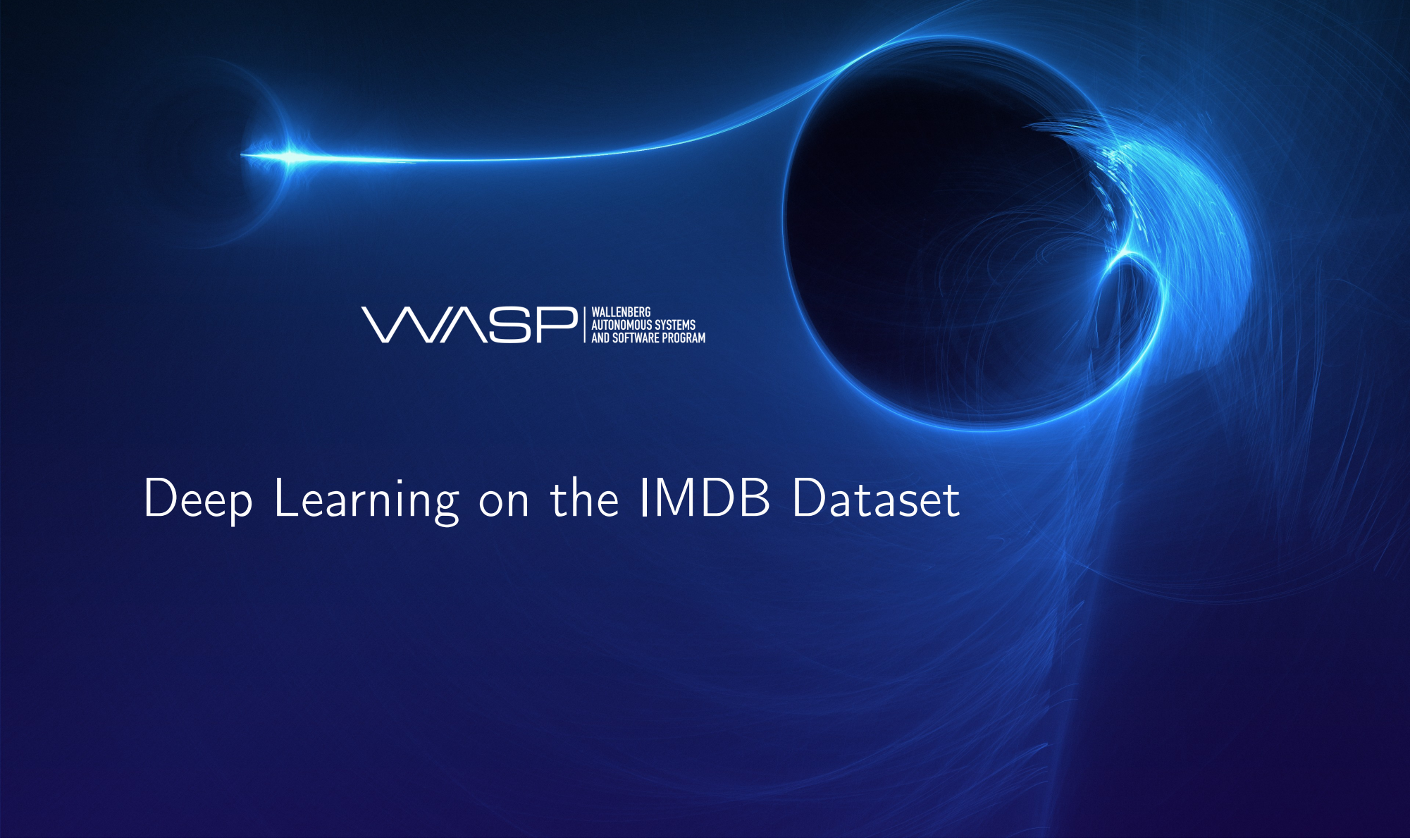


Deep Learning on the IMDB Dataset

The Hitmen — Agents 47

WASP Deep Learning

Gaussian Process
Introduction

Method for performing *non-parametric* regression. Instead of trying to find a function f to fit some dataset X, we can model the *function itself* pointwise as a multivariate Gaussian distribution.

$$\begin{matrix} f(X) \\ f(X_t) \end{matrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_{x_t} \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xx_t} \\ \Sigma_{x_t x} & \Sigma_{x_t x_t} \end{bmatrix} \right) \tag{1}$$

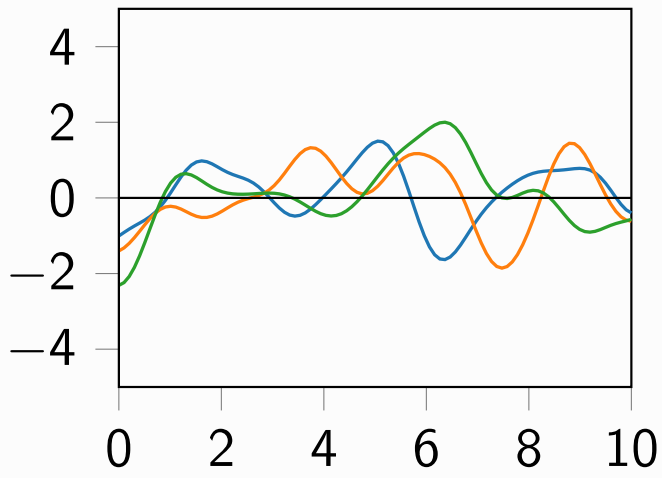
where X_t are the data points corresponding to the sought unknown function values. By designing a prior, μ and Σ can be found and the function search space is reduced.

Problem:
Given a set of 4 data points in the range $[4, 8]$, calculate the GP over the interval $[0, 10]$ in a set of 100 points.
Solution:
As prior, a multivariate Gaussian is created with $\mu = 0$ and the covariance using the Radial Basis Function as a kernel:

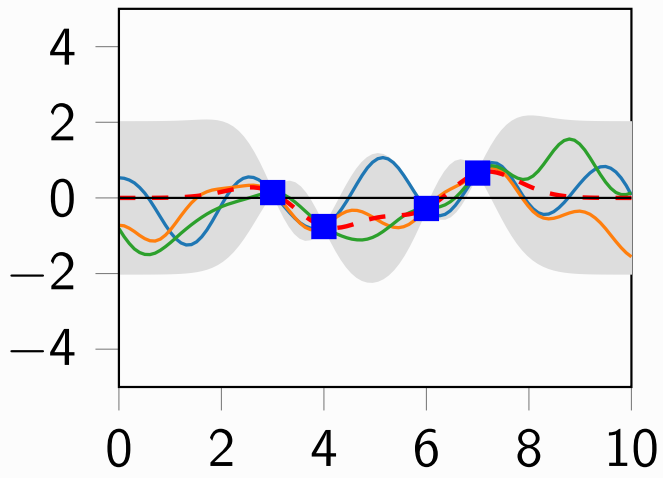
$$\Sigma = \left\{ \Sigma_{ij} = \exp\left(-\frac{1}{2\gamma}(x_i - x_j)^2 \right) \right\}, \tag{2}$$

where γ is a design parameter for smoothness.
The posterior $p(f(X_t)|f(X))$ can then be found.

Gaussian Process
Simple example II



(a) Three samples drawn from the prior.



(b) Three samples drawn from the posterior.

Figure: Samples drawn from both the prior and the posterior. Also shows the data points, posterior mean and standard deviation.

Gaussian Process
GPy

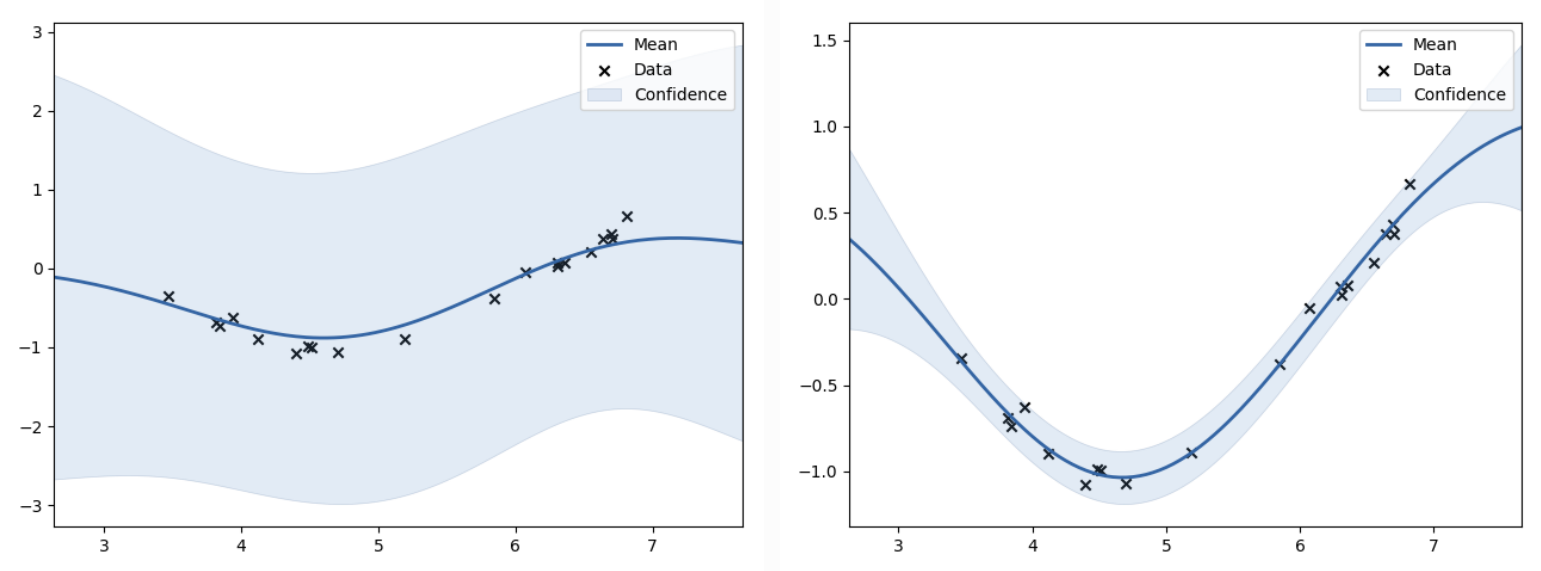
Tool for using GP's in python. Includes a modular, object oriented approach for easy use.

Example:
Instead of hard coding the kernel hyperparameters, the likelihood of the data can be maximized over the hyperparameter.

```
\# Data
n = 20
X = np.random.uniform(3, 7, n).reshape(-1,1)
Y = np.sin(X) + 0.1*np.random.randn(n, 1)

\# Basic regression
kernel = GPy.kern.RBF(input\_dim=1, variance=1,lengthscale=1)
m = GPy.models.GPRegression(X, Y, kernel)
```

Gaussian Process
GPy



(a) No optimization (b) Using `m.optimize()`

Figure

Gaussian Process
Bayesian Optimization

Tuning hyperparameters for classifiers is not trivial. For smaller algorithms, gridsearch or randomsearch could for example be used. For larger models, this becomes unfeasible as the sheer time of computing an evaluation is large. Instead some policy is needed to choose the next configuration of hyperparameters to test.

Consider some loss-metric for the classifier as a function f over the hyperparameters X, we would like to find the set of hyperparameters that minimizes this function. However, this function is unknown and can only be evaluated (slowly) pointwise.

Gaussian Process
Bayesian Optimization

The function f could instead be modeled as a Gaussian process. From the GP a proposal for the optimal choice of X can be generated, each new evaluation will then bring more information about the function.

How to actually choose the proposals can be done in a multitude of ways. Since the model captures uncertainties, proposals can be put in areas of low mean, favoring *Exploitation*, or in areas of high variance, favoring *Exploration*. Usually an *Aquisition function* is designed to balance the choice between the two extremes.

Gaussian Process
GPyOpt

Bayesian optimization tool built on top of GPy for an easy usage.

Example:
Find the minimum value of the function $(6x-2)^2 * \sin 12x - 4$.

```
def f(x):
        return (6*x - 2)**2 * np.sin(12*x-4)

domain = [{ 'name': 'var_1', 'type': 'continuous','domain': (0, 1)}]
Bopt = BayesianOptimization(f=f, domain=domain)
Bopt.run_optimization(max_iter=5)
```

Gaussian Process
GPyOpt

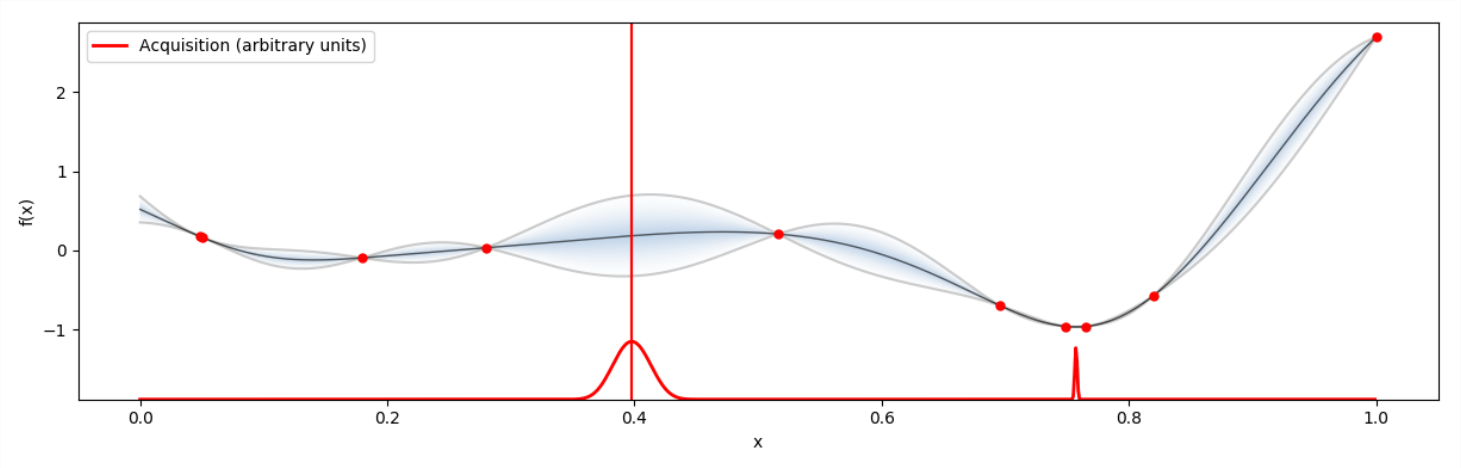


Figure: The Gaussian process and acquisition function after 5 iteration steps.

The End