuron in the NN



are independent random variables and $N \to \infty$, what is the distribution of the output values for fixed $\boldsymbol{x}$?

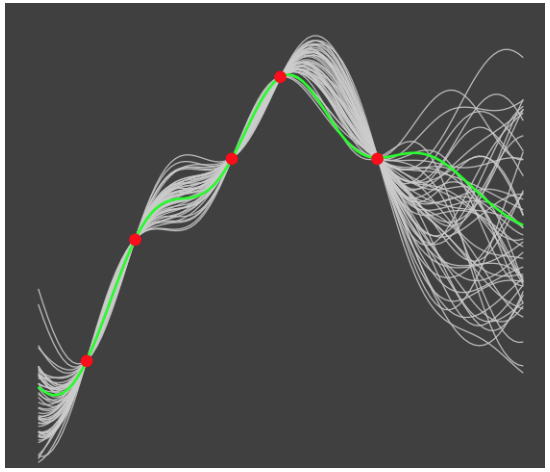
The central limit theorem says this distribution will be Gaussian.

This distribution must remain Gaussian for all $\boldsymbol{x}$.



Thus, $f(\boldsymbol{x})$ is a **Gaussian process**.

This helps a lot because the integrals on the previous page can be dealt with analytically!



$$\hat{\mu}(\boldsymbol{x}^*) = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\left[\mathbf{K} + \sigma^2\mathbf{I}\right]^{-1}\boldsymbol{y}$$

$$\hat{\sigma}^2(\boldsymbol{x}^*) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\left[\mathbf{K} + \sigma^2\mathbf{I}\right]^{-1}\boldsymbol{k}(\boldsymbol{x}^*)$$

We see that everything is determined by the kernel functions $K(\boldsymbol{x}, \boldsymbol{x'})$.

Final results for Gaussian Process regression:

Mean of the predictive distribution to be used as the prediction of the model:

$$\hat{\mu} = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*) \left[\mathbf{K} + \sigma^2 \mathbf{I}\right]^{-1} \boldsymbol{y}$$

Conditional variance to be used as Bayesian uncertainty:

$$\hat{\sigma}^2 = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*) \left[\mathbf{K} + \sigma^2 \mathbf{I}\right]^{-1} \boldsymbol{k}(\boldsymbol{x}^*)$$

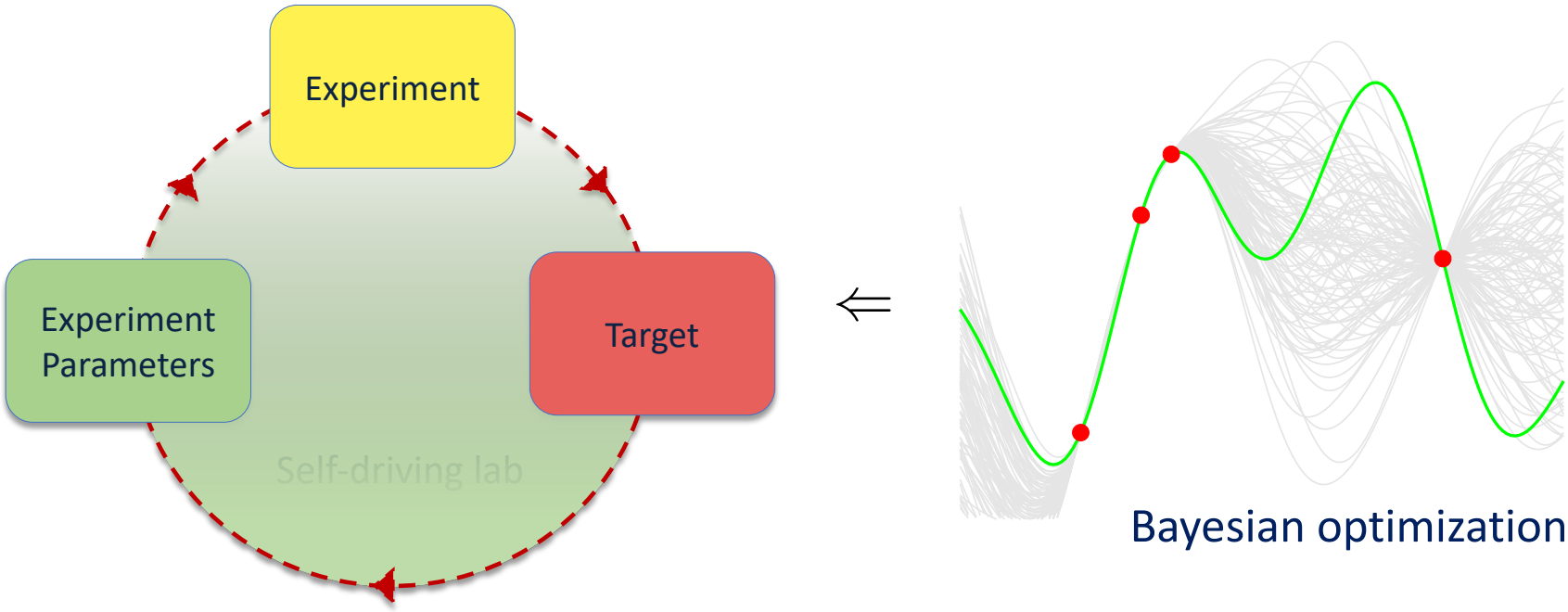
Compare this with what we obtained for Kernel Ridge Regression:

$$\hat{f}(\boldsymbol{x}^*) = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\boldsymbol{\alpha} = \\ = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*) \left[\mathbf{K} + \lambda \mathbf{I}\right]^{-1} \boldsymbol{y}$$

The prediction error in GPR can be used for Bayesian optimization

$$\hat{\sigma}^2 = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*) \left[\mathbf{K} + \sigma^2 \mathbf{I}\right]^{-1} \boldsymbol{k}(\boldsymbol{x}^*)$$

Optimization of expensive black-box functions

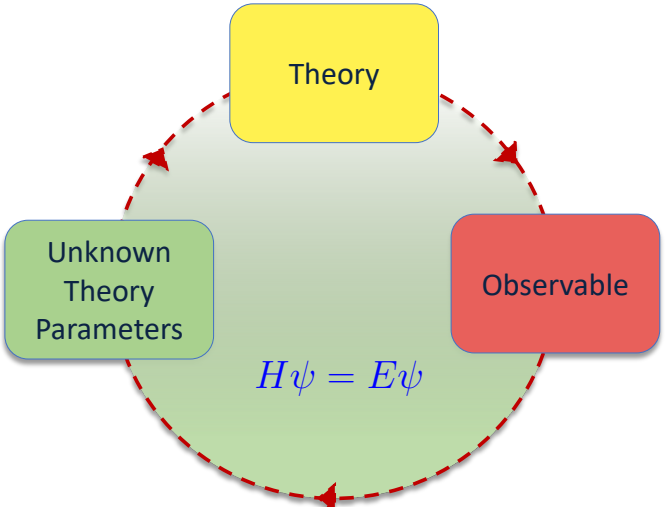


Experimental parameters $\Rightarrow \mathbf{x}$

Outcome of experiment $\Rightarrow y$

Surrogate ML model $\mathcal{A}$ of the experiment outcome $\Rightarrow$ Optimize $\mathcal{A}(\boldsymbol{x})$

Many applications in Chemistry – e.g. solving inverse problems



PCCP

PERSPECTIVE

View Article Online
View Journal | View Issue

Check for updates

Cite this: *Phys. Chem. Chem. Phys.*, 2019, **21**, 13392

Bayesian machine learning for quantum molecular dynamics

R. V. Krems

Bayesian Optimization (BO)

As mentioned earlier, BO is designed to optimize black-box (e.g. unknown) functions without gradients.

Therefore, it is particularly well suited for functions that are very difficult to evaluate.

How does it work? BO uses Gaussian processes as surrogate models of the black-box function.

Recall that training a Gaussian Process produces:

$$\hat{\mu} = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\left[\mathbf{K} + \sigma^2\mathbf{I}\right]^{-1}\boldsymbol{y} \quad \Leftarrow \quad \text{prediction}$$

$$\hat{\sigma}^2 = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*)\left[\mathbf{K} + \sigma^2\mathbf{I}\right]^{-1}\boldsymbol{k}(\boldsymbol{x}^*) \quad \Leftarrow \quad \text{uncertainty}$$

BO is an iterative process.

BO starts with a few evaluations of the black box function at random places. The evaluations are used to train a Gaussian Process.

The results of the training are used to build an acquisition function:

$$\alpha(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \kappa\hat{\sigma}^2(\boldsymbol{x})$$

At each iteration:

$\Rightarrow$ The acquisition function is maximized to find $\boldsymbol{x}_{\max}$.

$\Rightarrow$ The black-box function is evaluated at $\boldsymbol{x}_{\max}$.

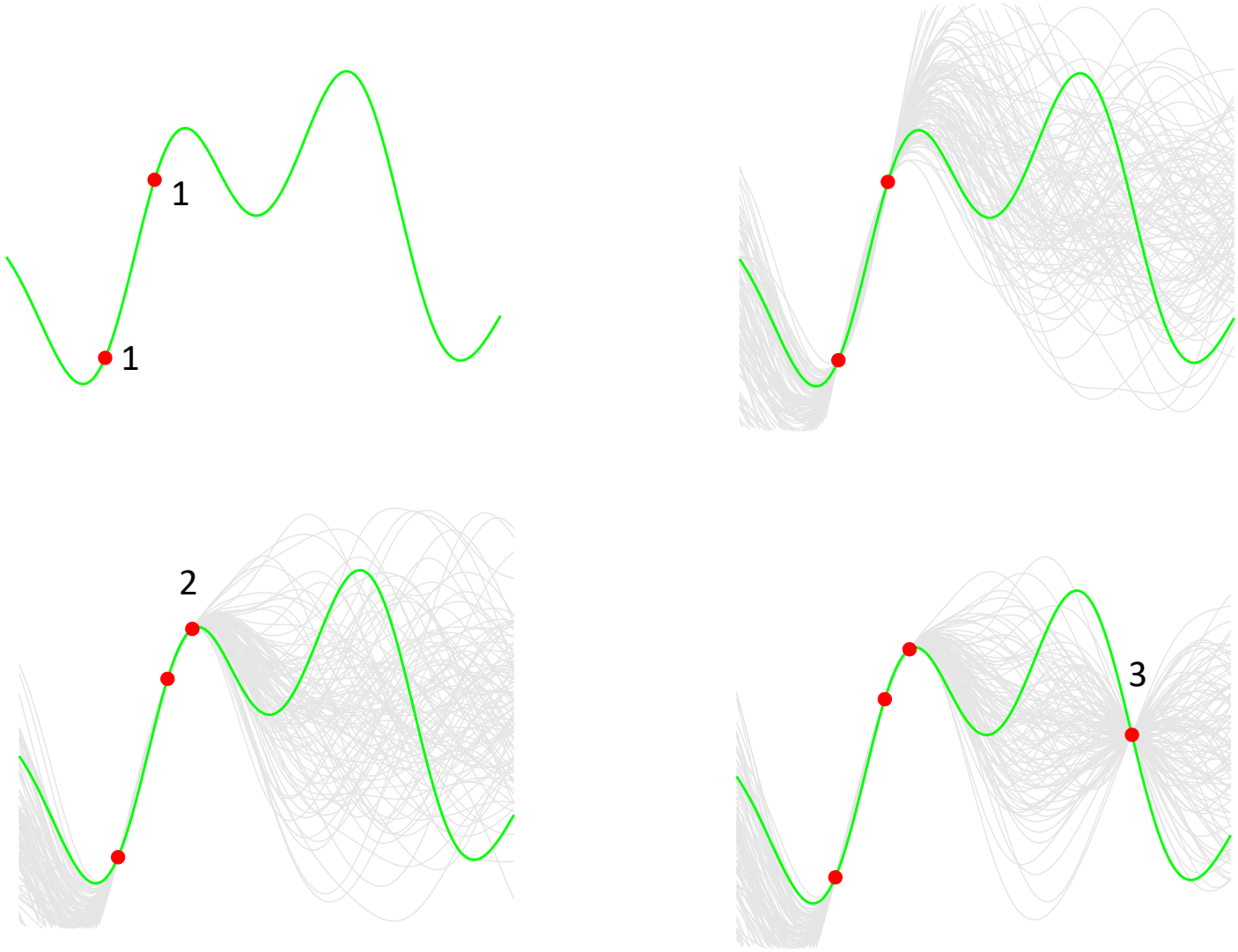
$\Rightarrow$ The result of the evaluation is added to the training set for GP.

$\Rightarrow$ A new (more accurate) GP is trained.

$\Rightarrow$ A new acquisition function is built from the new GP.

$\Rightarrow$ A new value of $\boldsymbol{x}_{\max}$ is determined.

This is how Bayesian Optimization works:



New Journal of Physics
The open access journal at the forefront of physics

FAST TRACK COMMUNICATION • OPEN ACCESS

Bayesian optimization for the inverse scattering problem in quantum reaction dynamics

R A Vargas-Hernández[1], Y Guan[2,3], D H Zhang[2,3] and R V Krems[1]

Published 28 February 2019 • © 2019 The Author(s). Published by IOP Publishing Ltd on behalf of the Institute of Physics and Deutsche Physikalische Gesellschaft

New Journal of Physics, Volume 21, February 2019

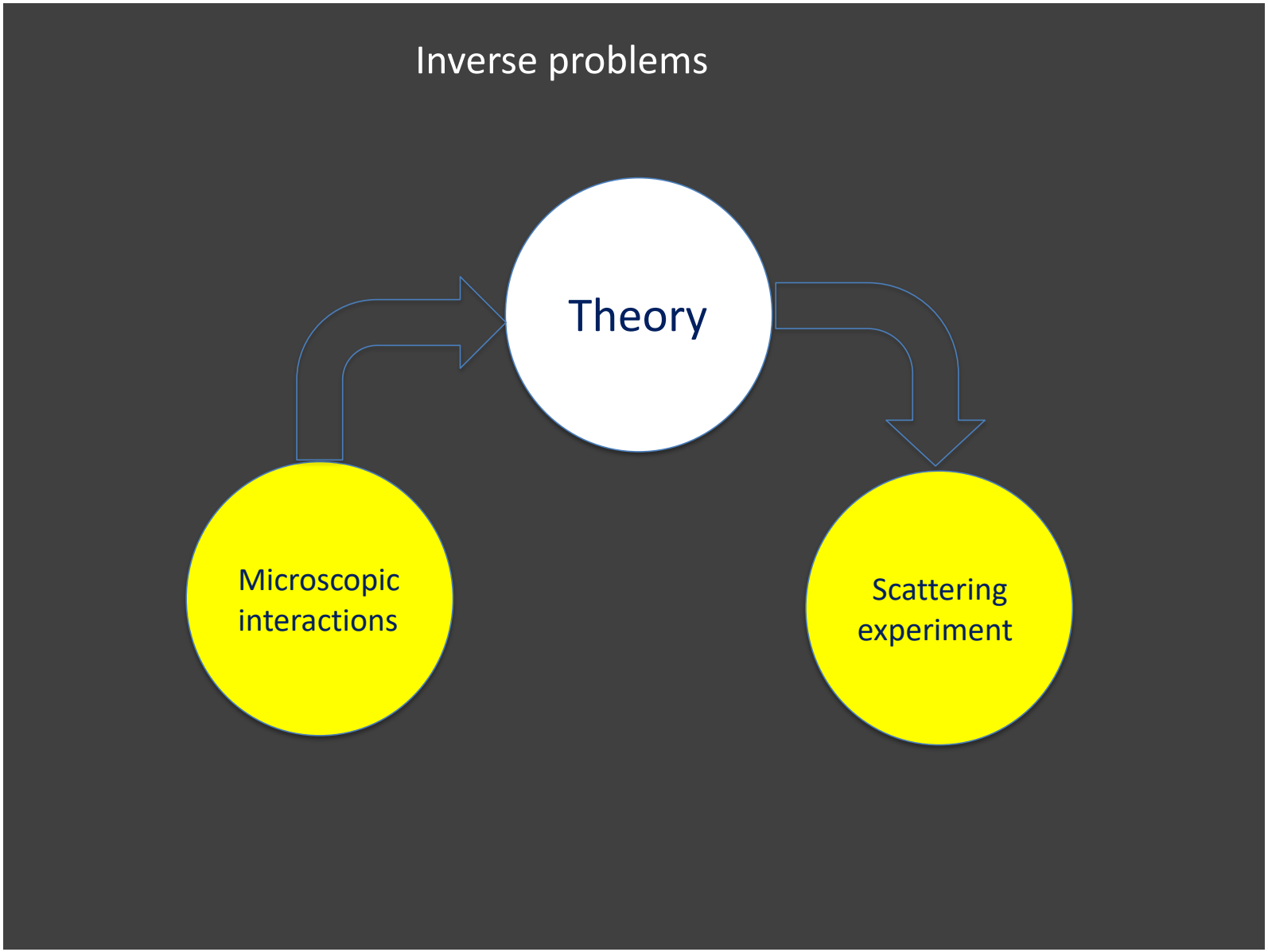
Citation R A Vargas-Hernández *et al* 2019 *New J. Phys.* **21** 022001

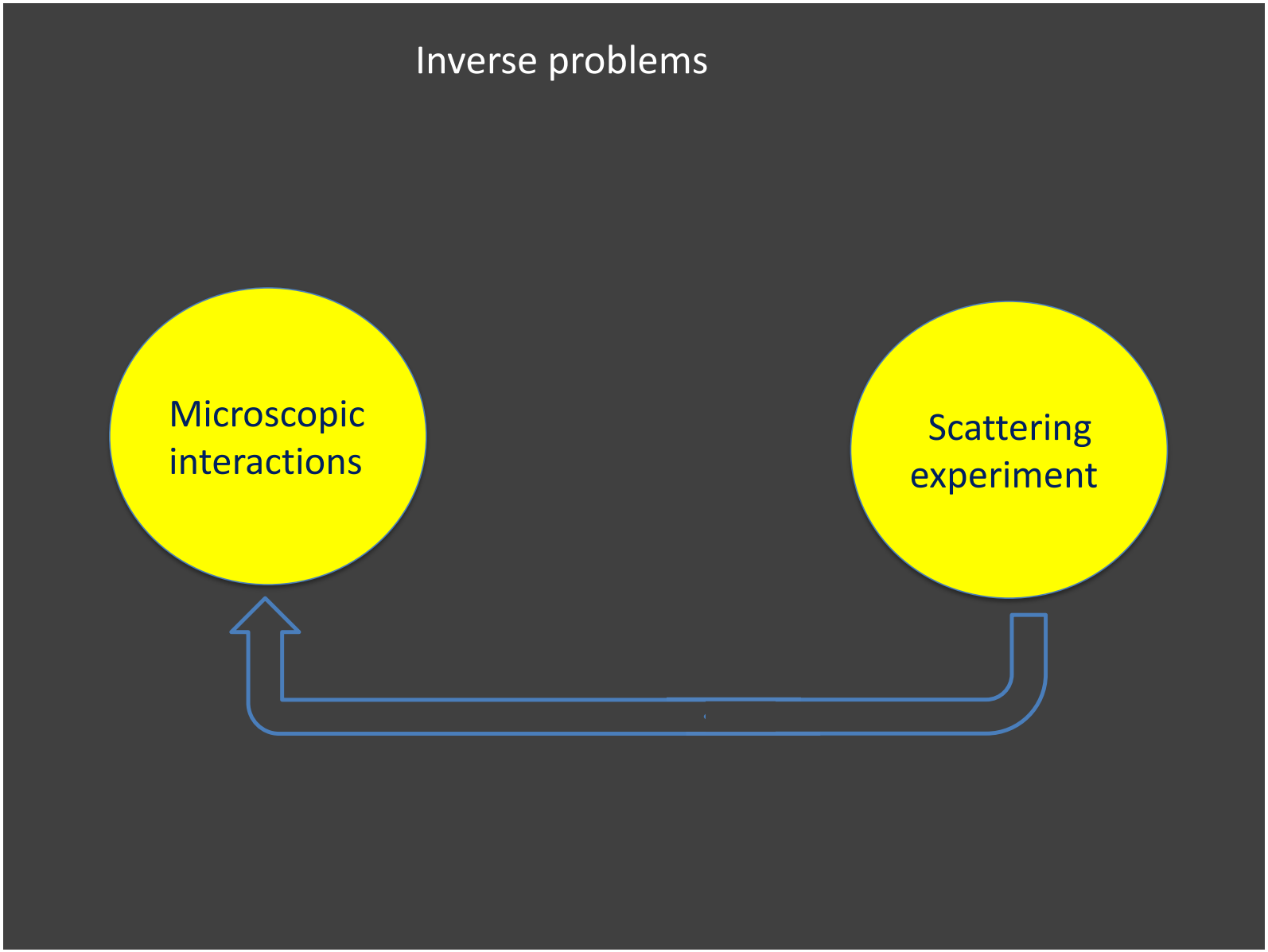
The Journal of Chemical Physics ARTICLE scitation.org/jo

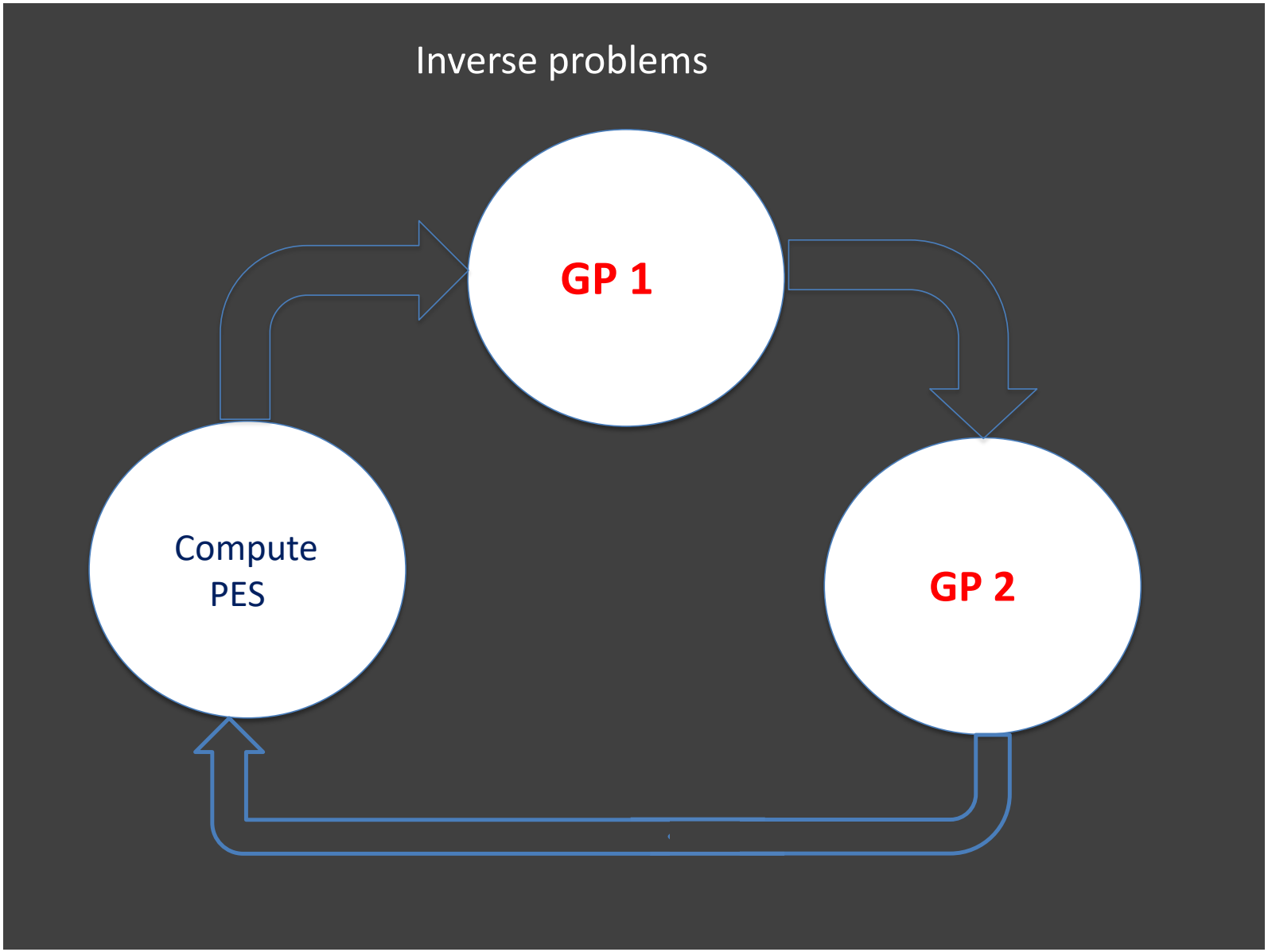
Bayesian optimization for inverse problems in time-dependent quantum dynamics

Cite as: J. Chem. Phys. 153, 164111 (2020); doi: 10.1063/5.0015896
Submitted: 30 May 2020 • Accepted: 7 October 2020 •
Published Online: 26 October 2020

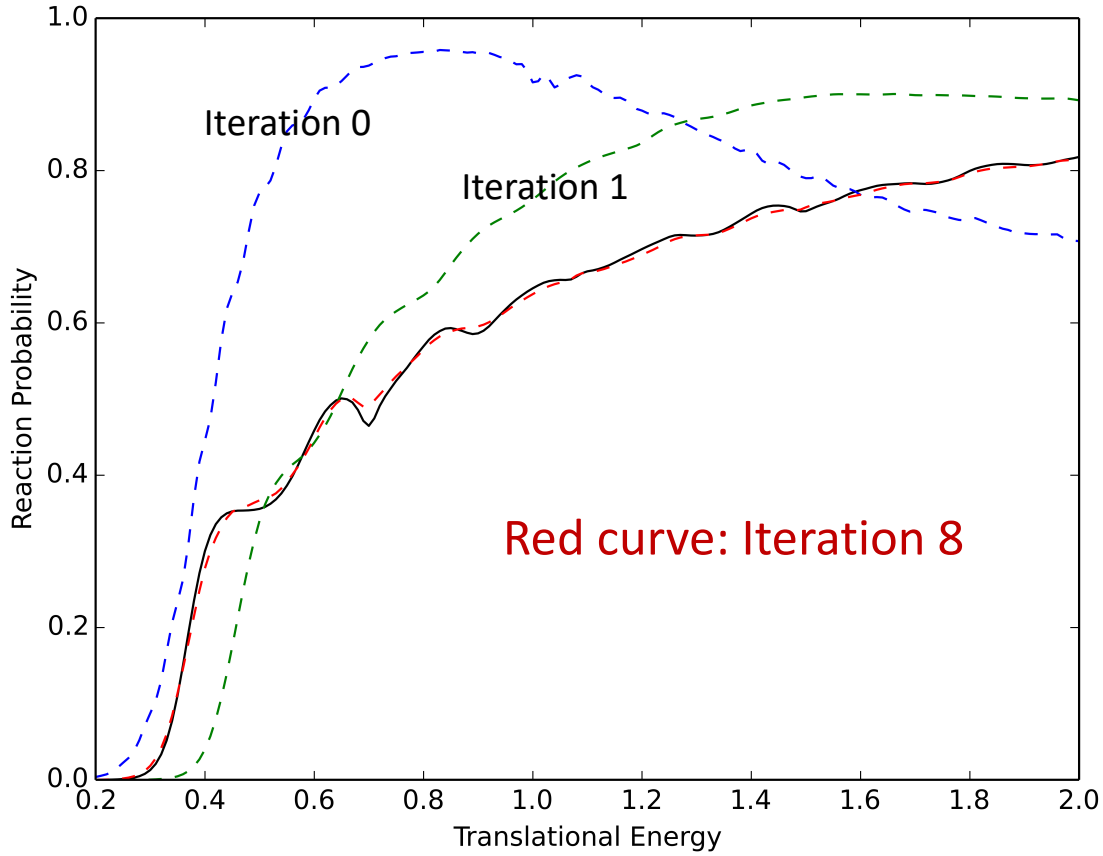
Z. Deng,[1] I. Tutunnikov,[2] I. Sh. Averbukh,[2] M. Thachuk,[1] and R. V. Krems[1,3,a)]


Inverse problems
Theory
Microscopic interactions
Scattering experiment

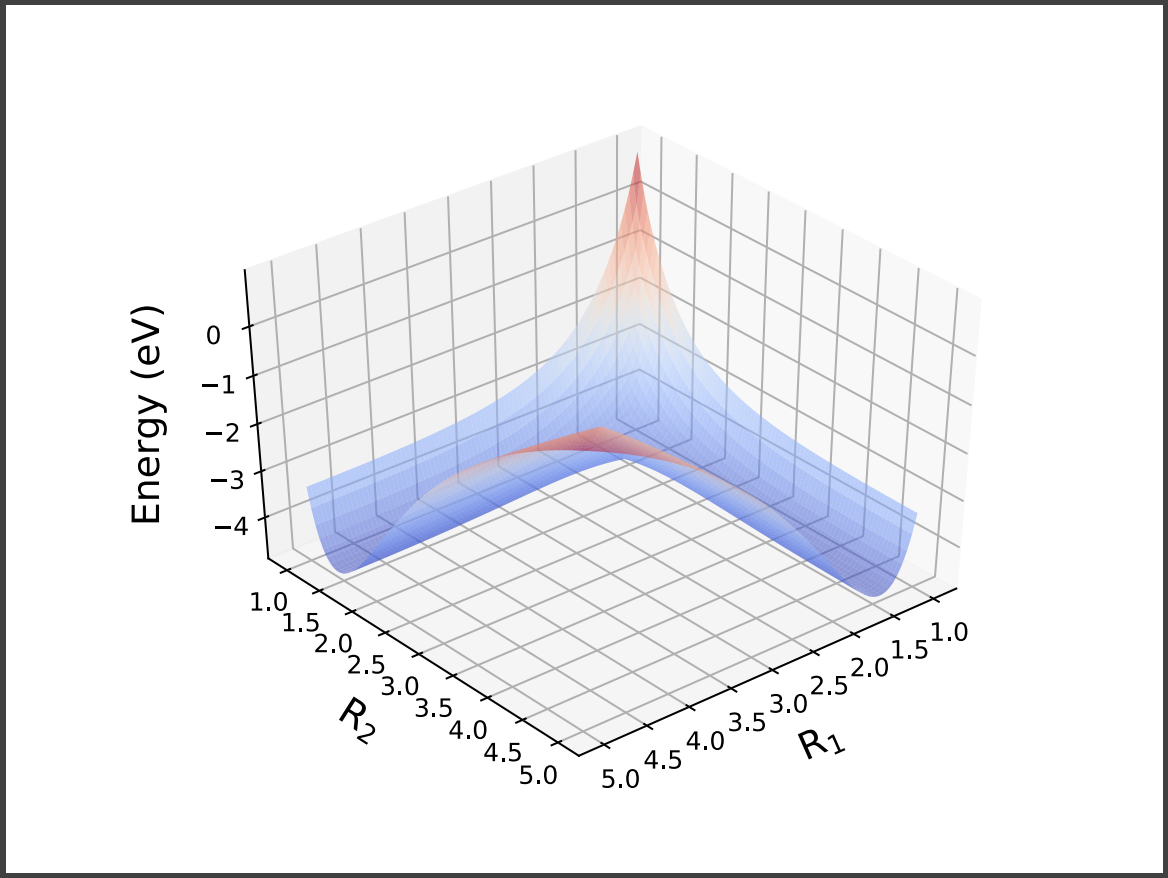

Inverse problems
Microscopic interactions
Scattering experiment

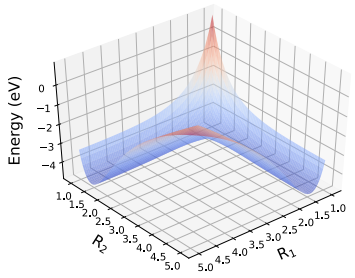

Inverse problems
GP 1
Compute
PES
GP 2

$$H + H_2 \longrightarrow H_2 + H$$

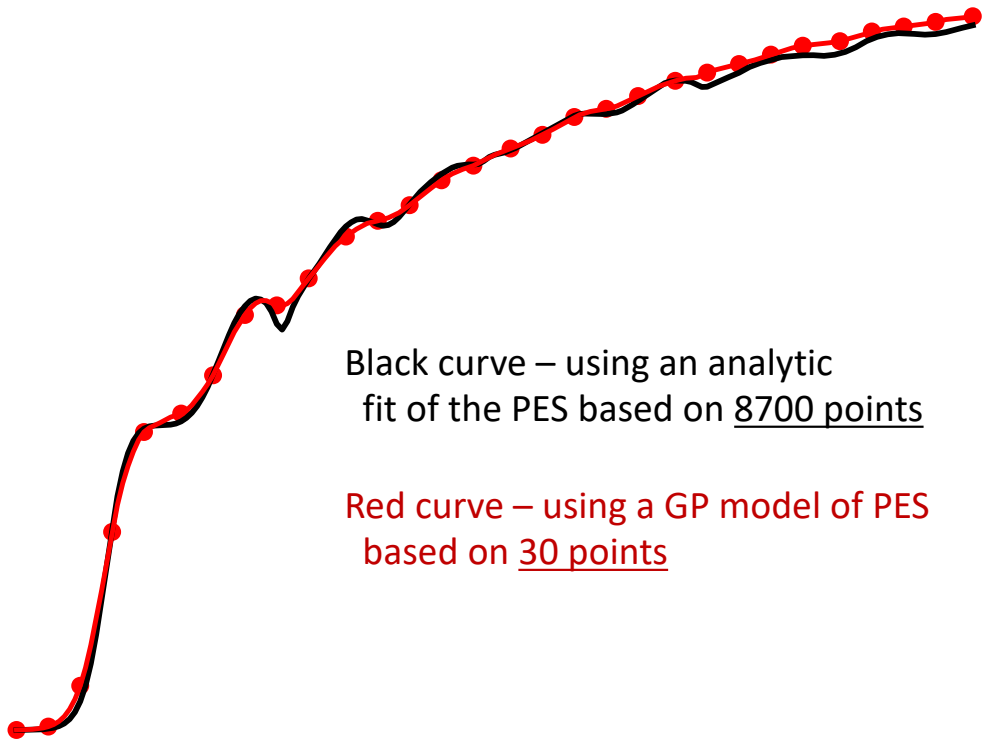


How many potential energy points does one need to know to describe the complete surface?



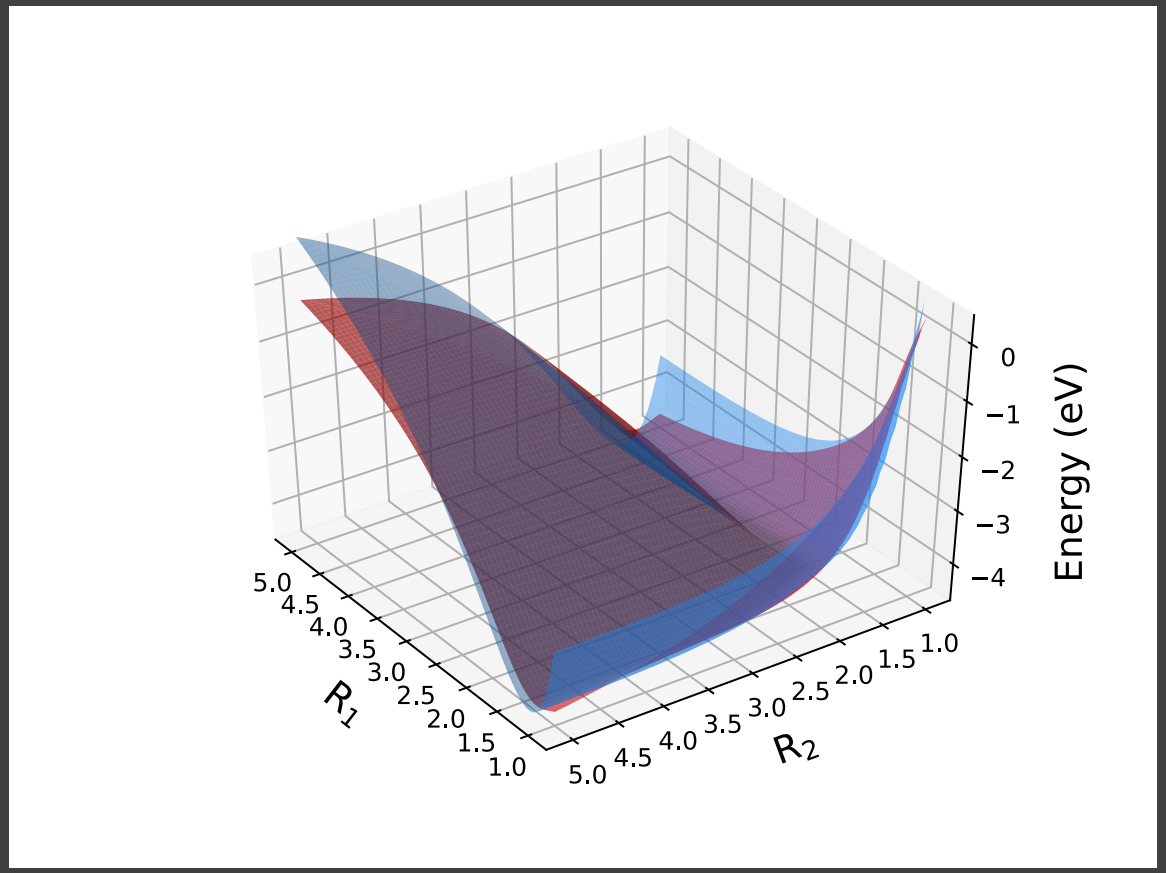


Quantum scattering calculation



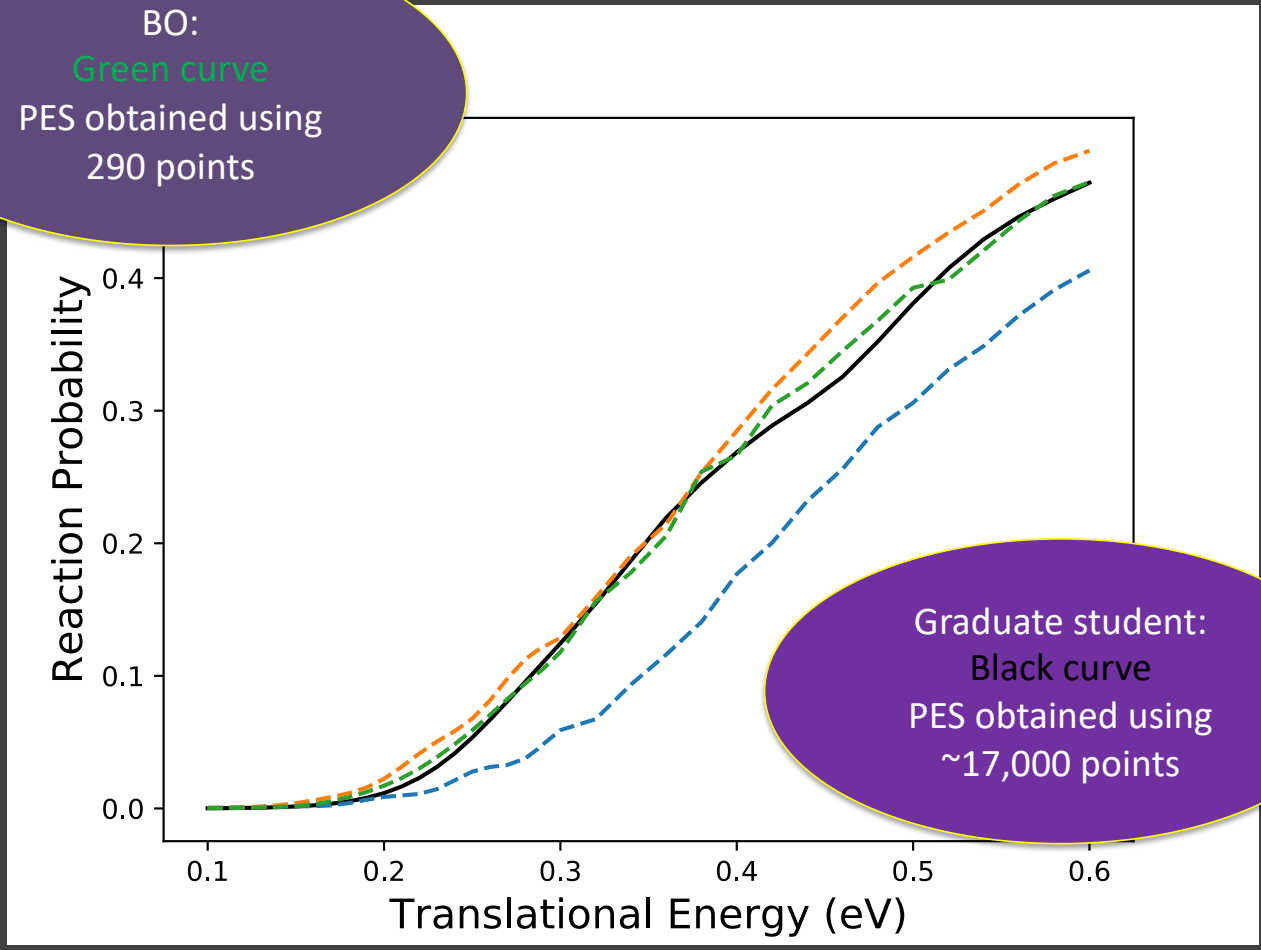
Black curve – using an analytic fit of the PES based on 8700 points

Red curve – using a GP model of PES based on 30 points



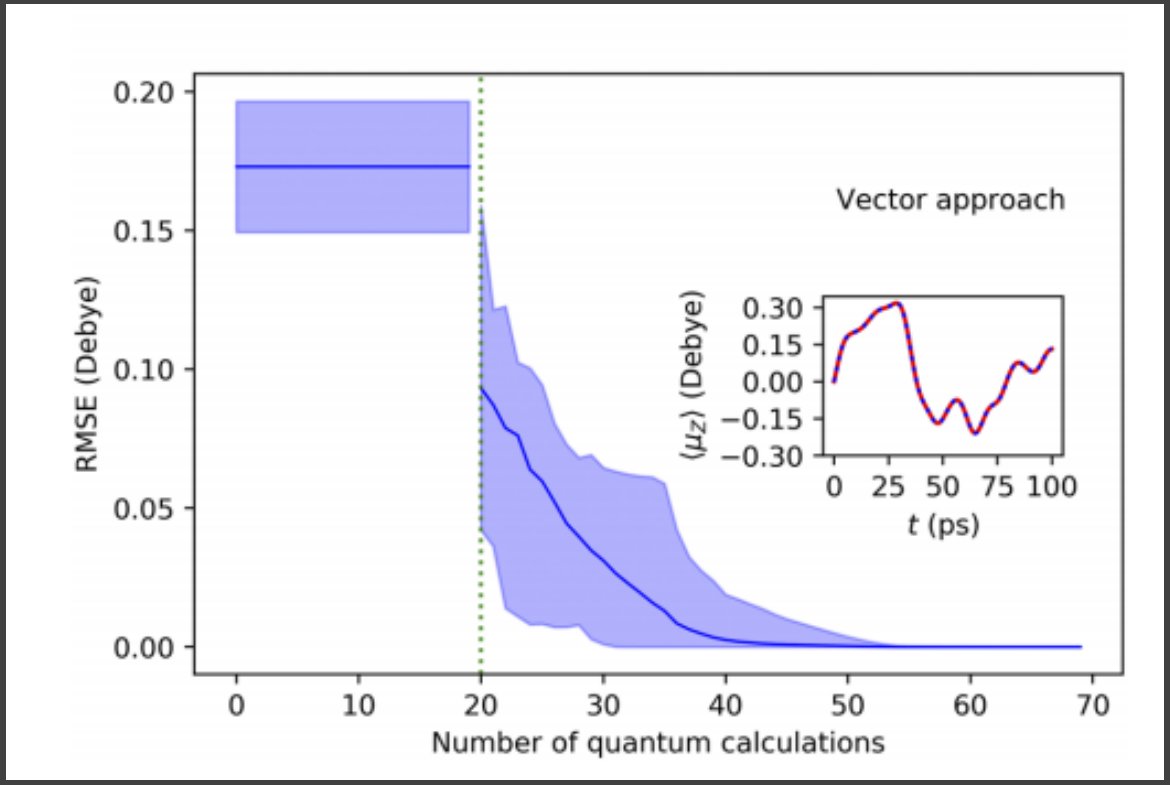
At every iteration, a completely new global surface is constructed automatically

BO:
Green curve
PES obtained using
290 points



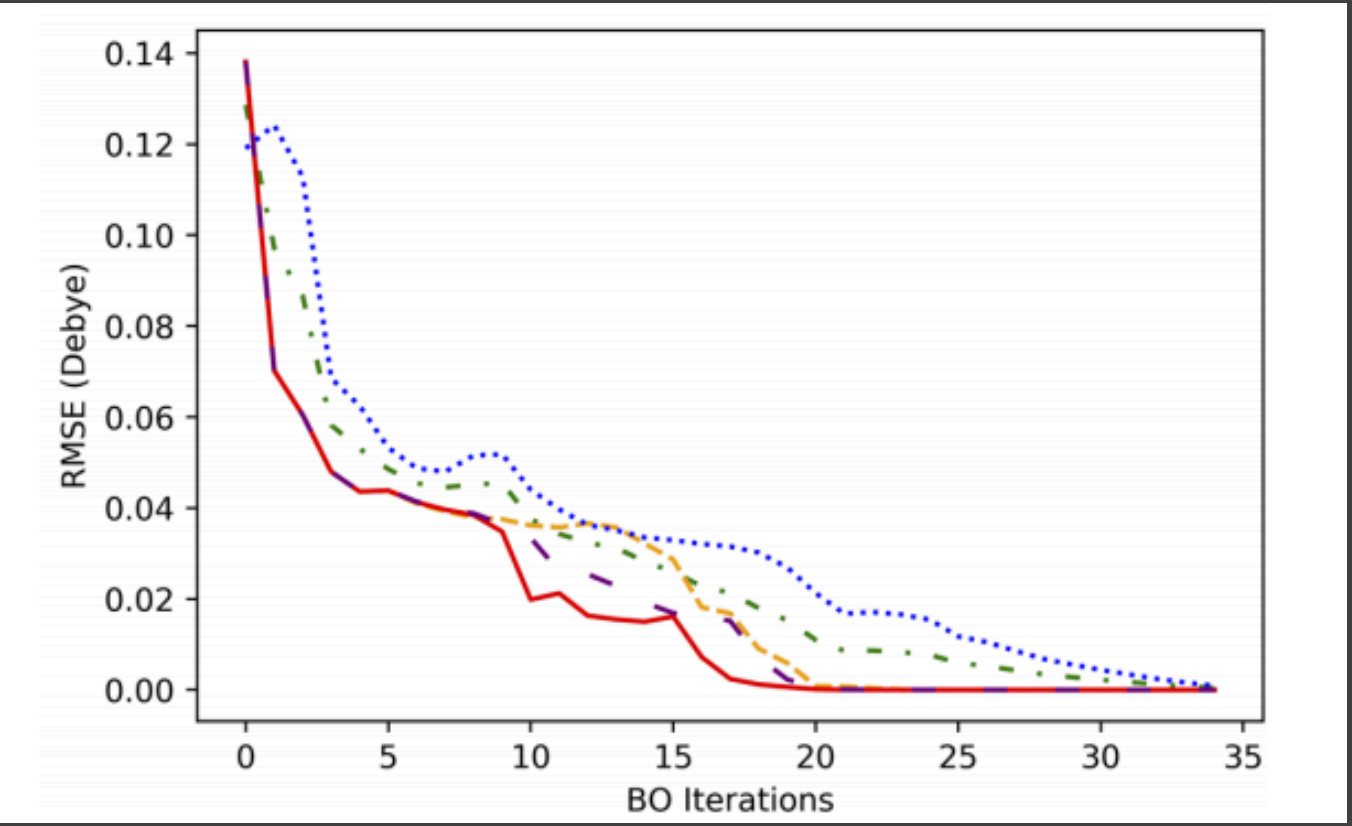
Graduate student:
Black curve
PES obtained using
~17,000 points

Example of a 6D system: OH + H_2 chemical reaction



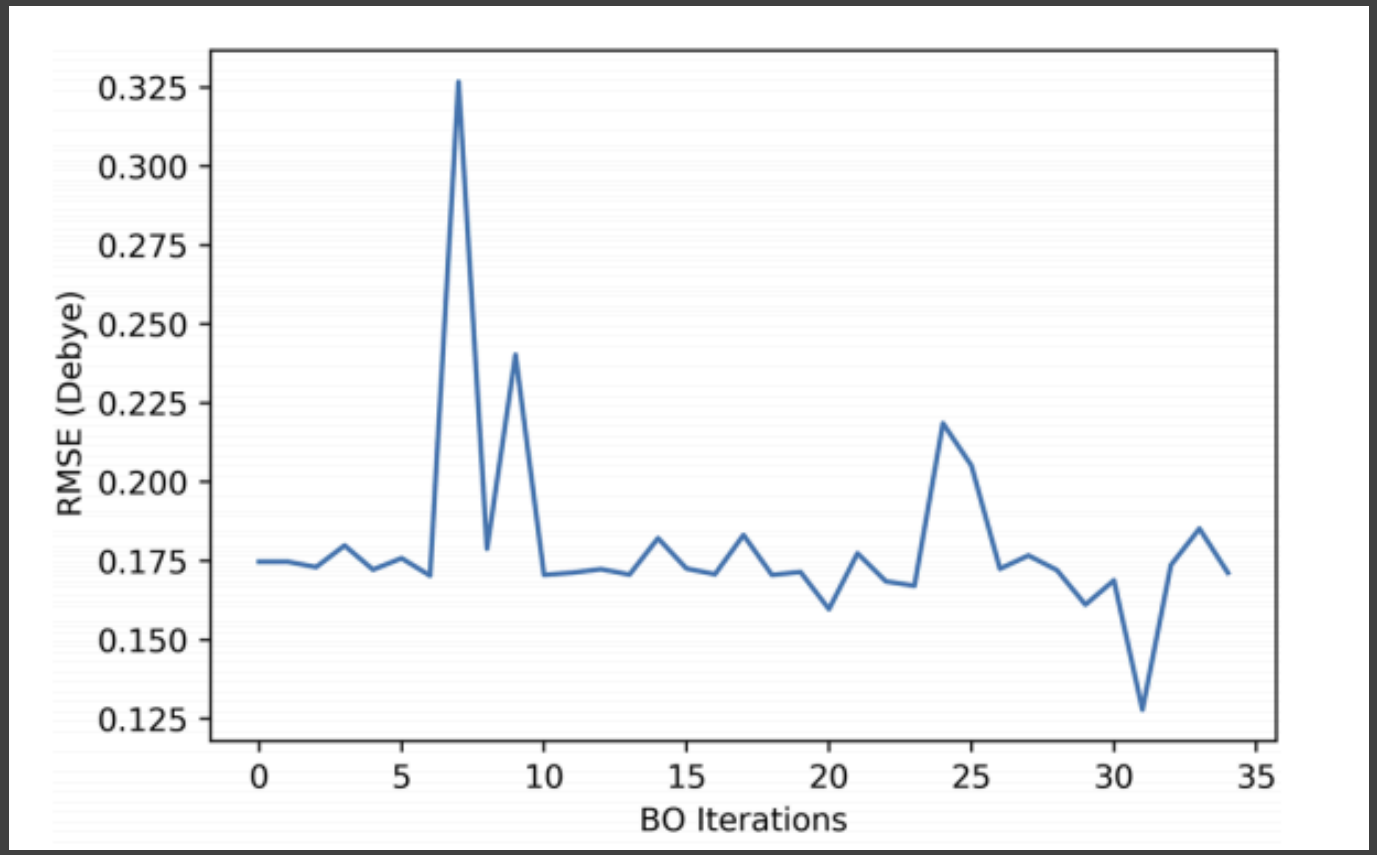
Inferring molecular properties from time-dependent observables

The choice of kernel is important!



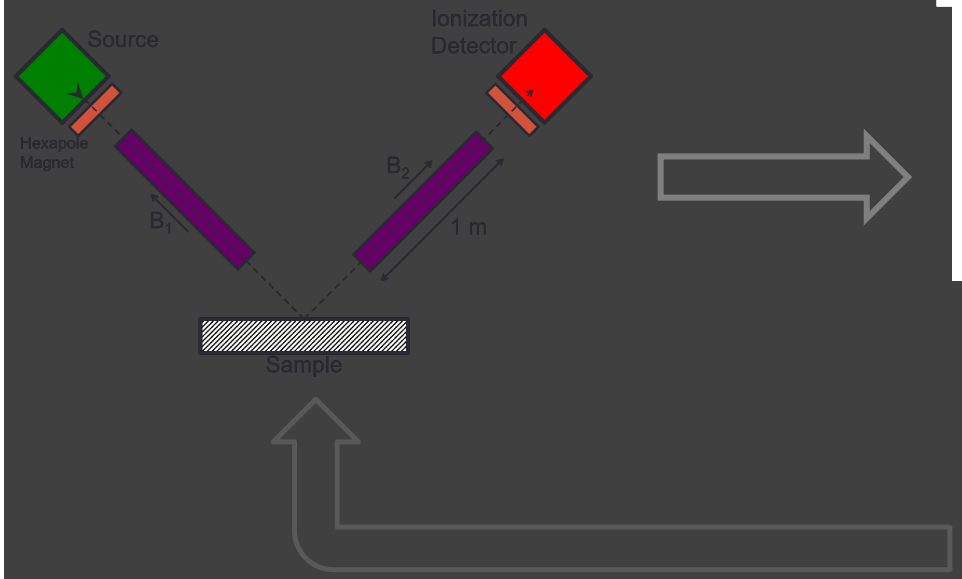
Inferring molecular properties from time-dependent observables

The choice of kernel is important!

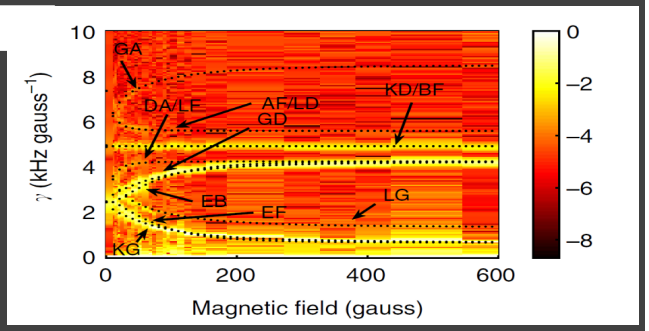


This is what happens with a poorly chosen kernel

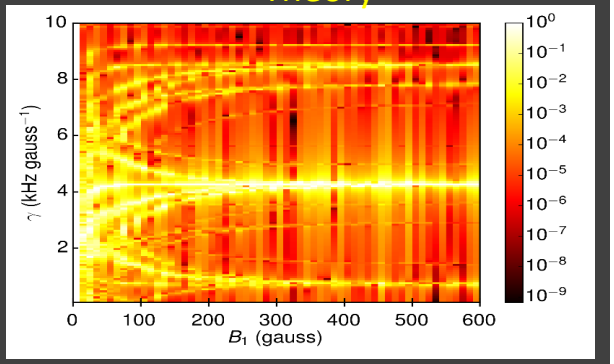
Molecular Hyperfine Interferometry



Experiment



Theory



Godsi, O.; Corem, G.; Alkoby, Y.; Cantin, J. T.; Krems, R. V.; Somers, M. F.; Meyer, J.; Kroes, G.-J.; Maniv, T.; and Alexandrowicz, G. *Nature Communications* 8, 1537 (2017).

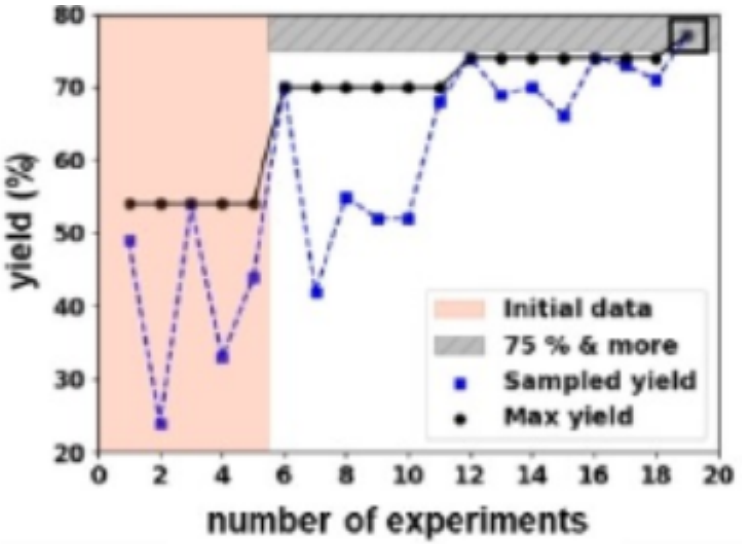
Chemistry—Methods

Full Papers
doi.org/10.1002/cmtd.202100053

Chemistry Europe
European Chemical Societies Publishing

Rapid and Mild One-Flow Synthetic Approach to Unsymmetrical Sulfamides Guided by Bayesian Optimization

Naoto Sugisawa,[a] Hiroki Sugisawa,[b] Yuma Otake,[c] Roman V. Krems,[d, e] Hiroyuki Nakamura,[f] and Shinichiro Fuse*[a]



Interplay of Machine Learning and Quantum Theory

Roman Krems

University of British Columbia, Vancouver, Canada

August 26, 2021

Part III:

- Model selection metrics
  Bayesian Information Criterion
- How to design the best kernel?
- Extrapolation in Hamiltonian Parameter Spaces
- Quantum Machine Learning
- How to build the best quantum kernels

Mean: $\mu = \frac{1}{n}\sum_{i=1}^{n} X_i$.

Expected value: $E[X] = \frac{1}{n}\sum_{i=1}^{n} X_i$

Variance: $\sigma_X^2 = E[(X - \mu)^2]$

Covariance:

$$\mathrm{Cov}(X, Y) = E[(X - E(X))(Y - E(Y))] = E[(X - \mu_x)(Y - \mu_Y)]$$

How are Gaussian Processes trained?

- As before, one starts by choosing a function for $K(\boldsymbol{x}, \boldsymbol{x'})$.
- The parameters of this function are then varied to maximize $p(\boldsymbol{y}|\mathbf{X})$.
- Recall $p(\boldsymbol{y}|\mathbf{X})$.. This is the term in the denominator of the Bayes' theorem:

$$p(\boldsymbol{y}|\mathbf{X}) = \int_{\theta} p(\boldsymbol{y}|\theta, \mathbf{X}) p(\theta|\mathbf{X}) d\theta$$

This multiplies the model likelihood $p(\boldsymbol{y}|\theta, \mathbf{X})$ by $p(\theta|\mathbf{X})$ and integrates over all random function values.

When we integrate over a variable as θ above, we are 'marginalizing' this variable.

The likelihood integrated as above is thus called 'marginal likelihood'.

Marginal likelihood

Marginal likelihood for a Gaussian process can be expressed in terms of kernels. – How?

To answer this questions, let's consider again the prior $p(\theta|\mathbf{X})$.

I remind you that

θ represents functions drawn from a Gaussian process

$p(\theta|\mathbf{X})$ is the unconditional distribution of the NN outputs at the locations of the input space given by $\mathbf{X}$

Because the functions θ are drawn from a Gaussian process, $p(\theta|\mathbf{X})$ is a multi-variate Gaussian.

This means that $p(\theta|\boldsymbol{x}_i)$ and $p(\theta|\boldsymbol{x}_j)$ are both Gaussian with some covariance $\mathrm{Cov}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ between them.

We will choose the covariance of the Gaussian Process prior to be the kernel function:

$$\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{x'}) = K(\boldsymbol{x}, \boldsymbol{x'})$$

This is actually equivalent to the choice of the prior we had previously made:

$$p(\boldsymbol{\beta}) \sim \mathcal{N}(0, \boldsymbol{\Sigma}_p)$$

To see this, consider

$$f(\boldsymbol{x}) = \boldsymbol{\beta}^T \boldsymbol{\varphi}(\boldsymbol{x})$$

Then:

$E[f(\boldsymbol{x})] = \boldsymbol{\varphi}(\boldsymbol{x})^{\mathrm{T}} E[\boldsymbol{\beta}] = 0 \quad$ and

$\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{x'}) = E[f(\boldsymbol{x}) f(\boldsymbol{x'})] = \boldsymbol{\varphi}^{\mathrm{T}}(\boldsymbol{x}) E[\boldsymbol{\beta}\boldsymbol{\beta}^{\mathrm{T}}] \boldsymbol{\varphi}(\boldsymbol{x'}) = \boldsymbol{\varphi}^{\mathrm{T}}(\boldsymbol{x}) \boldsymbol{\Sigma}_p \boldsymbol{\varphi}(\boldsymbol{x'})$

Our goal is to evaluate the marginal likelihood:

$$p(\boldsymbol{y}|\mathbf{X}) = \int_{\theta} p(\boldsymbol{y}|\theta, \mathbf{X}) p(\theta|\mathbf{X}) d\theta$$

It is more convenient to work with the logarithm of marginal likelihood.

Given that the prior's covariance is the kernel matrix and the integrand is a product of two Gaussians, it is actually possible to write this in terms of the kernel matrix as follows:

$$\log p(\boldsymbol{y}|\mathbf{X}) = -\frac{1}{2}\boldsymbol{y}^{\mathrm{T}} \left(\mathbf{K} + \sigma^2 \mathbf{I}\right) \boldsymbol{y} - \frac{1}{2} \log |\mathbf{K} + \sigma^2 \mathbf{I}| - \frac{n}{2} \log 2\pi$$

The purpose of training a Gaussian process is to find the parameters of the kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$ that maximizes the logarithm of marginal likelihood.

Key difference between GPR and KRR:

The parameters of the kernel function in KRR are found by cross-validation.

The parameters of the kernel function for GPR are found by maximizing marginal likelihood.

GPR model prediction:

$$\hat{\mu} = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*) \left[\mathbf{K} + \sigma^2 \mathbf{I}\right]^{-1} \boldsymbol{y}$$

KRR model predicition:

$$\hat{f}(\boldsymbol{x}^*) = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}^*) \left[\mathbf{K} + \lambda \mathbf{I}\right]^{-1} \boldsymbol{y}$$

Model selection

Let's say we have built two models with two different kernels.

How to tell which one is better?

The Bayesian approach gives us a way...

Model selection

- Consider a set of noisy data points.
- Consider two different models, such as, for example:

$$\mathcal{M}_1 \quad \text{and} \quad \mathcal{M}_2$$

- How can we tell which model is better?
- We will adopt the Bayesian view to answer this question.

- Within the Baysian approach, we have for model $\mathcal{M}_i$:

$$P(\mathcal{M}_i|\text{Data}) = \frac{P(\text{Data}|\mathcal{M}_i)P(\mathcal{M}_i)}{P(\text{Data})}$$

- For two models $\mathcal{M}_1$ and $\mathcal{M}_2$, we can write:

$$\frac{P(\mathcal{M}_1|\text{Data})}{P(\mathcal{M}_2|\text{Data})} = \frac{P(\text{Data}|\mathcal{M}_1)}{P(\text{Data}|\mathcal{M}_2)} \times \frac{P(\mathcal{M}_1)}{P(\mathcal{M}_2)}$$

- We see that the ratio of the posteriors is equal to the ratio of the priors times the following factor:

$$\frac{P(\text{Data}|\mathcal{M}_1)}{P(\text{Data}|\mathcal{M}_2)} \qquad \Leftarrow \text{the Bayes factor}$$

- To calculate the Bayes factor, we need to compute $P(\text{Data}|\mathcal{M}_i)$ for each model. How do we compute it?

- We can define the model likelihood as the following function:

$$\mathcal{L}(\boldsymbol{\beta}|\text{Data}, \mathcal{M}_i) = p(\text{Data}|\boldsymbol{\beta}, \mathcal{M}_i)$$

- $P(\text{Data}|\mathcal{M}_i)$ is the integral:

$$P(\text{Data}|\mathcal{M}_i) = \int_{\boldsymbol{\beta}} p(\text{Data}|\boldsymbol{\beta}, \mathcal{M}_i) p(\boldsymbol{\beta}|\mathcal{M}_i) d\boldsymbol{\beta}$$

- We have integrated the model likelihood over the distribution of the model parameters.

- The result is the marginal likelihood.

• If we don't know anything about the data, we can set the priors for different models equal:

$$P(\mathcal{M}_1) = P(\mathcal{M}_2)$$

• Then, the ratio of the posteriors is given by the ratio of the marginal likelihoods:

$$\frac{P(\mathcal{M}_1|\text{Data})}{P(\mathcal{M}_2|\text{Data})} = \frac{P(\text{Data}|\mathcal{M}_1)}{P(\text{Data}|\mathcal{M}_2)}$$

• Thus, the relative magnitudes of the marginal likelihoods can be used to tell which model is better.

• The trouble is, it is very difficult to calculate marginal likelihoods

Model selection .. let's start over

Let's say we have built two models with two different kernels.

How to tell which one is better?

When we train a model, we minimize a loss function or maximize log likelihood.

Why not to just chose the model that gives a lower loss or bigger likelihood? ... We can't, because of overfitting

Generalization error

Definition: Test error = generalization error

$$\text{Err}_{\mathcal{T}} = E\left[L(\boldsymbol{y}, \hat{f}(\boldsymbol{X}))|\mathcal{T}\right] \quad \text{where } \mathcal{T} = \text{fixed training set}$$

Expected prediction error:

$$\text{Err} = E\left[L(\boldsymbol{y}, \hat{f}(\boldsymbol{X}))\right] = E\left[\text{Err}_{\mathcal{T}}\right]$$

Bias vs Variance Trade-off

Consider an ensemble of data points that derives from the function $f(\boldsymbol{x})$ and some noise (ε) inherent to the data:

$$\text{Data} = f(\boldsymbol{x}) + \varepsilon$$

The function $f(\boldsymbol{x})$ is generally unknown and our goal is to infer it.

We build a regression fit of the data $\hat{f}(\boldsymbol{x})$.

Now we want to test the performance of the fit at some point $\boldsymbol{x}_0$.

The **expected prediction error** at $\boldsymbol{x}_0$:

$$\text{EPE}(\boldsymbol{x}_0) = E\left[\left(f(\boldsymbol{x}_0) + \varepsilon - \hat{f}(\boldsymbol{x}_0)\right)^2\right]$$

Let's expand the square in the last equation:

$$\mathrm{EPE}(\boldsymbol{x}_0) = E\left[\left(f(\boldsymbol{x}_0) + \varepsilon - \hat{f}(\boldsymbol{x}_0)\right)^2\right] =$$

$$E\left[\left(f(\boldsymbol{x}_0) - \hat{f}(\boldsymbol{x}_0)\right)^2\right] + E\left[2\varepsilon\left(f(\boldsymbol{x}_0) - \hat{f}(\boldsymbol{x}_0)\right)\right] + E\left[\varepsilon^2\right]$$

Note that for two independent random variables A and B:

$$E(A+B) = E(A) + E(B)$$
$$E(AB) = E(A)E(B)$$
$$E\left[(A-B)^2\right] = E\left[(A)^2\right] + E\left[(B)^2\right] - 2E(A)E(B)$$
$$\text{Variance of } A \text{ with mean } \mu = E\left[(A-\mu)^2\right]$$

In our case, what are the random variables?

$\Rightarrow$ Noise ε. Note that $E(\varepsilon) = 0$.

Because $E(\varepsilon) = 0$, the second term: $E\left[2\varepsilon\left(f(\boldsymbol{x}_0) - \hat{f}(\boldsymbol{x}_0)\right)\right] = 0$

The third term: $E(\varepsilon^2) = \sigma_\varepsilon^2$ is the variance of noise.

$\Rightarrow \hat{f}(\boldsymbol{x}_0)$ is another independent random variable. To see this, imagine selecting training data at random and producing $\hat{f}(\boldsymbol{x})$. Each time, $\hat{f}(\boldsymbol{x}_0)$ will be different.

$\Rightarrow$ Note also that $E\left[f(\boldsymbol{x}_0)\right] = f(\boldsymbol{x}_0)$ because the function $f(\boldsymbol{x})$ is not random; it is a well-defined (although unknown) function.

Let's now deal with the first term:

$$E\left[\left(f(\boldsymbol{x}_0) - \hat{f}(\boldsymbol{x}_0)\right)^2\right] = E\left[\hat{f}^2(x_0)\right] - 2f(\boldsymbol{x}_0)E\left[\hat{f}(x_0)\right] + f^2(\boldsymbol{x}_0)$$

We will now add and subtract $E\left[\hat{f}(x_0)\right] E\left[\hat{f}(x_0)\right]$ to get:

$$E\left[\left(f(\boldsymbol{x}_0) - \hat{f}(\boldsymbol{x}_0)\right)^2\right] =$$

$$\left(E\left[\hat{f}(x_0)\right] - f(\boldsymbol{x}_0)\right)^2 + \left(E\left[\hat{f}^2(x_0)\right] - E\left[\hat{f}(x_0)\right] E\left[\hat{f}(x_0)\right]\right)$$

$$\text{Bias}^2\left[\hat{f}(\boldsymbol{x}_0)\right] + \text{Var}\left[\hat{f}(\boldsymbol{x}_0)\right]$$

The first term is the square of the bias of the model $\hat{f}$.

The second term is the variance of the model $\hat{f}$.

So, we see that for a model $\hat{f}(\boldsymbol{x})$, the **expected prediction error** at $\boldsymbol{x}_0$ is given by

$$\text{EPE}(\boldsymbol{x}_0) = \text{Variance of data noise}$$
$$+ \text{Square of the bias of the model}$$
$$+ \text{Variance of the model}$$

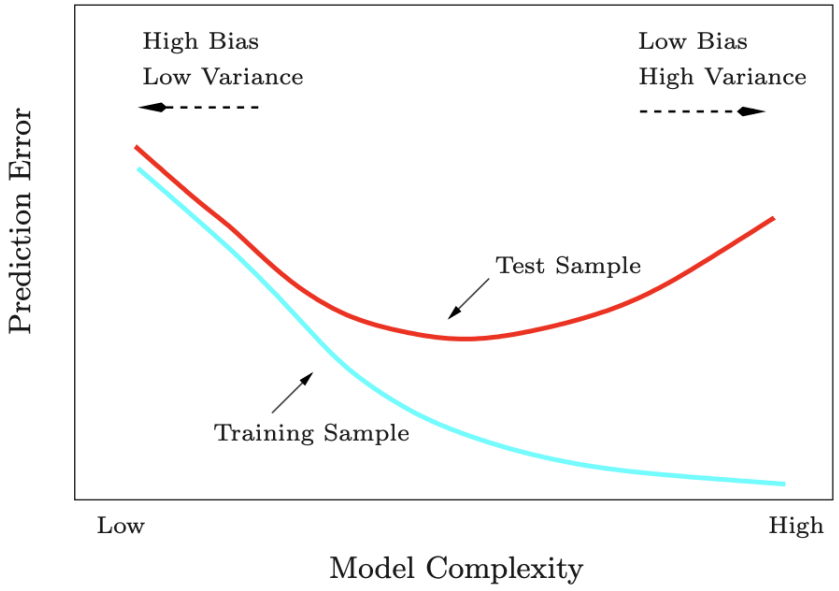


Figure source: T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer (2001)

So, we need a good way to discriminate between models that also accounts for their complexity.

How do we account for model complexity?

The Bayesian information criterion:

$$\text{BIC} = -2\text{loglik} + d\log(n)$$

where n is the number of training points

d is the number of model parameters

loglik is the maximum value of the log likelihood

It turns out that BIC is very closely related to the logarithm of marginal likelihood in the large n limit

In particular,

$$\log(P(\text{Data}|\mathcal{M}_i)) \approx \log(P(\text{Data}|\text{model } \hat{\text{p}}\text{arameters}, \mathcal{M}_i)) - \frac{d}{2}\log(n)$$

BIC can be used to estimate the posterior probability of each model $\mathcal{M}_m$ as

$$\frac{e^{-\frac{1}{2}\text{BIC}_m}}{\mathcal{N}}, \quad \text{where } \mathcal{N} = \sum_{l=1}^{M} e^{-\frac{1}{2}\text{BIC}_l}$$

BIC is asymptotically consistent as a model selection metric

Given the family of models, including the true model, the probability that BIC will select the true model approaches one as $n \to \infty$

Given,

$$\log(P(\text{Data}|\mathcal{M}_i)) \approx \log(P(\text{Data}|\text{model } \hat{\text{p}}\text{arameters}, \mathcal{M}_i)) - \frac{d}{2}\log(n)$$

We can define

$$\text{BIC} = \text{LML}_{\text{GPR}} - \frac{d}{2}\log n$$

where d is the number of kernel parameters in GP regression to discriminate between kernels

As we saw, this is an approximation to the logarithm of the marginal likelihood, which is valid in the large n limit.

This can be used to build better kernels

The beauty: we will see that this works even for low values of n

Algorithm for optimal kernel construction

Start with conventional covariance functions (kernels), such as these ones:

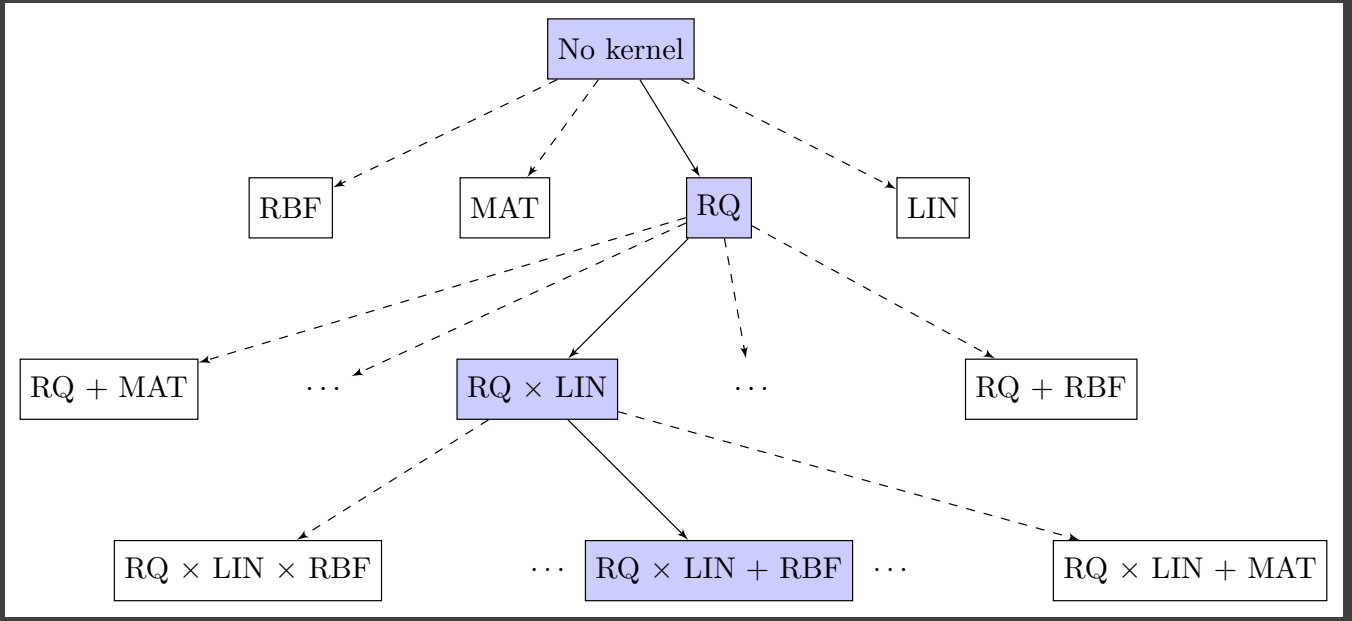
$$k_{\mathrm{LIN}}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$$

$$k_{\mathrm{RBF}}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2} r^2(\mathbf{x}_i, \mathbf{x}_j)\right)$$

$$k_{\mathrm{MAT}}(\mathbf{x}_i, \mathbf{x}_j) = \left(1 + \sqrt{5} r^2(\mathbf{x}_i, \mathbf{x}_j) + \frac{5}{3} r^2(\mathbf{x}_i, \mathbf{x}_j)\right) \times \exp\left(-\sqrt{5} r^2(\mathbf{x}_i, \mathbf{x}_j)\right)$$

$$k_{\mathrm{RQ}}(\mathbf{x}_i, \mathbf{x}_j) = \left(1 + \frac{|\mathbf{x}_i - \mathbf{x}_j|^2}{2\alpha\ell^2}\right)^{-\alpha}$$

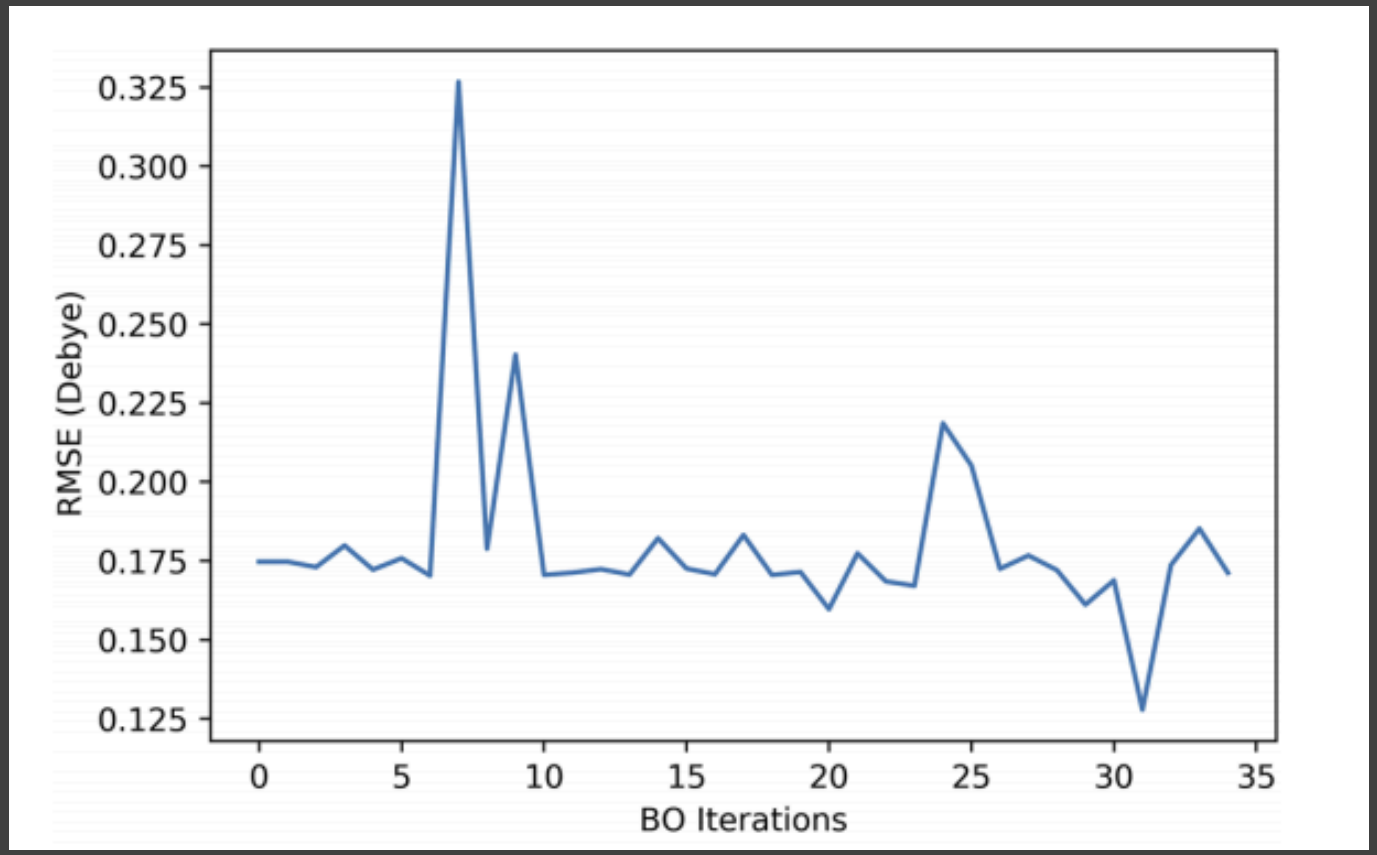
Combine them using a greedy search algorithm



Duvenaud, D. K.; Nickisch, H.; Rasmussen, C. E. Additive Gaussian Processes, Adv. Neur. Inf. Proc. Sys. 2011, 24, 226

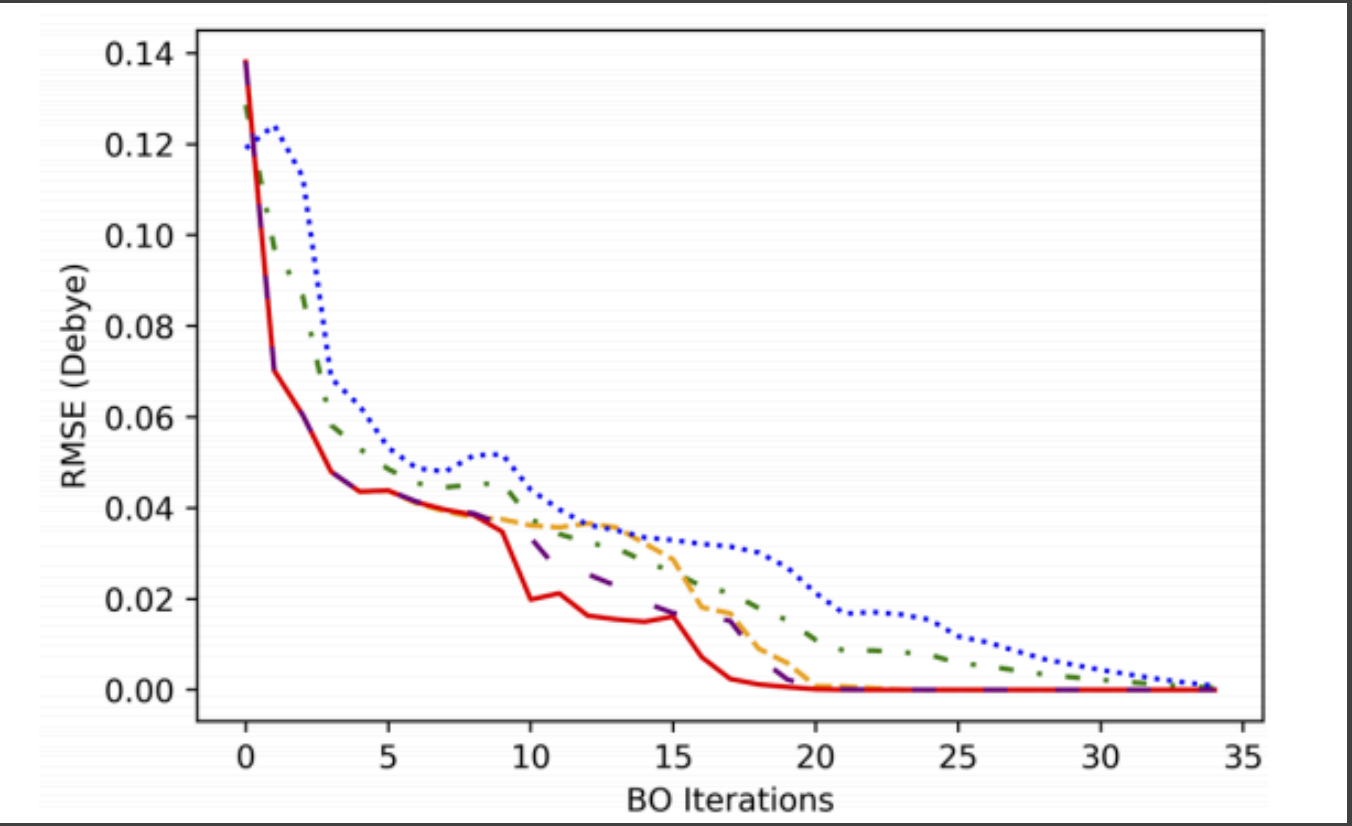
Duvenaud, D. K.; Lloyd, J.; Grosse, R.; Tenenbaum, J. B.; Ghahramani, Z.; Structure Discovery in Nonparametric Regression through Compositional Kernel Search, Proceedings of the 30th International Conference on Machine Learning Research 2013, 28, 1166

The choice of kernel is important!



This is what happens with a poorly chosen kernel

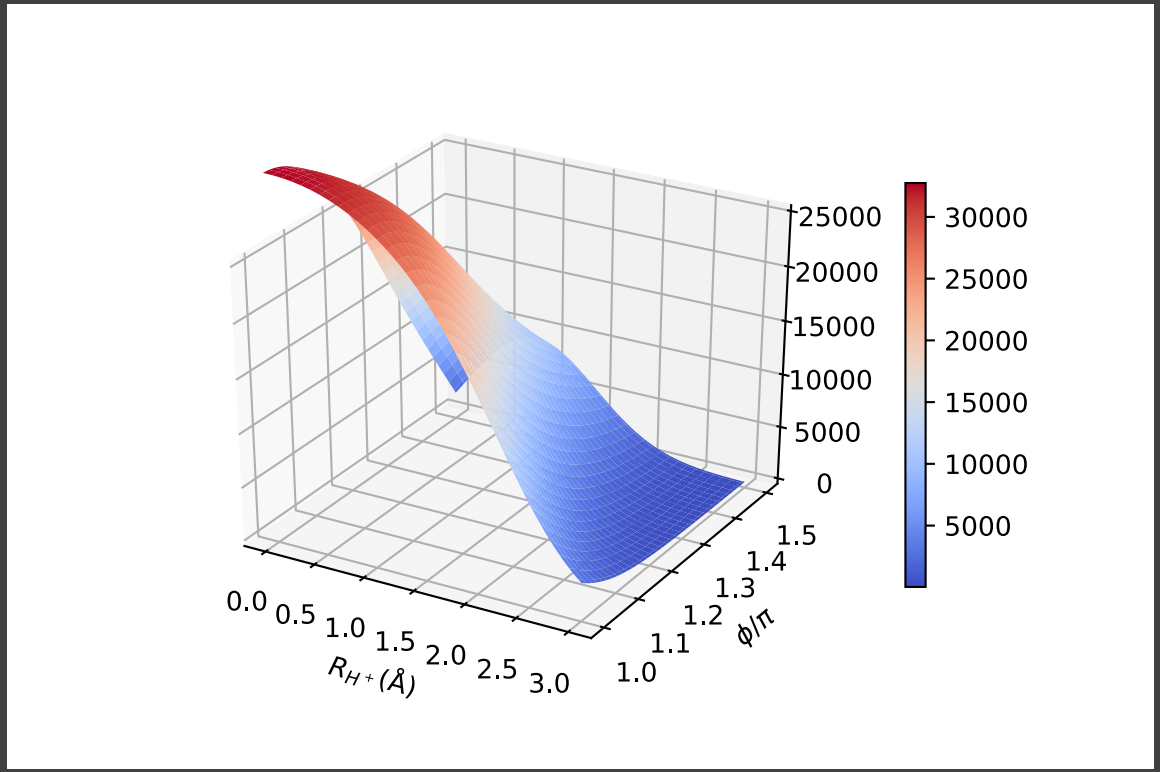
The choice of kernel is important!



Inferring molecular properties from time-dependent observables

Extrapolation of potential energy surfaces

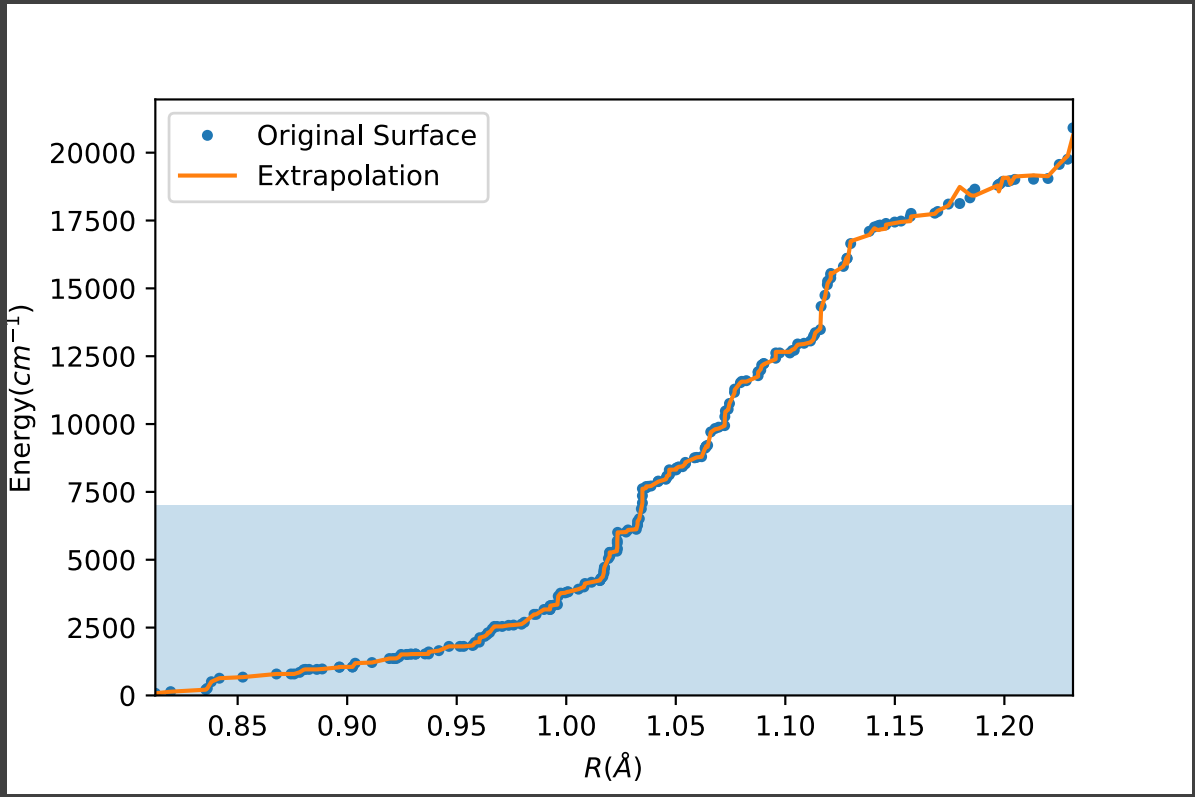
Six-dimensional surface for H_3O^+



Jun Dai and R. V. Krems, J. Chem. Theory Comp. 16, 1386 (2020)

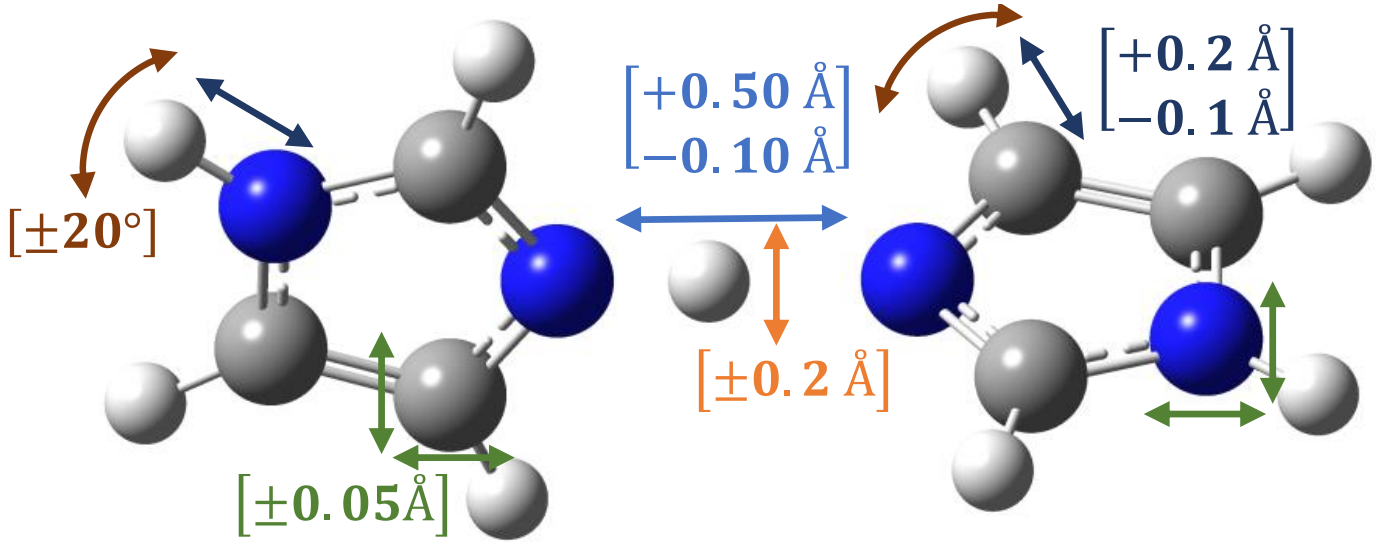
Extrapolation of potential energy surfaces

Six-dimensional surface for H_3O^+ (1000 ab initio geometries)



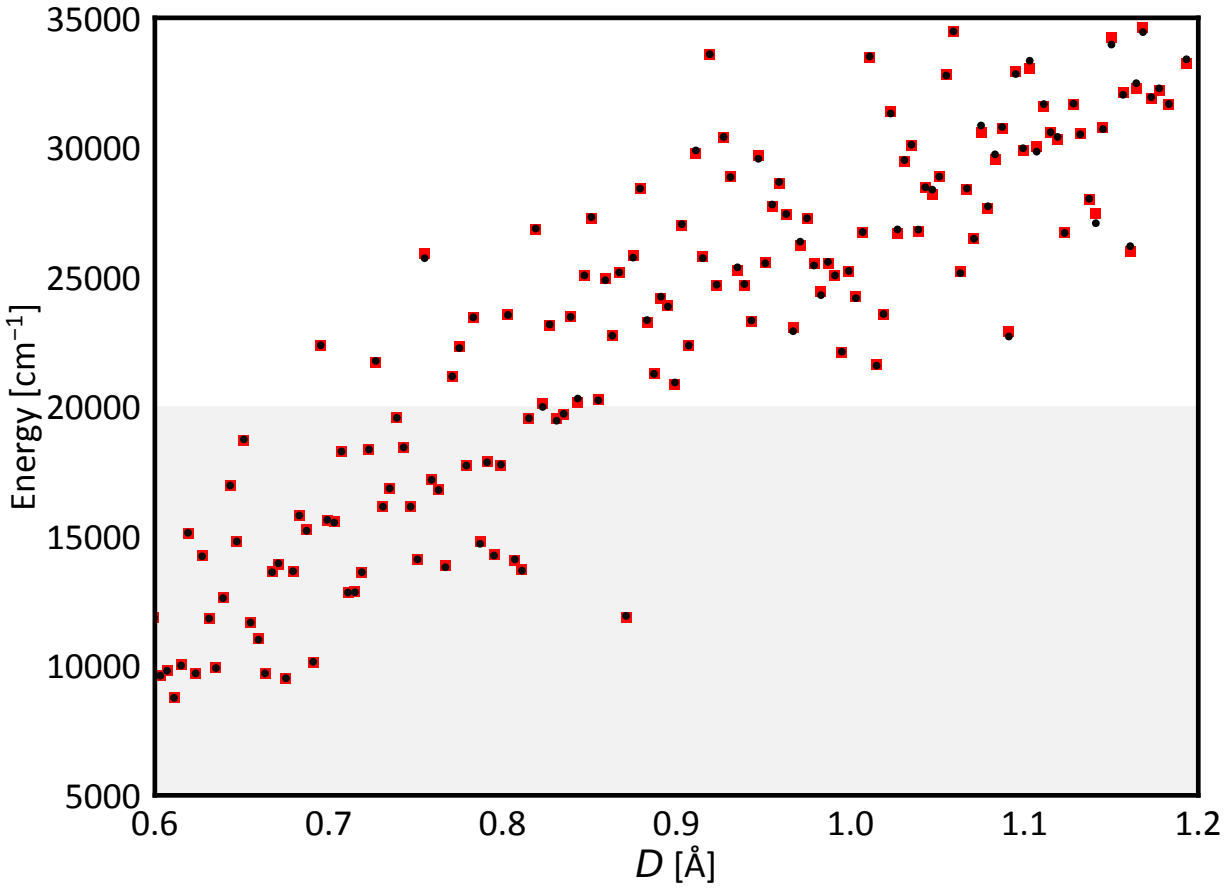
Jun Dai and R. V. Krems, J. Chem. Theory Comp. 16, 1386 (2020)

51 dimensions (5000 molecular geometries)



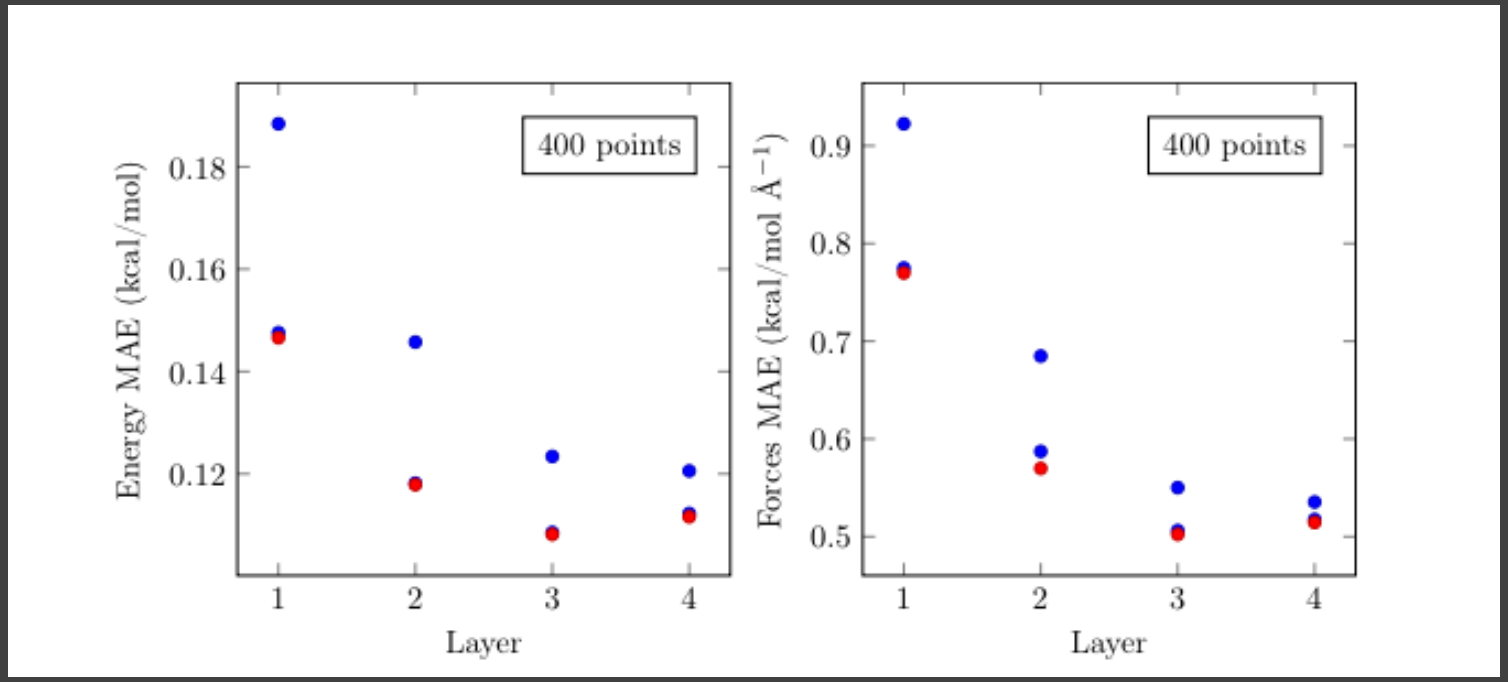
Hiroki Sugisawa, I. Sato and RVK, *J. Chem. Phys.* 153, 114101 (2020)

51 dimensions (5000 molecular geometries)

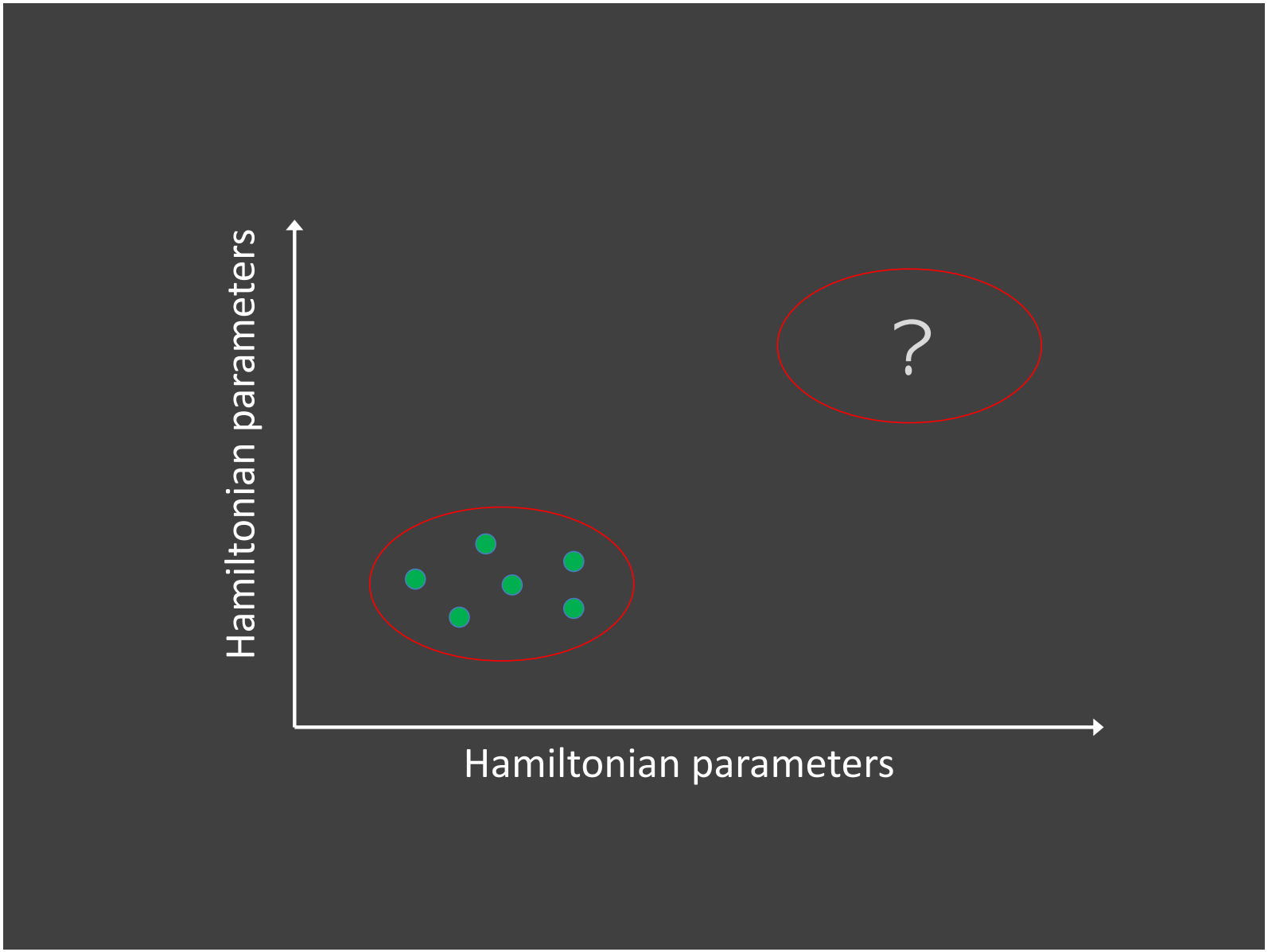


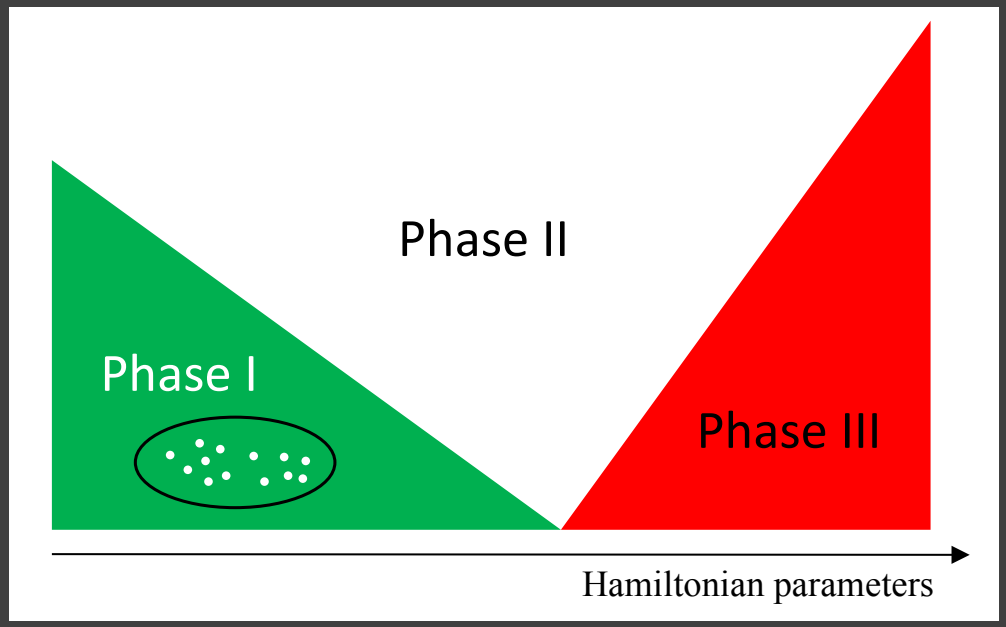
Hiroki Sugisawa, I. Sato and RVK, *J. Chem. Phys.* 153, 114101 (2020)

57 Dimensional surface for aspirin

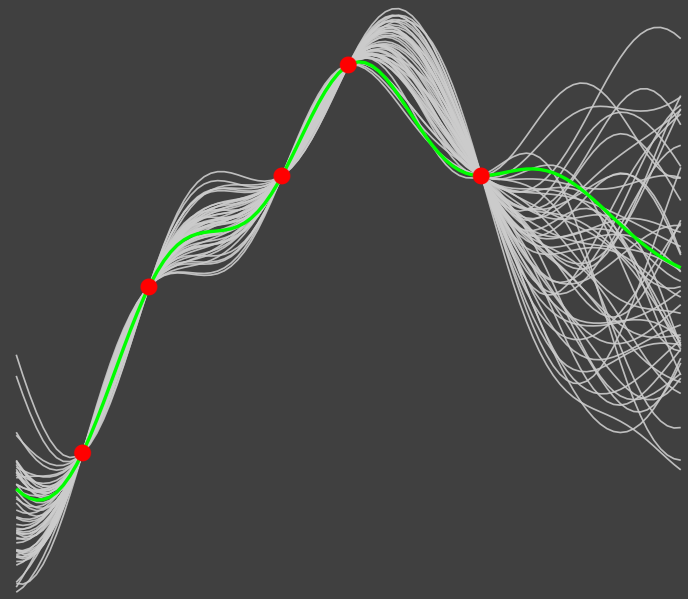


Kasra Asnaashari and R. V. Krems, arXiv: 2707.04779 (2021)


Hamiltonian parameters
?
Hamiltonian parameters



Can Phase I be used to predict the properties of Phases II and III?

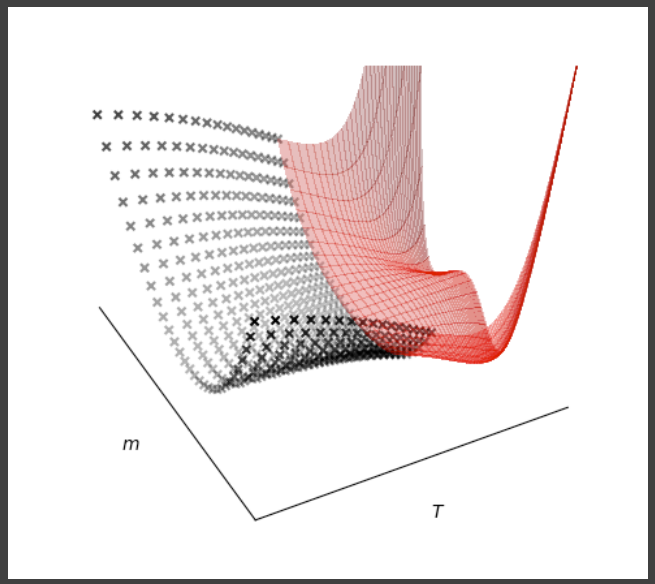


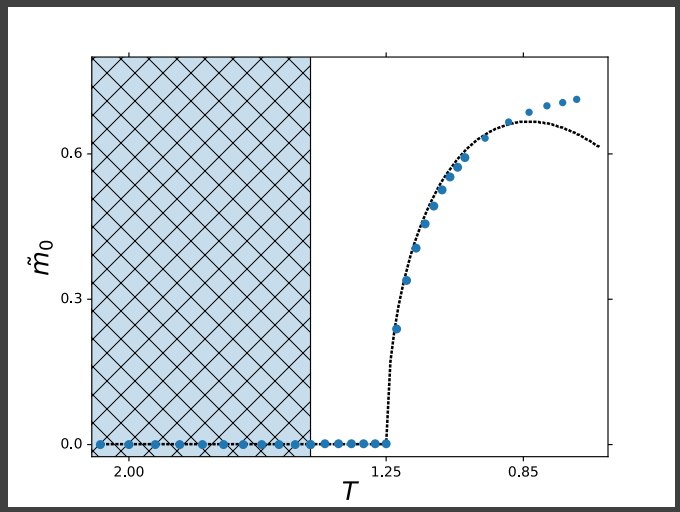
To do this, one needs to solve the extrapolation problem. How?

The key is in the complexity of the kernel function.

The complexity needs to be built up, but in a `physical' way.

Heisenberg spin model





Rodrigo Vargas, John Sous, Mona Berciu and R. V. Krems, **Phys. Rev. Lett. 121, 255702 (2018)**
Jun Dai and R. V. Krems**, J. Chem. Theory Comp. 16, 1386 (2020)**

Generalized polaron model

$$\mathcal{H} = \sum_k \epsilon_k c_k^\dagger c_k + \sum_q \omega_q b_q^\dagger b_q + V_{\mathrm{e-ph}}$$

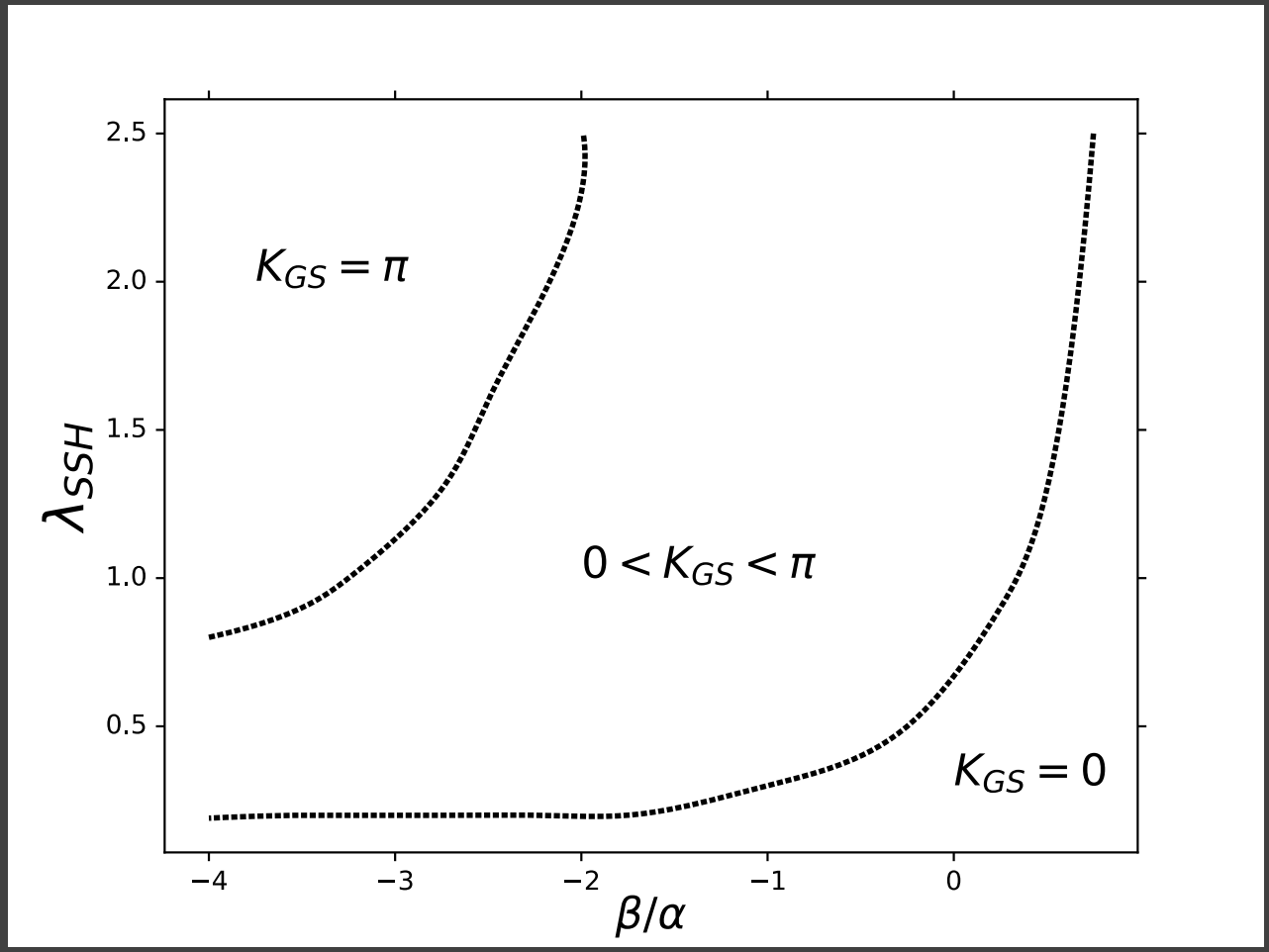
$$V_{\mathrm{e-ph}} = \alpha H_1 + \beta H_2$$

$$H_1 = \sum_{k,q} \frac{2i}{\sqrt{N}} \left[\sin(k+q) - \sin(k)\right] c_{k+q}^\dagger c_k \left(b_{-q}^\dagger + b_q\right)$$

$$H_2 = \sum_{k,q} \frac{2i}{\sqrt{N}} \sin(q) c_{k+q}^\dagger c_k \left(b_{-q}^\dagger + b_q\right)$$

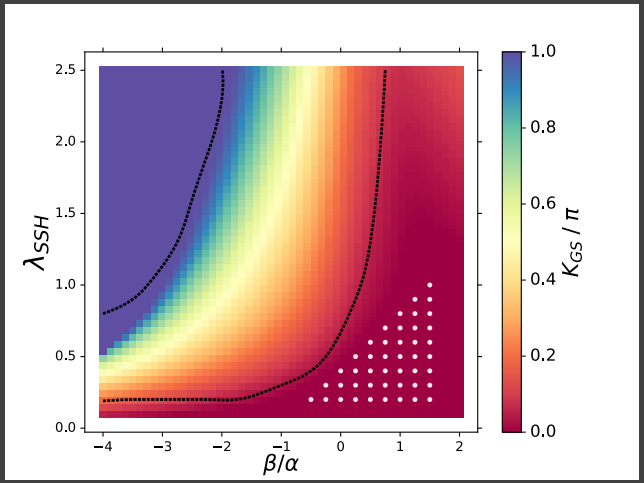
Felipe Herrera, Kirk Madison, RK, Mona Berciu, *Phys. Rev. Lett.* 110, 223002 (2013)

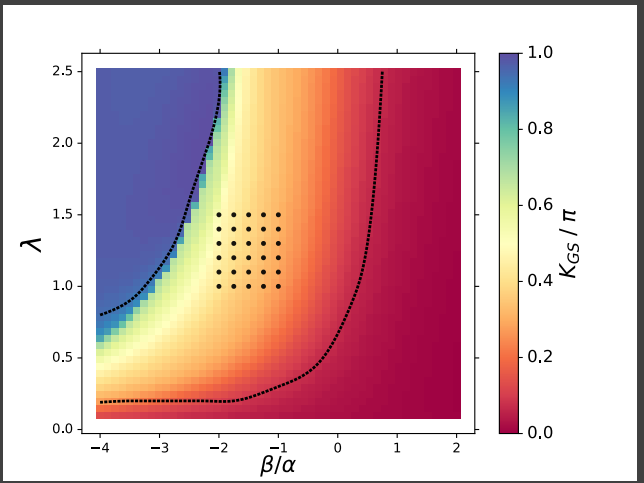
Generalized polaron model



Felipe Herrera, Kirk Madison, RK, Mona Berciu, *Phys. Rev. Lett. 110, 223002 (2013)*

Polaron model

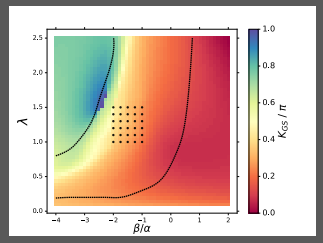
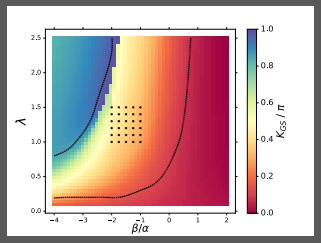
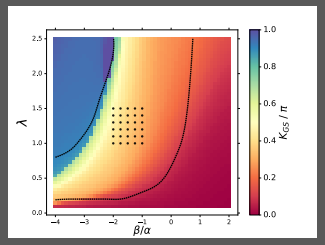
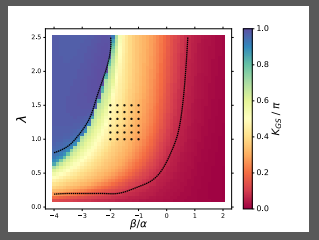


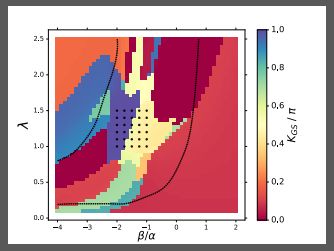
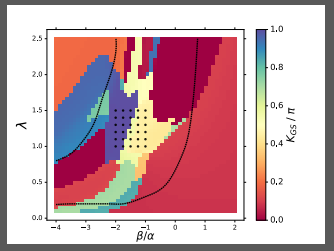
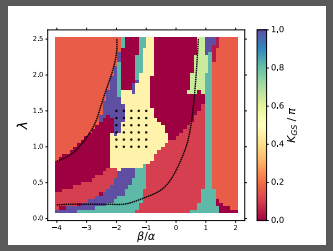
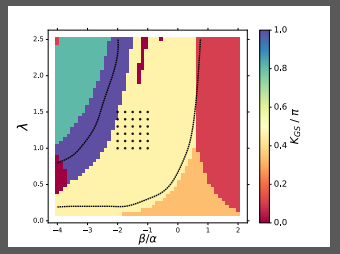


Rodrigo Vargas, John Sous, Mona Berciu and RK, *Phys. Rev. Lett. 121, 255702 (2018)*

Tree level

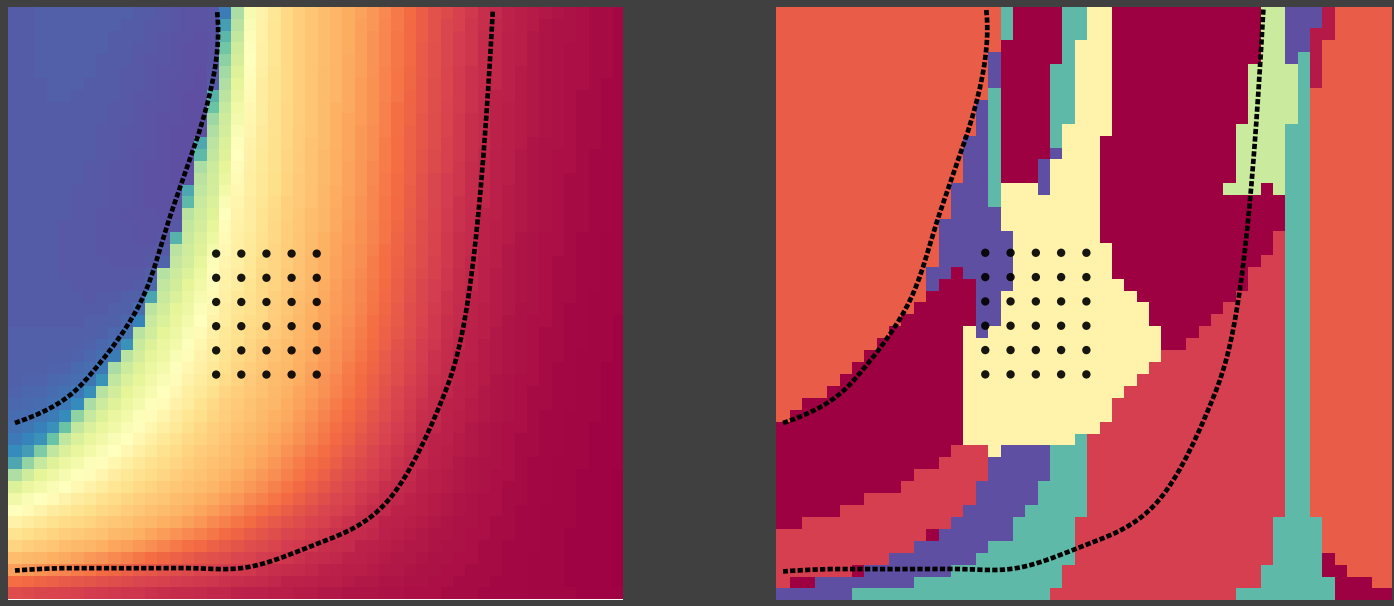
I II III IV

Rodrigo Vargas and RK, arXiv: 1901.00854

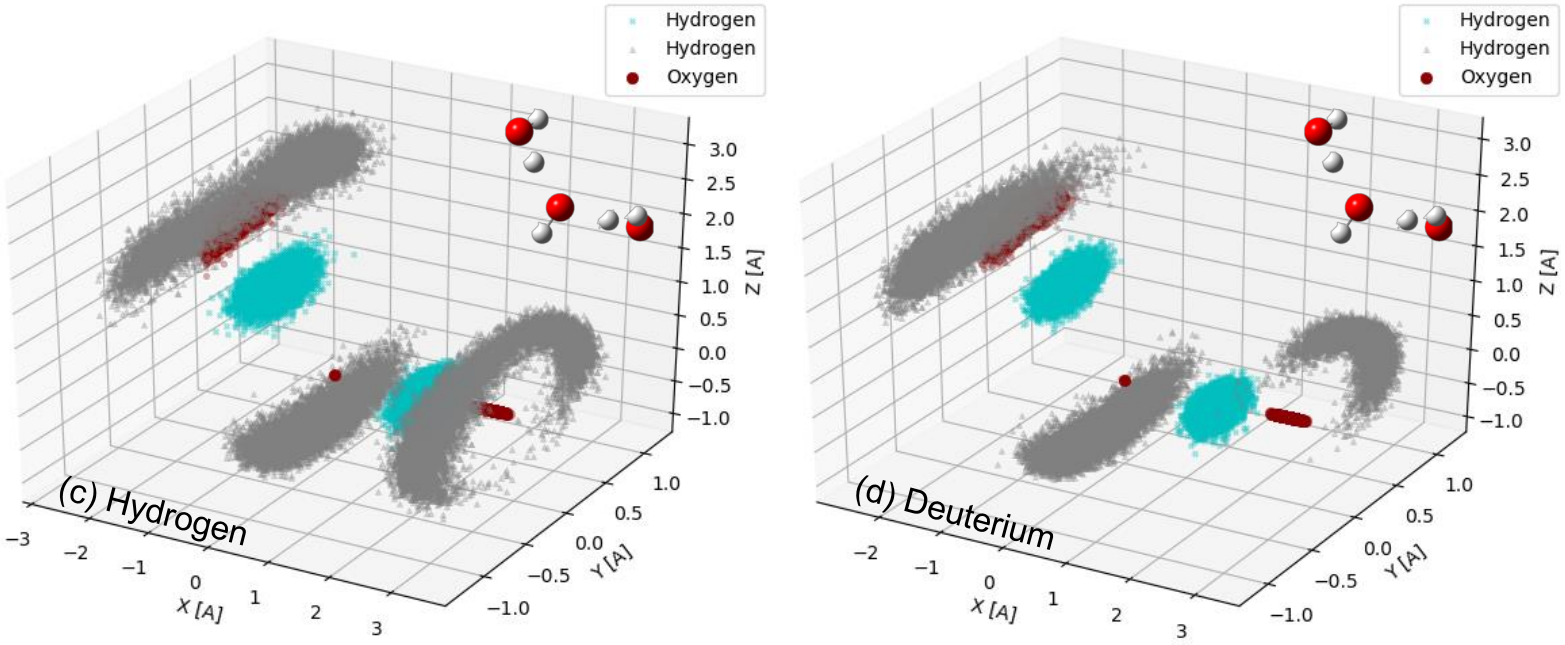
Quantum extrapolation problems



Choosing better kernels not only makes extrapolation possible, but also allows models to extrapolate farther

Choosing better kernels is like replacing spectacles with binoculars when it comes to quantum phase diagrams

This allows us to compute things we couldn't imagine just a few years ago… Work of Hiroki Sugisawa



Wave function of $H_5O_3^-$ at the MP2 quantum chemistry level

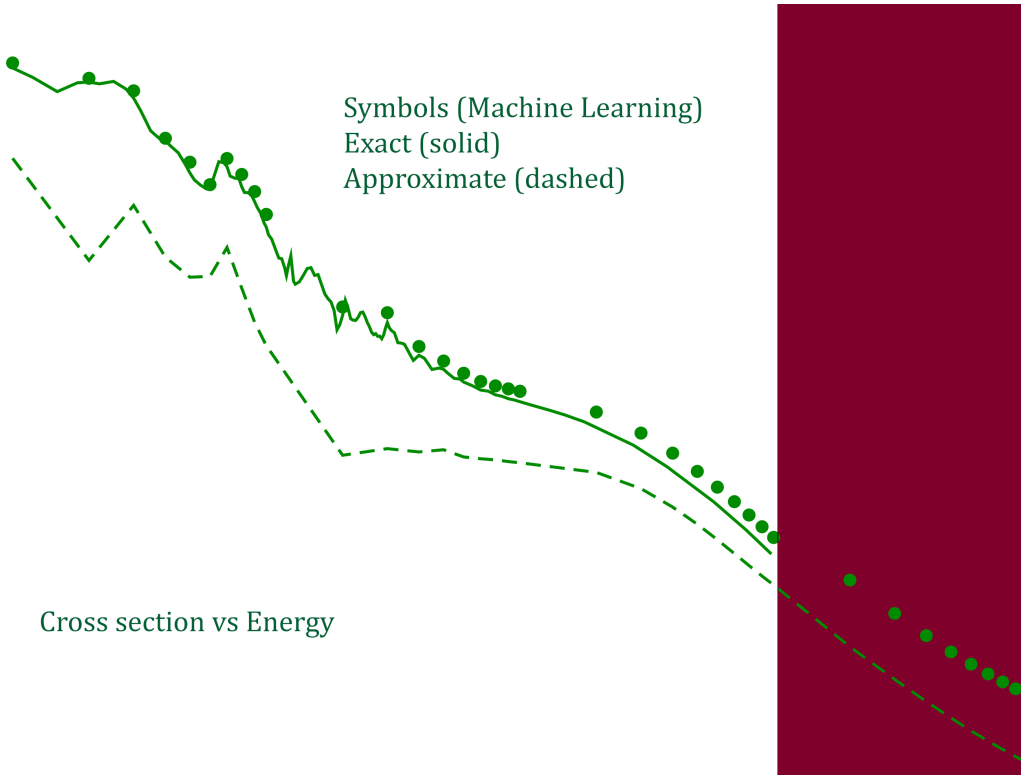
Direct approach: 220 years on a single core of Intel i7-9700K

Our approach: 8 days

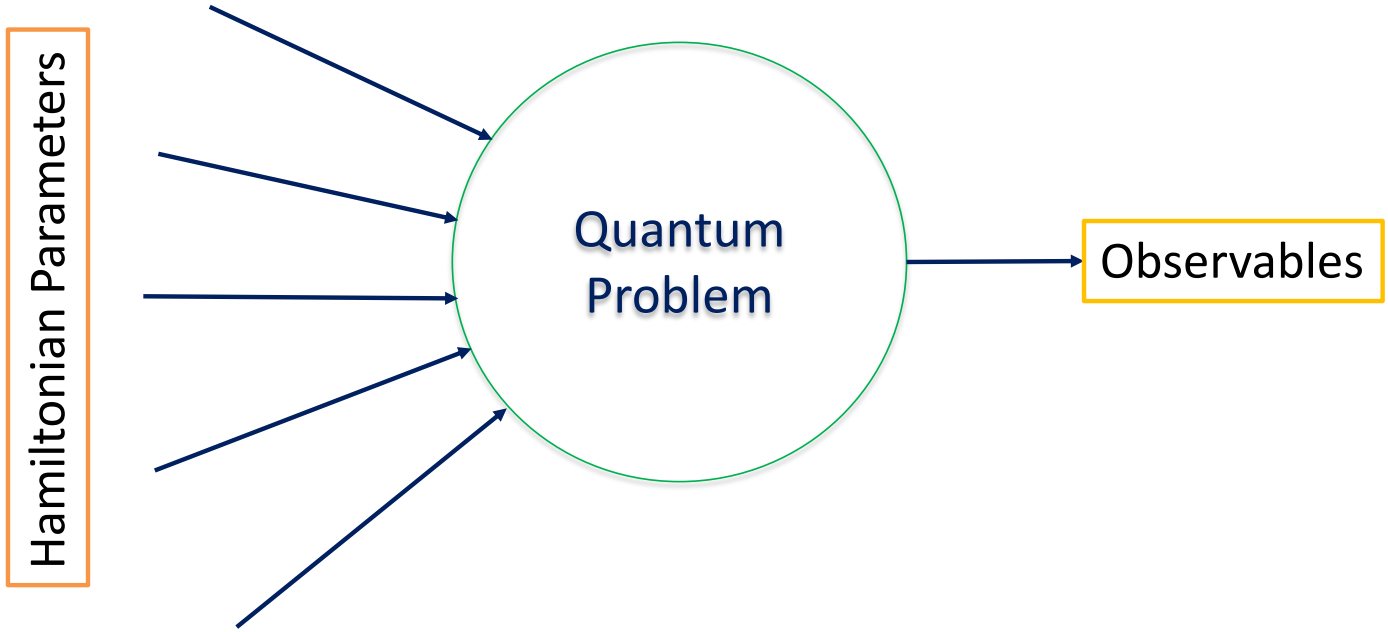
Same models can be used for transfer learning

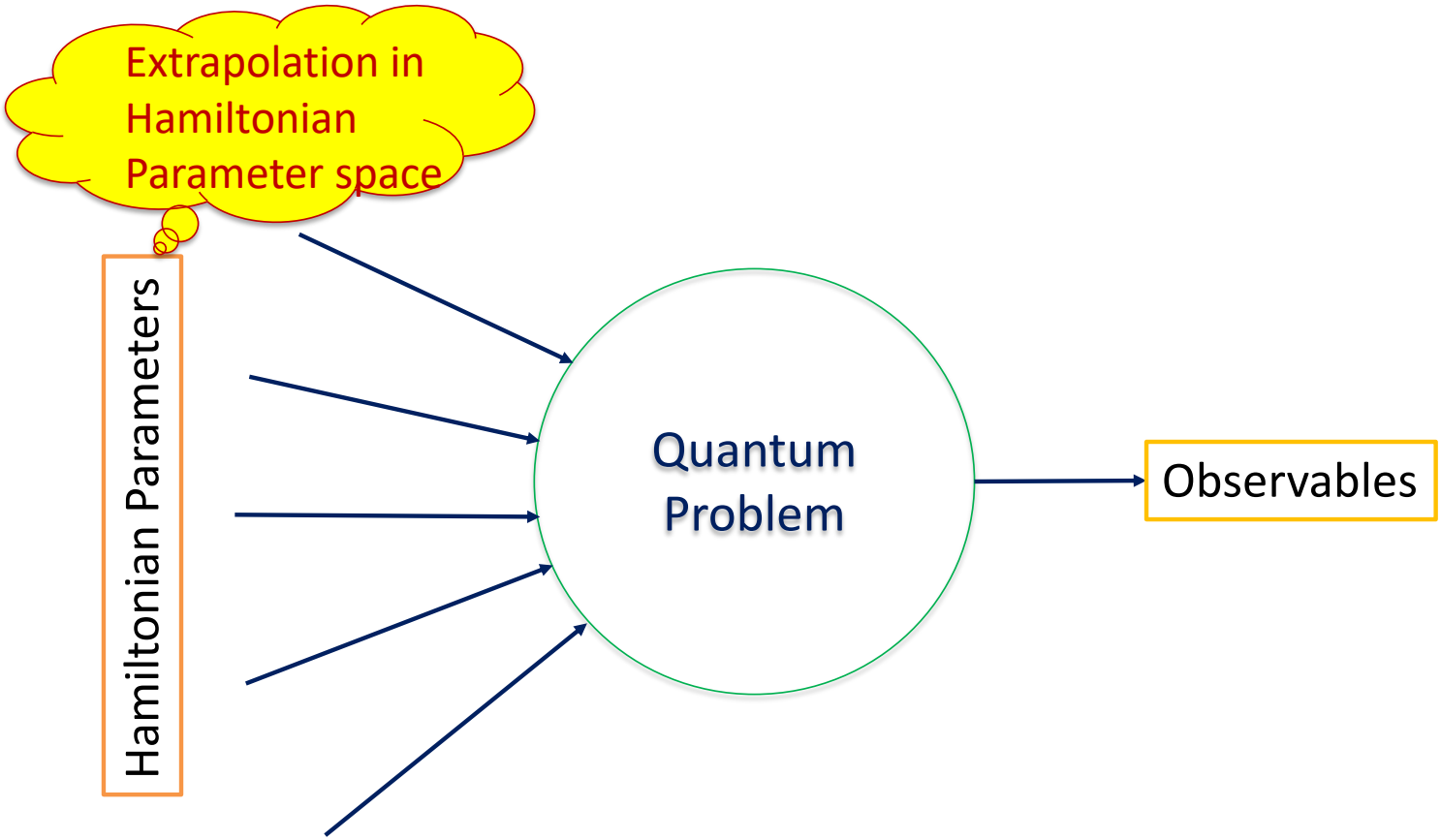
- Quantum dynamics calculations for complex systems are difficult
- Such calculations must rely on approximations
- Can the results of approximate quantum calculations be corrected by machine learning?

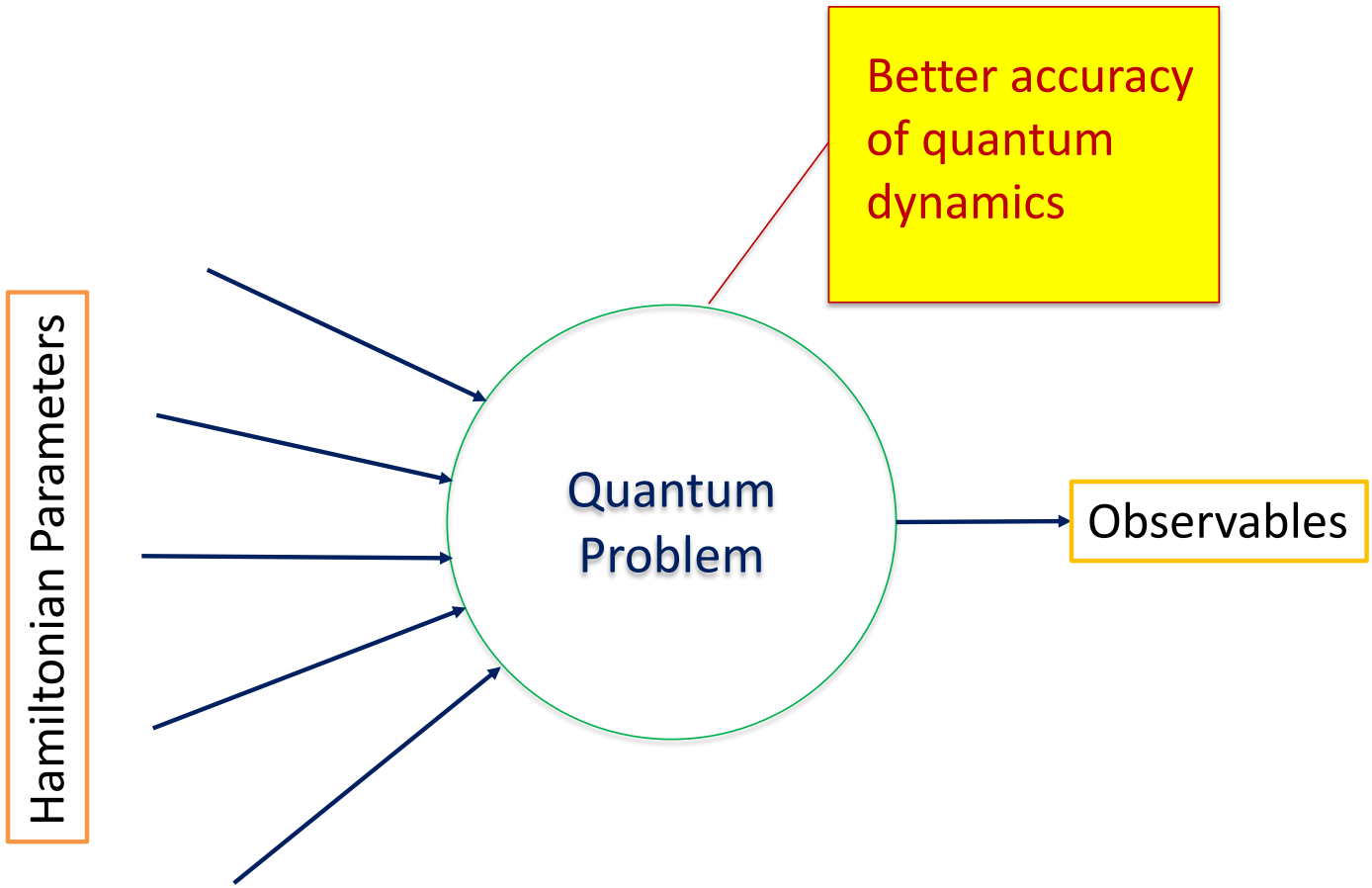
A. Jasinski, J. Montaner, R. C. Forrey, B. H. Yang, P. C. Stancil, N. Balakrishnan, R. Vargas-Hernandez, J. Dai and R. V. Krems, **PRR 2, 032051 (2020)**

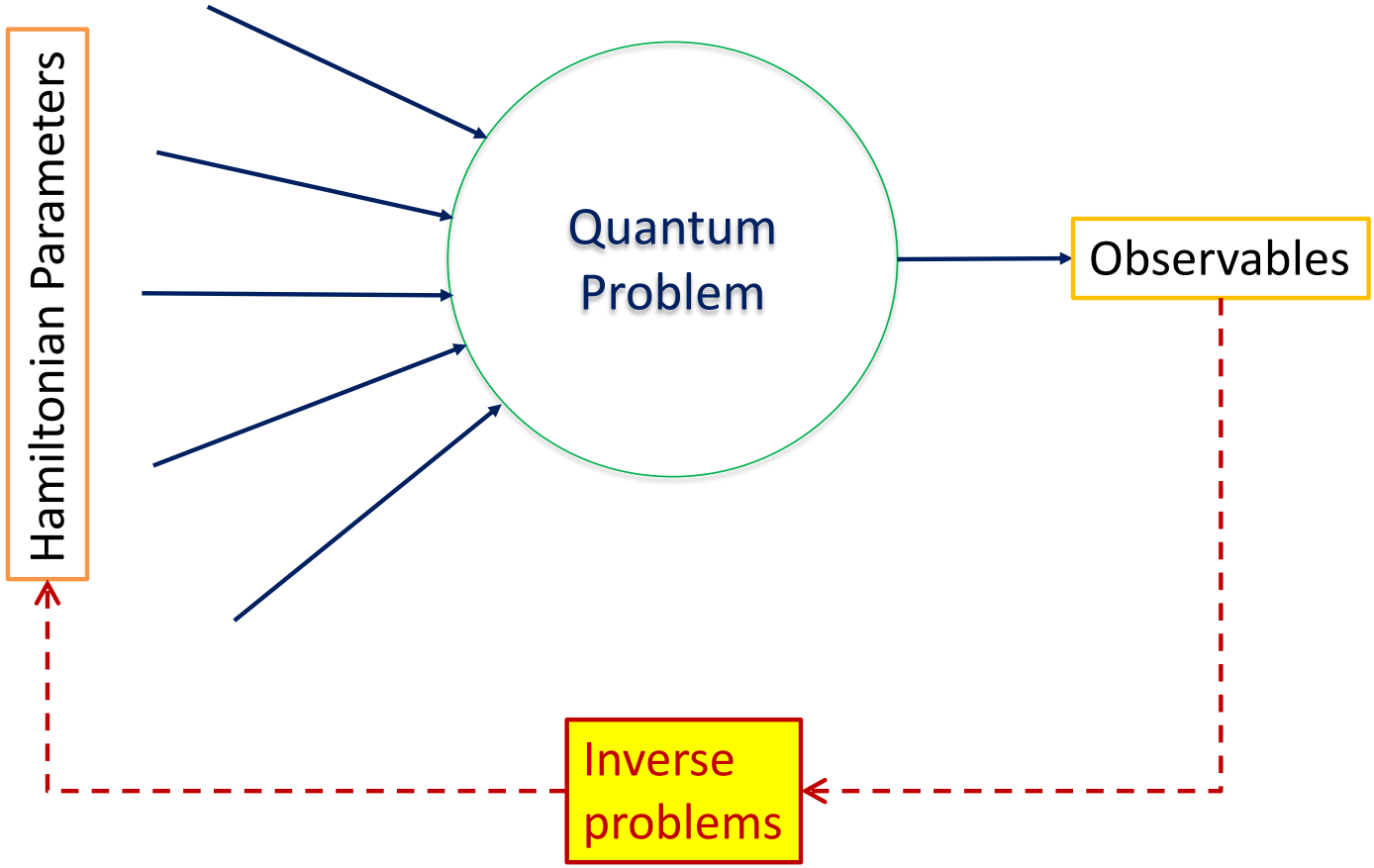


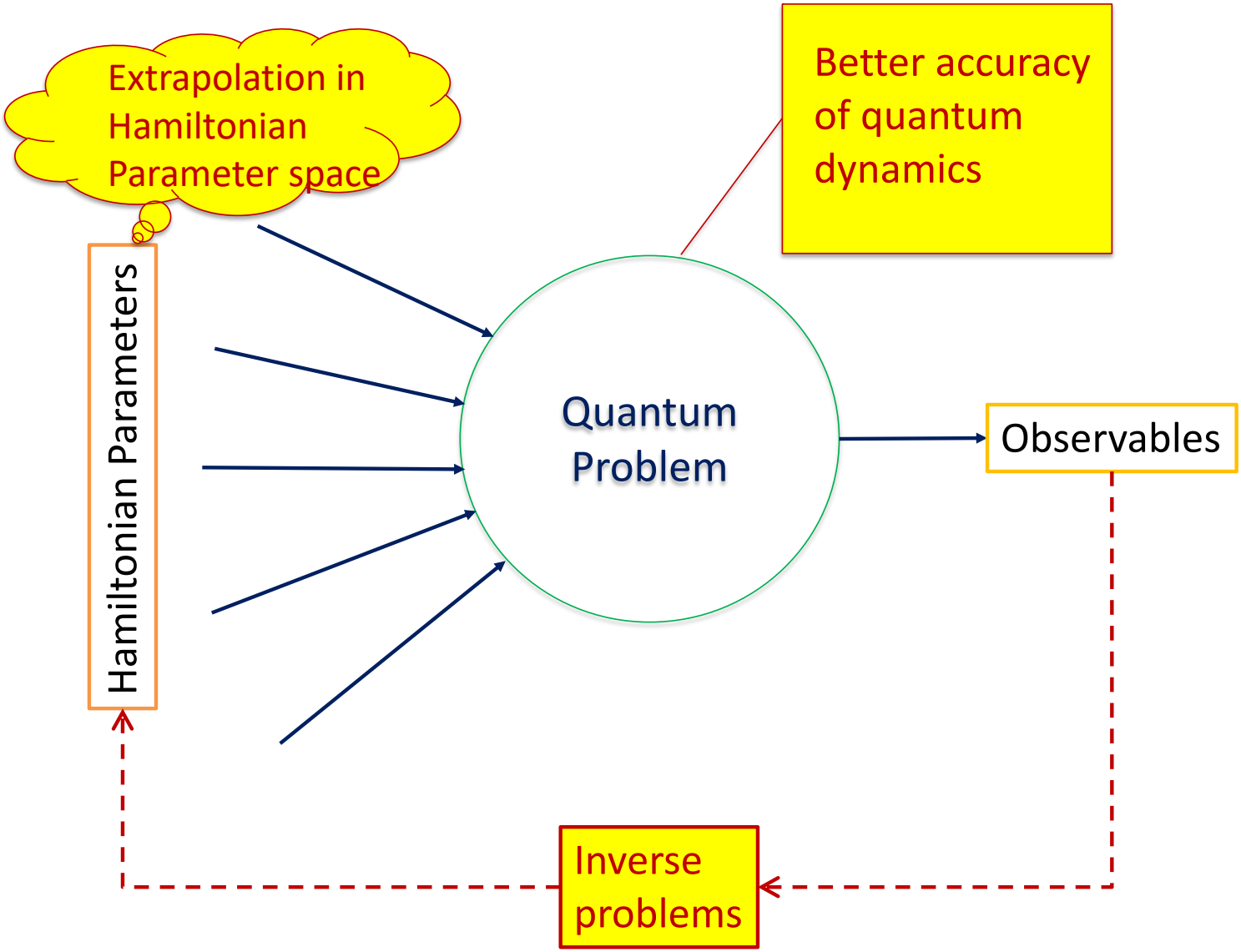
A. Jasinski, J. Montaner, R. C. Forrey, B. H. Yang, P. C. Stancil, N. Balakrishnan, R. Vargas-Hernandez, J. Dai and R. V. Krems, **PRR 2, 032051 (2020)**


Hamiltonian Parameters
Quantum Problem
Observables


Extrapolation in Hamiltonian Parameter space
Hamiltonian Parameters
Quantum Problem
Observables


Better accuracy of quantum dynamics
Hamiltonian Parameters
Quantum Problem
Observables

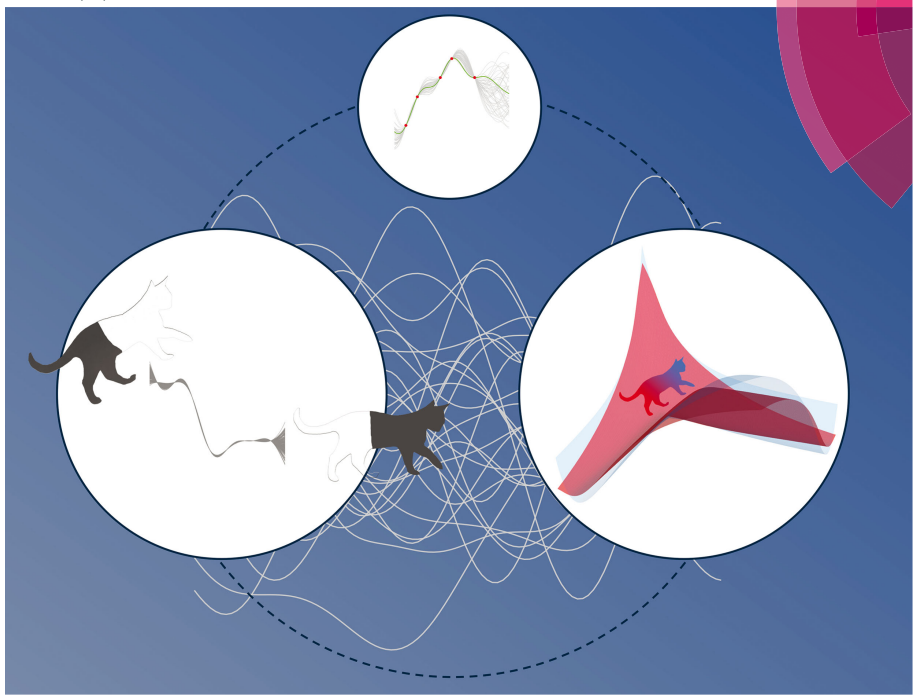

Hamiltonian Parameters
Quantum Problem
Observables
Inverse problems


Extrapolation in Hamiltonian Parameter space
Better accuracy of quantum dynamics
Hamiltonian Parameters
Quantum Problem
Observables
Inverse problems

Volume 21 | Number 25 | 7 July 2019 | Pages 13377–13858

PCCP

Physical Chemistry Chemical Physics
rsc.li/pccp



ISSN 1463-9076

ROYAL SOCIETY OF CHEMISTRY | Celebrating IYPT 2019

PERSPECTIVE
R. V. Krems
Bayesian machine learning for quantum molecular dynamics

R. V. Krems,
PCCP 21, 13392 (2019)

If you know how to build the right kernel, predictions are easy!

... Can Quantum Computers help?

Better kernels lead to more powerful ML models!

Kernel functions are inner products in a Hilbert space.

Consider a quantum computer with n qubits, initially in state $|0^n\rangle$.

Introduce a sequence of gates that produces a quantum state $\mathcal{U}(\boldsymbol{x})|0^n\rangle$

and another state $\mathcal{U}(\boldsymbol{x'})|0^n\rangle$

The measurable square of the inner product:

$$|\langle 0^n|\mathcal{U}^\dagger(\boldsymbol{x'})\mathcal{U}(\boldsymbol{x})|0^n\rangle|^2$$

has all the properties of a kernel of an RKHS

Thus, projecting

$$\mathcal{U}^\dagger(\boldsymbol{x}')\mathcal{U}(\boldsymbol{x})|0^n\rangle \quad \text{onto} \quad |0^n\rangle$$

can be another way of building kernels for kernel ML:

$$K(\boldsymbol{x}, \boldsymbol{x}') = |\langle 0^n|\mathcal{U}^\dagger(\boldsymbol{x}')\mathcal{U}(\boldsymbol{x})|0^n\rangle|^2$$

Question: Can such measurements be used to build kernels for ML?

Question: What is the best way to build quantum kernels for ML?

Question: Can quantum kernels outperform classical kernels for ML?

$\Rightarrow$ **Quantum Machine Learning**

What is the RKHS with the kernel function produced on a quantum computer?

Consider the embedding of input features into quantum states:

$$x \rightarrow |\psi(x)\rangle$$

Define the kernel function as follows:

$$K(x, x') := |\langle\psi(x)|\psi(x')\rangle|^2$$

Then,

$$K(x, x') = |\langle\psi(x)|\psi(x')\rangle|^2 = \langle\psi(x)|\psi(x')\rangle\langle\psi(x')|\psi(x)\rangle$$
$$= \langle\psi(x)|\rho(x')|\psi(x)\rangle = Tr[\rho(x)\rho(x')]$$

What is the RKHS that corresponds to this kernel function?

Consider a space of functions defined as follows:

$$f(x) = Tr[\rho(x)M] = \langle\psi(x)|M|\psi(x)\rangle$$

where M is a Hermitian operator acting in the space of d qubits.

$K(x,\cdot) := Tr[\rho(x)\rho(\cdot)]$ is a reproducing kernel for each element of this space:

$$\langle f(\cdot), K(x,\cdot)\rangle = Tr[\rho(x)M] = f(x)$$

Question: what is the dimensionality of this RKHS?

We can write:

$$\langle K_x, K_{x'} \rangle = |\langle \psi(x) | \psi(x') \rangle|^2 = \sum_n \lambda_n \phi_n(x) \phi_n(x')$$

where $K_x \in$ RKHS and $\psi(x)$ is in the Hilbert space of the quantum computer with d qubits.

This kernel function has $\leq 4^d$ non-zero eigenvalues.

What implications does the finiteness of the RKHS have for problems that can be addressed with current-day quantum computers?

Thus, projecting

$$\mathcal{U}^\dagger(\boldsymbol{x}')\mathcal{U}(\boldsymbol{x})|0^n\rangle \quad \text{onto} \quad |0^n\rangle$$

can be another way of building kernels for kernel ML:

$$K(\boldsymbol{x}, \boldsymbol{x}') = |\langle 0^n|\mathcal{U}^\dagger(\boldsymbol{x}')\mathcal{U}(\boldsymbol{x})|0^n\rangle|^2$$

Question: Can such measurements be used to build kernels for ML?

Question: What is the best way to build quantum kernels for ML?

Question: Can quantum kernels outperform classical kernels for ML?

$\Rightarrow$ **Quantum Machine Learning**

What does outperform mean?

$\Rightarrow$ quantum advantage

$=$ kernels that can't be simulated classically?

$=$ kernels that describe big data better?

$=$ relevant in the $\infty\mathcal{D}$ limit for $\boldsymbol{x}$?

A rigorous and robust quantum speed-up in supervised machine learning

Yunchao Liu,[1,2,*] Srinivasan Arunachalam,[2,†] and Kristan Temme[2,‡]

[1] *Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720*

[2] *IBM Quantum, T.J. Watson Research Center, Yorktown Heights, NY 10598*

(Dated: December 1, 2020)

Over the past few years several quantum machine learning algorithms were proposed that promise quantum speed-ups over their classical counterparts. Most of these learning algorithms either assume quantum access to data – making it unclear if quantum speed-ups still exist without making these strong assumptions, or are heuristic in nature with no provable advantage over classical algorithms. In this paper, we establish a rigorous quantum speed-up for supervised classification using a general-purpose quantum learning algorithm that only requires classical access to data. Our quantum classifier is a conventional support vector machine that uses a fault-tolerant quantum computer to estimate a kernel function. Data samples are mapped to a quantum feature space and the kernel entries can be estimated as the transition amplitude of a quantum circuit. We construct a family of datasets and show that no classical learner can classify the data inverse-polynomially better than random guessing, assuming the widely-believed hardness of the discrete logarithm problem. Meanwhile, the quantum classifier achieves high accuracy and is robust against additive errors in the kernel entries that arise from finite sampling statistics.

The Inductive Bias of Quantum Kernels

Jonas M. Kübler* Simon Buchholz* Bernhard Schölkopf
Max Planck Institute for Intelligent Systems
Tübingen, Germany
{jmkuebler, sbuchholz, bs}@tue.mpg.de

June 8, 2021

Abstract

It has been hypothesized that quantum computers may lend themselves well to applications in machine learning. In the present work, we analyze function classes defined via *quantum kernels*. Quantum computers offer the possibility to efficiently compute inner products of exponentially large density operators that are classically hard to compute. However, having an exponentially large feature space renders the problem of generalization hard. Furthermore, being able to evaluate inner products in high dimensional spaces efficiently by itself does not guarantee a quantum advantage, as already classically tractable kernels can correspond to high- or infinite-dimensional reproducing kernel Hilbert spaces (RKHS).

We analyze the spectral properties of quantum kernels and find that we can expect an advantage if their RKHS is low dimensional and contains functions that are hard to compute classically. If the target function is known to lie in this class, this implies a quantum advantage, as the quantum computer can encode this *inductive bias*, whereas there is no classically efficient way to constrain the function class in the same way. However, we show that finding suitable quantum kernels is not easy because the kernel evaluation might require exponentially many measurements.

In conclusion, our message is a somewhat sobering one: we conjecture that quantum machine learning models can offer speed-ups only if we manage to encode knowledge about the problem at hand into quantum circuits, while encoding the same bias into a classical model would be hard. These situations may plausibly occur when learning on data generated by a quantum process, however, they appear to be harder to come by for classical datasets.

What does outperform mean?

$\Rightarrow$ quantum advantage

$=$ kernels that can't be simulated classically?

$=$ kernels that describe big data better?

$=$ relevant in the $\infty\mathcal{D}$ limit for $\boldsymbol{x}$?

Perhaps, there is no practical quantum advantage to present-day QML

$\Rightarrow$ better inference

Building good kernels is not trivial

Quantum kernels offer an alternative to classical kernels

What if we think about this as just another way to do classical ML?

What does outperform mean?

What if we think about this as just another way to do classical ML?

Can quantum kernels offer better inference for practical problems with finite-dimensional $\boldsymbol{x}$?

What does better inference mean?

$\Rightarrow$ More accurate predictions with the same (small) number of training points

$\Rightarrow$ Goal: find the best kernels for each small-data problem

Question: Can QC be used to build useful kernels for ML?

Question: What is the best way to build quantum kernels for ML?

... How does Quantum Machine Learning work in practice?

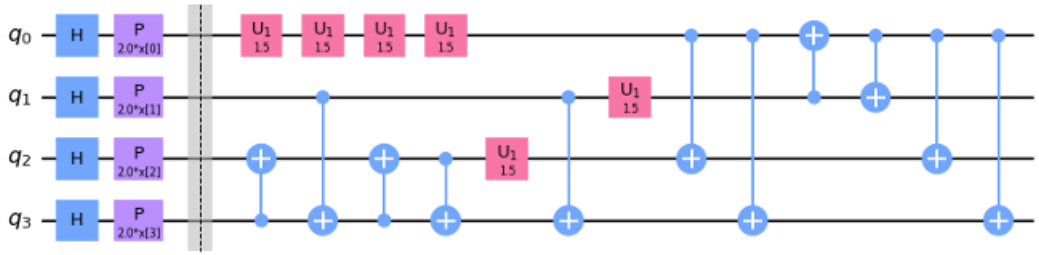
Embedding input space into a quantum computer

We said before, let's embed the input features into quantum states:

$$x \to |\psi(x)\rangle$$

How is this done in practice?

The quantum states are produced by gates acting on qubits:



Gates:

$$R_Z(\lambda) \Rightarrow \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix} \qquad \text{Simplest embedding:} \quad \lambda = x$$

Embedding input space into a quantum computer

$$R_Z(\lambda) \Rightarrow \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix} \qquad \text{Simplest embedding: } \lambda = x$$

Need as many qubits as the number of input dimensions, one qubit per dimension

One possible Strategy:

$\Rightarrow$ Embed using $R_Z(\lambda)$

$\Rightarrow$ Entangle using CNOTs:

$$\text{CNOT} \Rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Embedding input space into a quantum computer

Another Strategy:

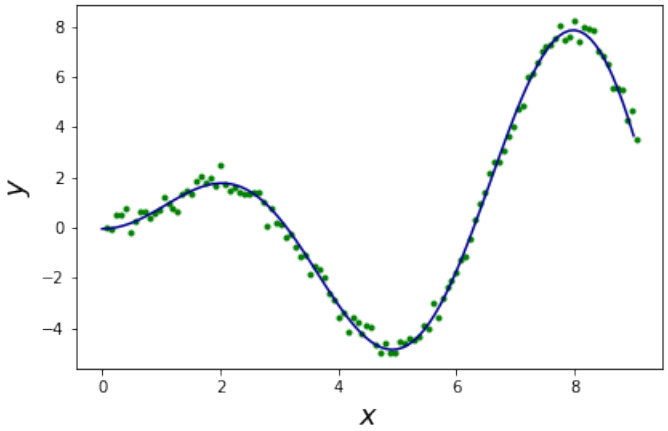
$\Rightarrow$ Embed using a combination of one- and two-qubit gates

$$R_Z(\lambda) \Rightarrow \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix} \qquad \lambda_i = \theta_i x_i$$

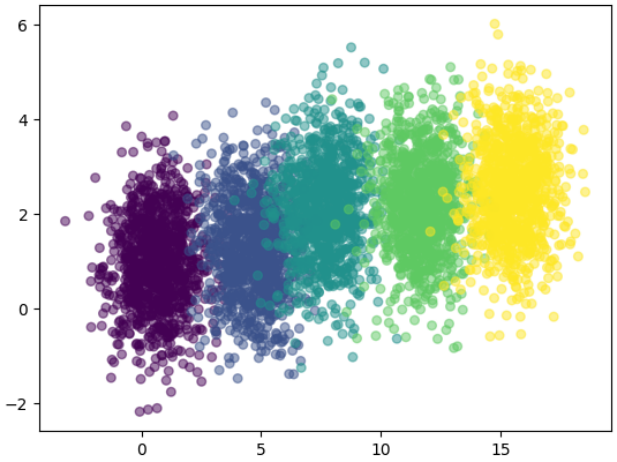
$$R_{ZZ}(\lambda_i j) \Rightarrow \begin{pmatrix} e^{-i\lambda_{ij}} & 0 & 0 & 0 \\ 0 & e^{i\lambda_{ij}} & 0 & 0 \\ 0 & 0 & e^{i\lambda_{ij}} & 0 \\ 0 & 0 & 0 & e^{-i\lambda_{ij}} \end{pmatrix}$$

$$\lambda_{ij} = \exp\left[-(x_i - x_j)^2/\theta_{ij}\right]$$

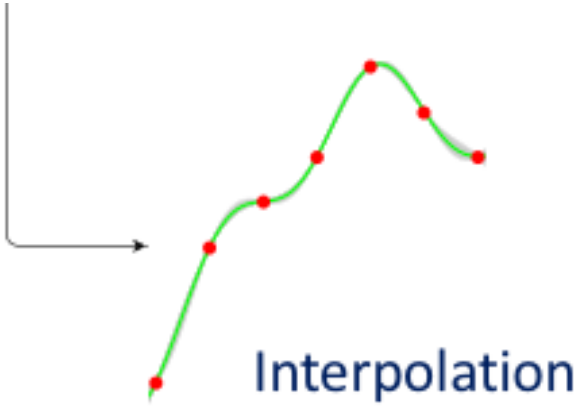
Regression:



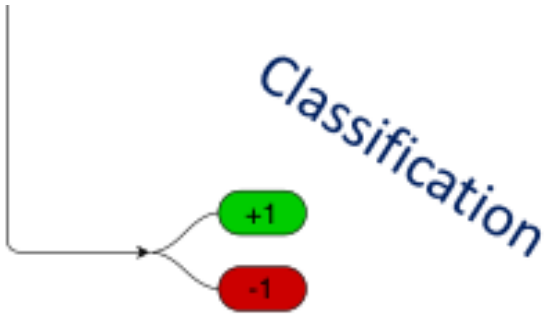
Classification:



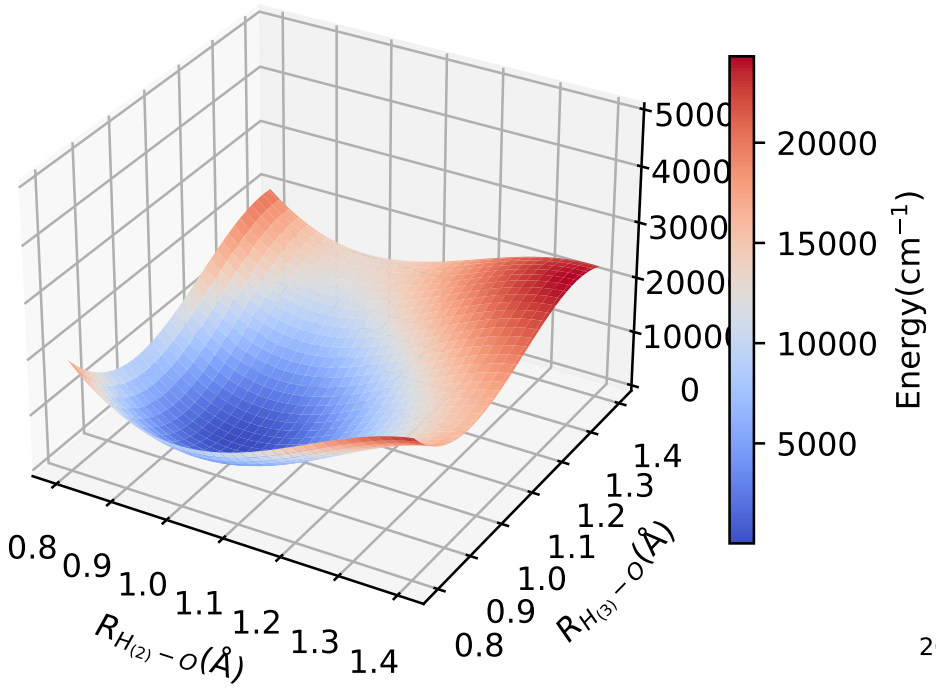
Model



Model

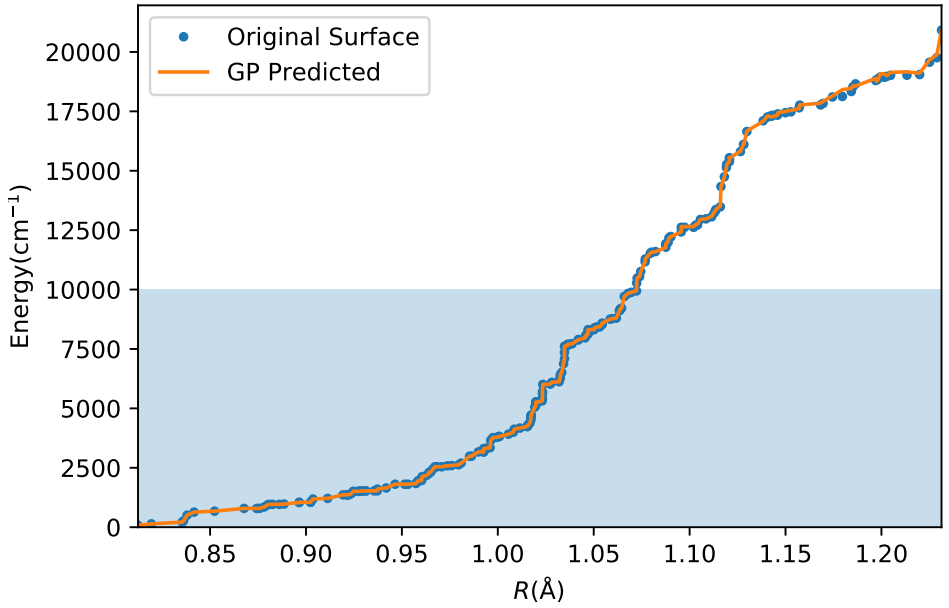


Regression: a 6D PES for molecule H_3O^+ – work of Jun Dai at UBC

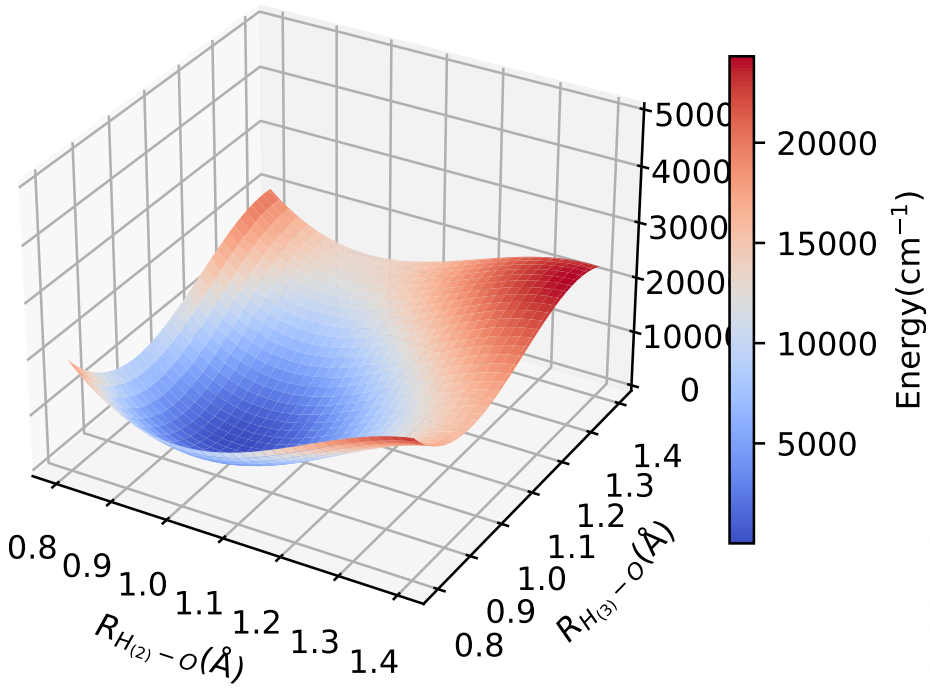


⇐ PES built
with a quantum kernel
6 qubits for 6 dimensions

This is what we
would like to see ⇒

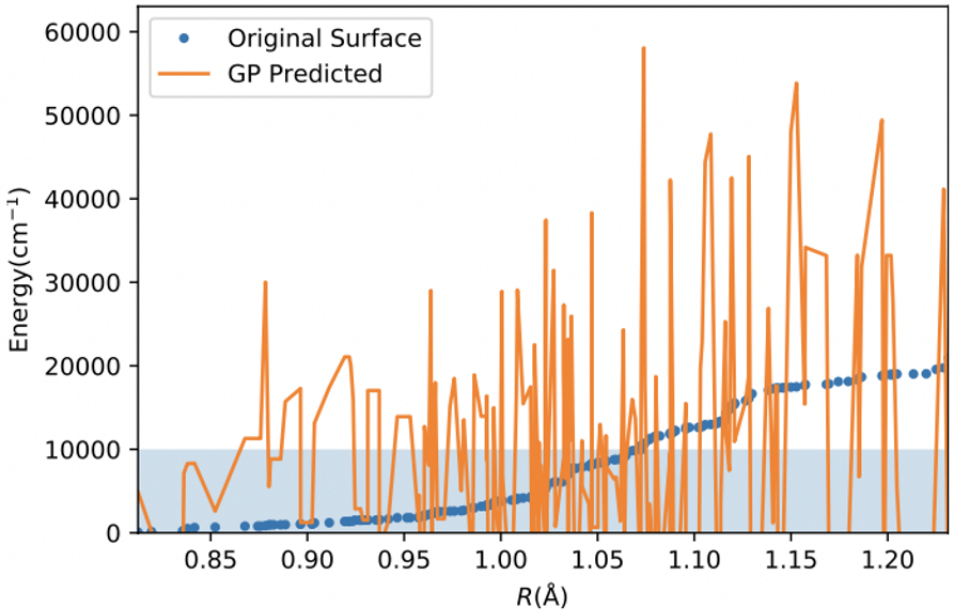


Regression: a 6D PES for molecule H_3O^+ – work of Jun Dai at UBC

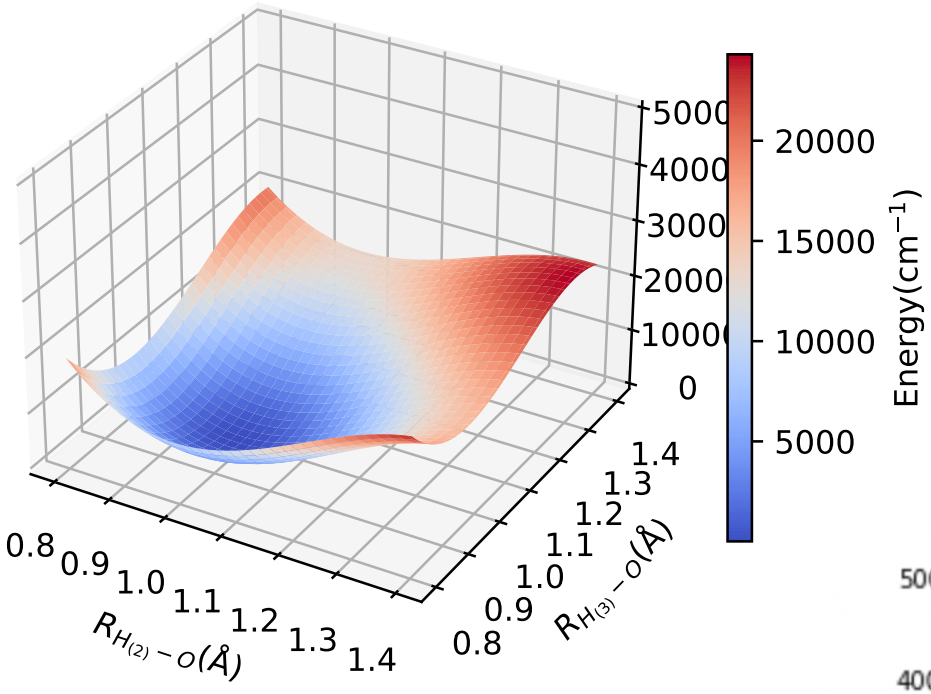


⇐ PES built
with a quantum kernel
6 qubits for 6 dimensions

Jun's 1st attempt ⇒

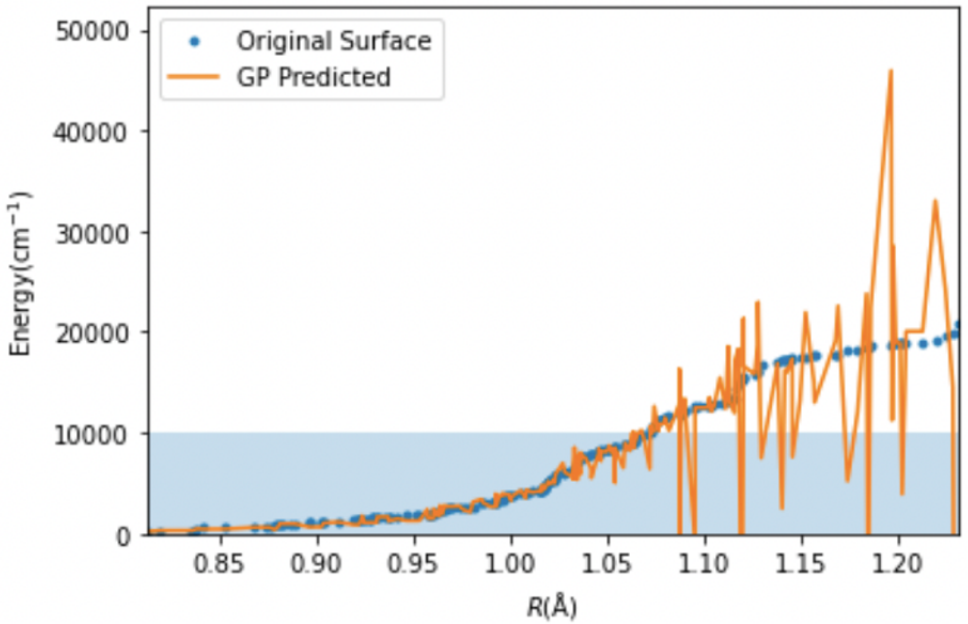


Regression: a 6D PES for molecule H_3O^+ – work of Jun Dai at UBC

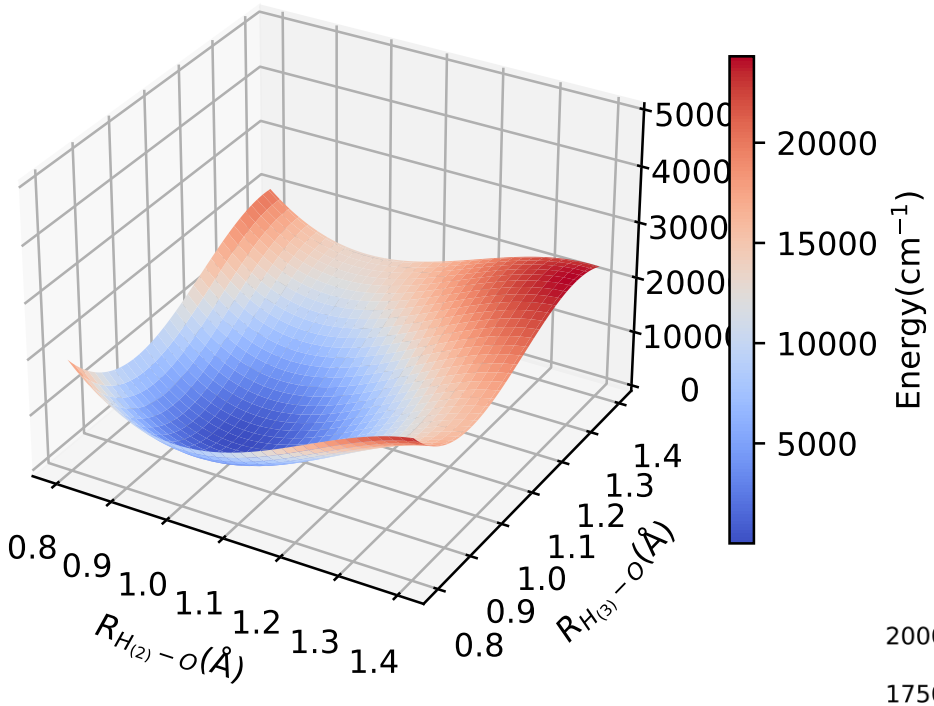


⇐ PES built
with a quantum kernel
6 qubits for 6 dimensions

Jun's 2nd attempt ⇒

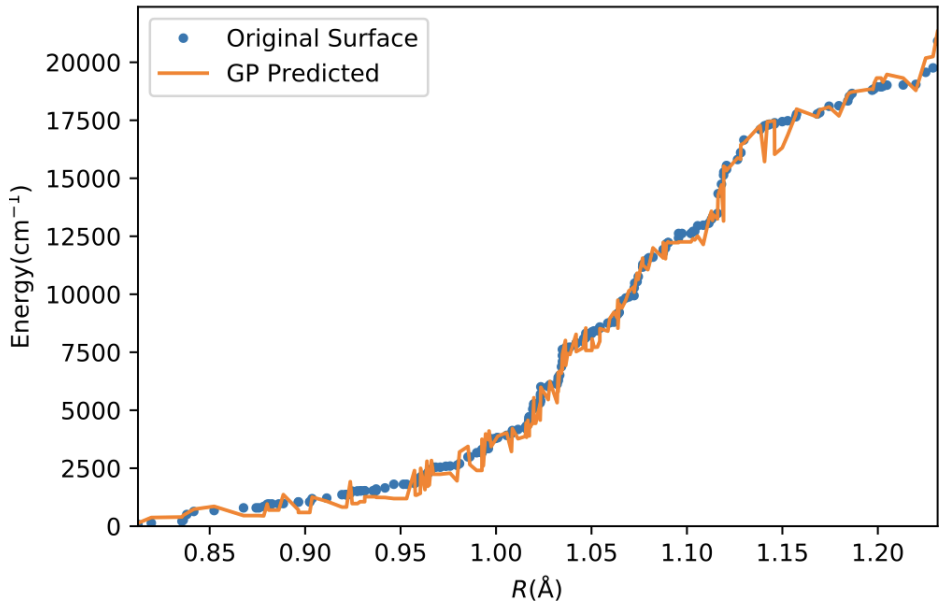


Regression: a 6D PES for molecule H_3O^+ – work of Jun Dai at UBC

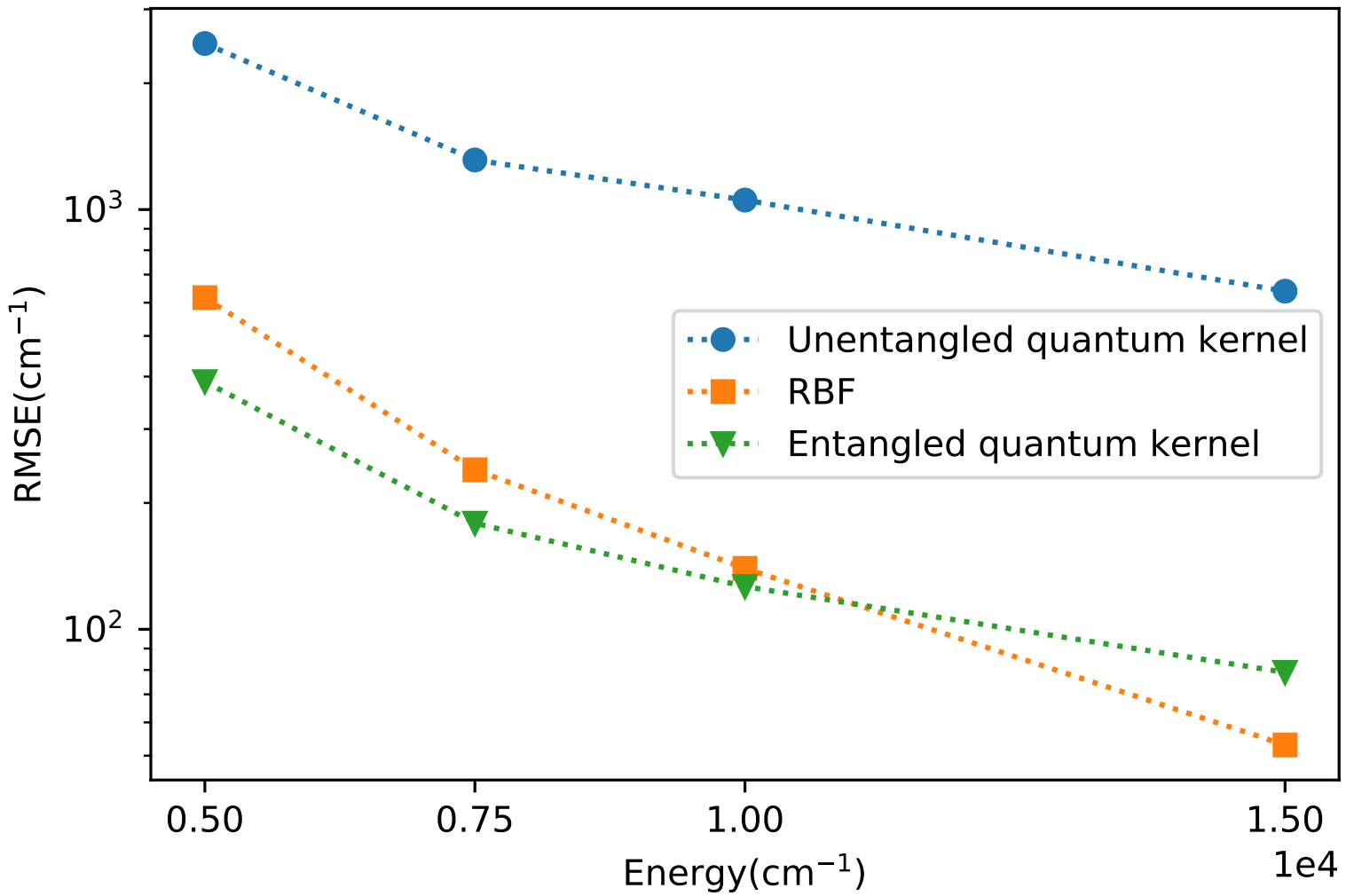


⇐ PES built
with a quantum kernel
6 qubits for 6 dimensions

Jun's 3d attempt ⇒



Regression: a 6D PES for molecule H_3O^+ – work of Jun Dai at UBC



What is the best way to build quantum kernels for ML?

$\Rightarrow$ work of Elham Torabian at UBC

Does the architecture of a quantum circuit matter?

$\Rightarrow$ work of Elham Torabian at UBC

Classification problem – are perovskites metals?

$\Rightarrow$ work of Elham Torabian at UBC

Perovskites are compounds that have the crystal structure of $CaTiO_3$

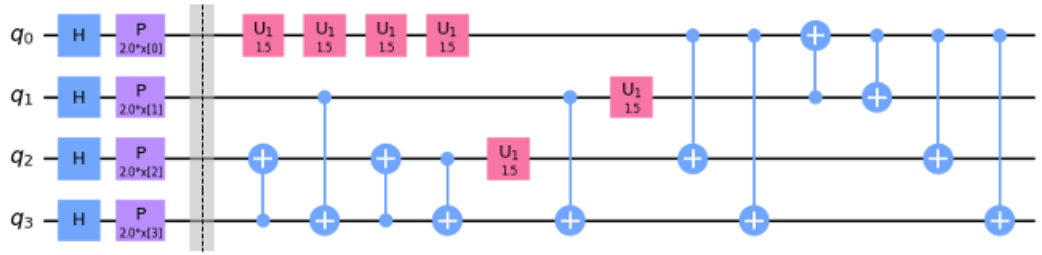
$^{XII}A^{2+}{}^{VI}B^{4+}X_3^{2-}$

Tf	A_i_r	A_M_B_electronegativity	A_e_affinity	A_IP_0	B1_i_r	B1_M_B_electronegativity	B1_e_affinity	B1_IP_2	B2_i_r	...	output_label
0.93754783	1.64	0.8	0.50147	4.34066	0.92	0.9	0.618049	5.39172	0.9	...	1
0.873004911	1.88	0.77	0.471626	3.8939	1.64	0.8	0.50147	4.34066	0.9	...	1
0.962801644	1.72	0.8	0.48592	4.17713	0.92	0.9	0.618049	5.39172	0.9	...	1
0.97114781	1.64	0.8	0.50147	4.34066	0.92	0.9	0.618049	5.39172	0.745	...	1
...	...	...	...	...	...	...	...	...	...	...	...
0.94602682	1.72	0.8	0.48592	4.17713	0.77	1.08	1.235	7.72638	0.69	...	1
0.922000013	1.64	0.8	0.50147	4.34066	0.77	1.08	1.235	7.72638	0.72	...	0
0.92672015	1.64	0.8	0.50147	4.34066	0.77	1.08	1.235	7.72638	0.69	...	1
0.805877252	1.39	0.89	0.547926	5.13908	1.28	1.07	1.302	7.5762	0.62	...	0
0.829579524	1.39	0.89	0.547926	5.13908	1.14	1.49	0	10.4375	0.58	...	0

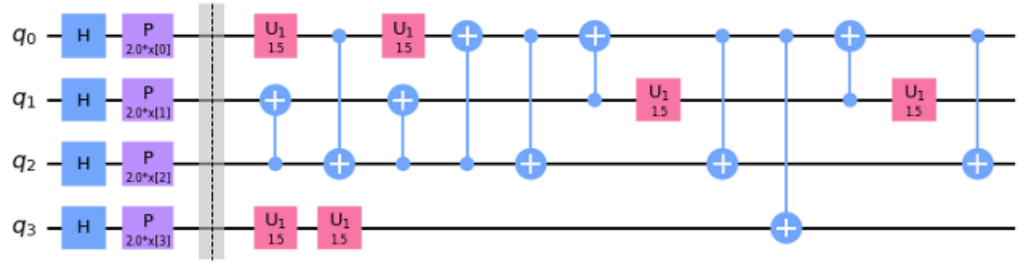
Quantum SVM – not all quantum circuits are created equal...

$\Rightarrow$ work of Elham Torabian at UBC

A good circuit in term of average accuracy



A bad circuit in term of average accuracy

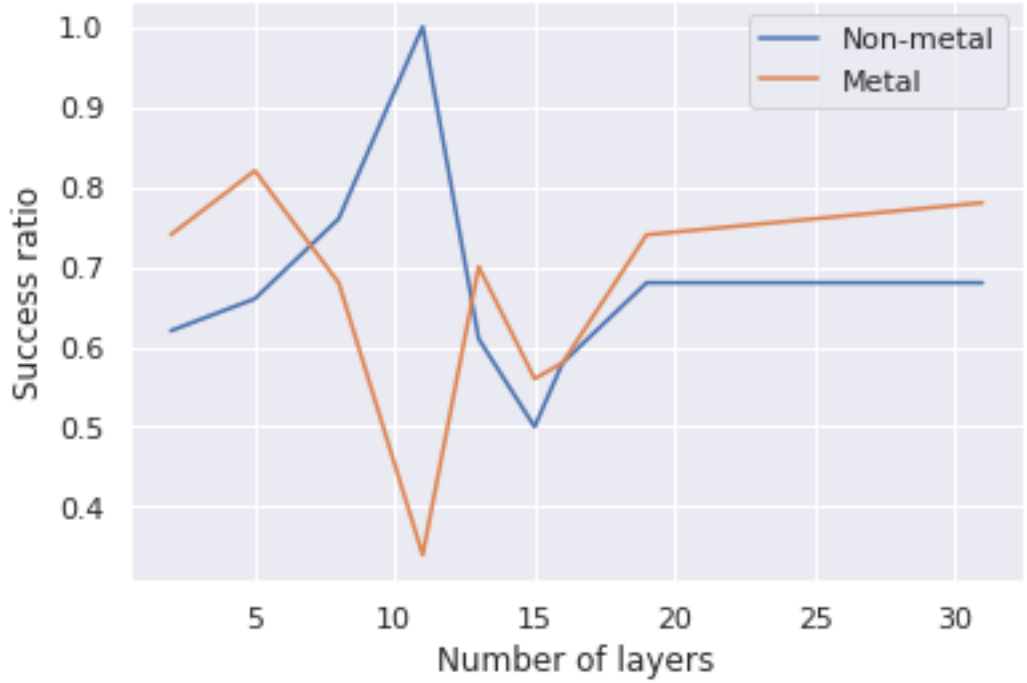


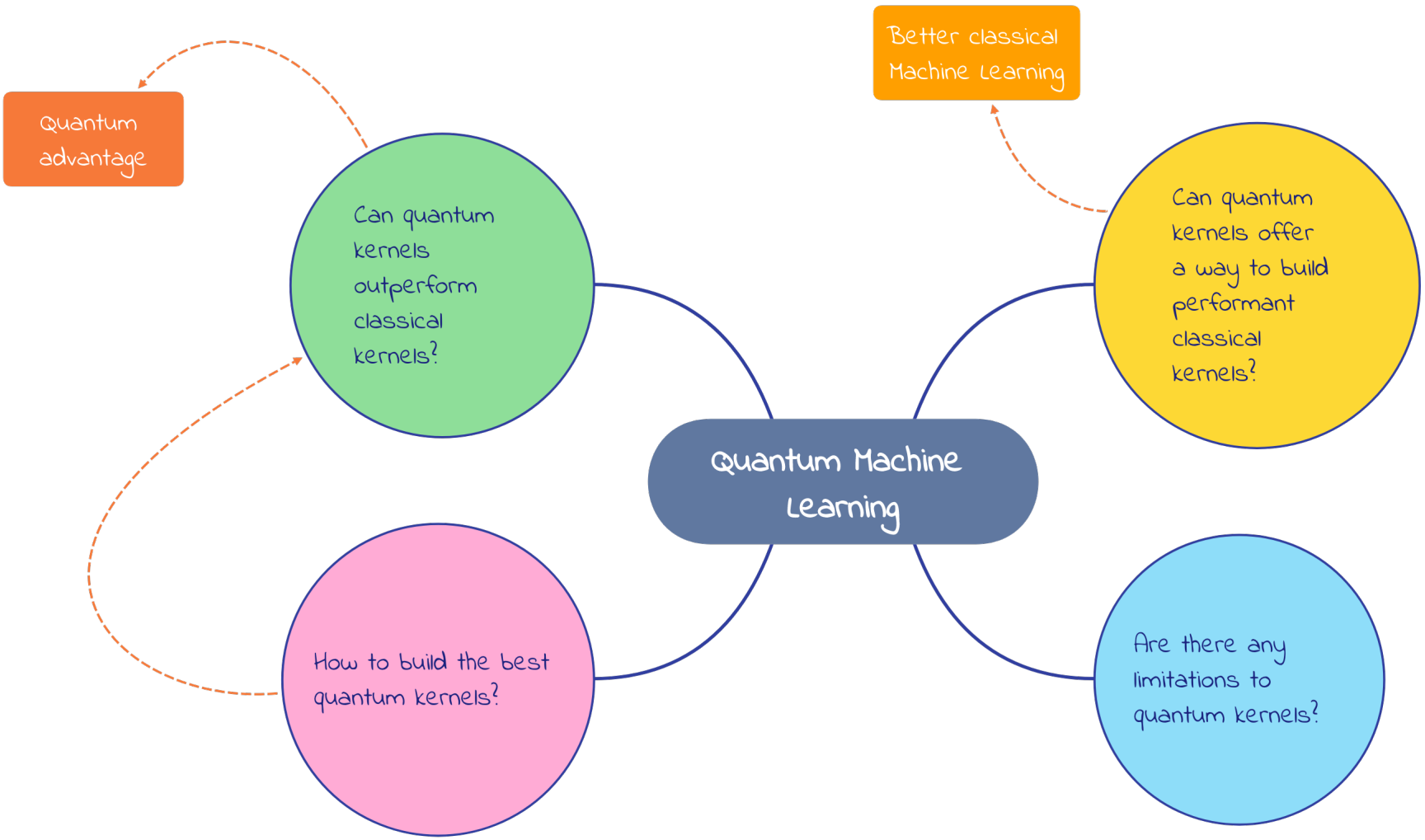
The corresponding accuracies:

Circuit design	Accuracy of class one	Accuracy of class zero	Average accuracy
The good circuit	0.72	0.92	0.82
The bad circuit	0.64	0.54	0.59

Quantum SVM – not all quantum circuits are created equal...

$\Rightarrow$ work of Elham Torabian at UBC




Quantum Machine Learning
Can quantum kernels outperform classical kernels?
Quantum advantage
Can quantum kernels offer a way to build performant classical kernels?
Better classical Machine Learning
How to build the best quantum kernels?
Are there any limitations to quantum kernels?